МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему **Разработка программного обеспечения мультиагентной системы распределения заказов торговой компании ОАО «Приморское»**

Студент Д.О. Кузьмина _____
(И.О. Фамилия) (личная подпись)

Руководитель Э.В. Егорова _____
(И.О. Фамилия) (личная подпись)

Консультанты А.В. Москалюк _____
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н, доцент А.В. Очеповский, _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____»_____2018 г.

Тольятти 2018

АННОТАЦИЯ

Тема выпускной квалификационной работы - Разработка программного обеспечения мультиагентной системы распределения заказов торговой компании ОАО «Приморское».

Ключевые слова: мультиагентная система, автоматизация, распределение заказов, JADE, разработка, программное обеспечение.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка используемой литературы и приложения. Объем работы составляет 69 страниц, 37 рисунков, 10 формул и 5 таблиц. Использовано 22 источника литературы.

Цель исследования – разработать программное обеспечение для мультиагентной системы распределения заказов.

Объектом исследования является распределение заказов в торговой компании. **Предметом исследования** является мультиагентная система распределения заказов, предназначенная для повешения эффективности процесса распределения поставщиков для заказчиков.

Введение к ВКР включает в себя описание цели и актуальности создания мультиагентной системы распределения заказов, а также краткую структуру ВКР.

В первой главе проведен анализ принципов и методов создания мультиагентных систем распределения заказов. Определен подход к разработке. Сформулированы требования к создаваемому программному обеспечению мультиагентной системы.

Во второй главе смоделировано и разработано программное обеспечение мультиагентной системы распределения заказов.

В третьей главе проведено тестирование разработанного программного обеспечения мультиагентной системы распределения заказов и представлен пользовательский интерфейс.

В заключении представлены развернутые выводы по проделанной работе и ее результатам.

ABSTRACT

The title of the graduation work is Development of Software for MAS of Order Distribution for OAO Primorskoye Company.

The object of the graduation work is the distribution of the orders in the trading company. The subject of the work is the multi-agent system of order distribution, which is used for more effective distribution selection for customers. The aim of the graduation work is to model and develop a multi-agent system of order distribution. This system will allow the company to automatize and accelerate the process of distributing orders.

The first part of the work contains analysis of methods, tools and fundamentals of creating a multi-agent system. It also describes advantages and disadvantages of different programming languages and explains the choice of JADE library of Java as the programming language for the software of multi-agent system.

The second part of the work is concentrated on modeling and developing software for multi-agent system of order distribution.

The third part of the work deals with testing of created system.

The result of the work is a working software for multi-agent system of order distribution, which automatically assigns distributors to the customers using communicating agents.

The graduation work consists of an explanatory note on 69 pages, including 37 pictures, 10 formulas, 5 tables and the list of 22 references.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность проблемы.

Проблема автоматизации процессов остро стоит перед человечеством с появления труда. Человек испокон веков пытался облегчить себе труд, используя те или иные средства. Торговые компании не исключение. Был сделан колоссальный прорыв в этом направлении с появлением первых информационных систем. Это поспособствовало улучшению качества обслуживания клиентов и экономии крупных сумм денег торговым компаниям.

В процессе исследования и сравнения разных информационных систем данной области, был выявлено, что все компании достигли пика развития автоматизации торговых процессов, а подходы, используемые на данный момент можно считать устаревшими. Требуется новый способ автоматизации, который реализован в данной работе.

Целью бакалаврской работы является разработка программного обеспечения для мультиагентной системы распределения заказов.

Объект бакалаврской работы: распределение заказов в торговой компании.

Предмет бакалаврской работы: мультиагентная система распределения заказов, предназначенная для повешения эффективности процесса распределения поставщиков для заказчиков.

Для достижения настоящей цели были поставлены следующие задачи:

- изучить используемую организацией информационную систему;
- выбрать подходящее программное обеспечение для разработки;
- спроектировать программный продукт;
- оценить эффективность создаваемого программного продукта с экономической стороны.

Новизна работы заключается в использовании мультиагентного подхода, позволяющего программе самостоятельно распределять заказы между поставщиками и клиентами в реальном времени.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка используемых источников. Введение состоит из актуальности выбранной темы, создания проблемы и выделяются цели и задачи. В первой главе описаны теоретические основы математического моделирования и мультиагентной системы. Так же изучаются основные принципы моделирования и область их применения. Во второй главе представлен выбор языка программирования, математическая модель и разработка мультиагентной системы. В третьей главе представлено тестирование. В заключении сделаны основные выводы и итоги по проделанной работе.

В результате, был разработан, описан и протестирован программный код для мультиагентной системы распределения заказов торговой компании.

В ходе работы над выпускной квалификационной работы была продемонстрирована профессиональная компетентность, направленная на применение методов моделирования при исследовании и проектировании мультиагентной системы распределения заказов.

ГЛАВА 1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПОСТРОЕНИЯ МУЛЬТИАГЕНТНОЙ СИСТЕМЫ

1.1 Принципы работы мультиагентной системы

В традиционной теории искусственного интеллекта решение той или иной проблемы сводится к формированию системы, именуемой агентом, которая обладает всеми нужными знаниями, возможностями и вычислительными ресурсами, может найти решение для какой-то сложной проблемы.

Мультиагентная система (МАС) - система взаимодействия множества агентов для решения непростых задач и проблем [5].

Агент не имеет полного понимания проблемы, следовательно, он способен найти решение только на определенную часть общей проблемы. Поэтому для того, чтобы решить какую-то проблему следует разработать несколько агентов и наладить их продуктивное взаимодействие, которое приведет к возможности создания единой работоспособной мультиагентной системы [1]. В мультиагентных системах каждый агент является частью какой-либо группы или организации, все необходимые задачи делятся между агентами по некоторым правилам. Каждому агенту дается своя роль, сложность роли задается от способностей агента [19].

Особенными чертами агентов считаются:

- коллективность, то есть умение объединиться для решения поставленной проблемы;
- самостоятельность, иными словами, умение без помощи других улаживать проблемы;
- инициативность, способность действовать для достижения определенных целей;
- приспосабливаемость, умение адаптироваться к неясным обстоятельствам в динамичной среде.

Таким образом, любой агент, обладающий конкретной частью знаний о проблеме и может обмениваться этими данными с иными агентами [12].

Систематизацию агентов возможно осуществить в 2-х направленностях:

- по языку программирования агентов;
- по основным приметам, которыми владеют агенты.

Любой агент может создать копию самого себя с абсолютной или узкой функциональностью. Это обеспечивает вероятность настройки среды посредством исключения малоэффективных способов, и замещая их на более новые [8]. Агент, может изменять собственный паттерн поведения, не делая изменений в классе, от которого он наследуется, что влечет нарушение классического устройства ООП. Многозначное наследование дает возможность формировать новые экземпляры агентов, изменяя их паттерн поведения, схемы наследования и свойства. Таким образом, для создания агентов система разработки должна отвечать таким условиям:

- возможность перенесения программного кода на разные платформы;
- поддержка кроссплатформенности;
- наличие сетевых средств общения агентов;
- поддержка многопоточности.

Таким образом, внедрение мультиагентных систем поможет справиться с такими тяжелыми проблемами как распределение заказов, повышение управляемости компании.

Однако при всем разнообразии реализаций мультиагентных систем, все еще очень мало методологических созданных математических моделей и алгоритмов взаимодействия агентов друг с другом [17]. Далее будет рассмотрена математическая модель мультагентной системы, которая даст возможность создать новые взгляды на обработку распределения заказов компании ОАО Приморское.

1.2 Понятие математической модели и математического моделирования в мультиагентных системах

Математическая модель - простое преставление действительности используя математические определения.

Математическое моделирование – процесс исследования и создания математических моделей существующих процессов и явлений. Все без исключения науки, применяющие математический аппарат, пользуются математическим моделированием: замещают существующий объект на его модель и потом изучают. Математическая модель не занимается подробным описанием исследуемого явления, и результаты, полученные в ходе моделирования оказываются крайне полезными и информативными.

Рассмотрим мультиагентную систему, которая обрабатывает поступающие заказы и выдает итог их обработки (рис.1.1). В системе находятся n взаимодействующих агентов a1,a2,a3...an, которые взаимодействуют друг с другом в соотношении с графом потенциальных сетевых взаимодействий [22].

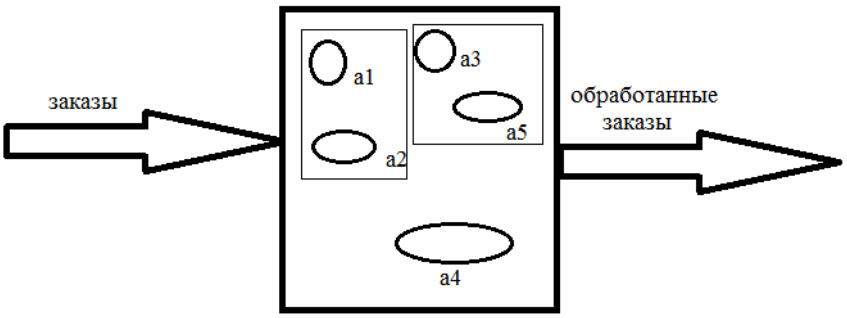

Рисунок 1.1- Мультиагентная система обработки заказов

Система может изменяться в ходе работы. Дальше будет рассматриваться система заказов торговой компании ОАО Приморское.

Агент - сущность, которая рационально мыслит и наблюдает за окружающей его средой, в которой он находится и все его действия нацелены на достижения какого-либо результата [9].

У каждого агента есть ментальные категории, такие как взгляды, стремления и планы:

- взгляды - понятия о текущем пребывание в среде;
- стремления - состояние среды, к которому агент желает привести свою среду;
- планы - множество избранных, совместимых и достижимых стремлений.

Распределение стремлений и планов нужно, потому что агенты могут обладать несовместимыми стремлениями или стремления могут быть не реальными. Так как у агента недостаточно ресурсов, он не может достигнуть абсолютно всех стремлений сразу, поэтому агент избирает более важные цели - планы.

Отметим, $A = \{a_i\}_{i=1}^{n}$ как множество всех существующих агентов.

Можно отметить 3 критерия, определяющие агентов:

- чувствуют среду в которой находятся;
- хранят знания;
- выполняют действия.

Отметим, $R = \{r_i\}_{i=1}^{q}$ как множество всех возможных разнообразных способностей, свойственных агентам из множества A.

Для всех агентов установим функцию: $ir_a\ a : A \rightarrow 2^q$, которая создает граф сетевых взаимосвязей ресурсов с агентами. Учёт этих, предварительно установленных взаимосвязей даст возможность уменьшить количество вычислений при математическом решении класса установленных проблем.

Рассмотрим деятельность системы на определенном промежутке времени [0,T]. Реализуемость агентов создает ограничения на допустимое количество задействованных в определенных период времени $t \in [0, T]$ ресурсов.

$$\forall a \in A, \forall i \in ir_a\ a \quad x_i\ t\ \leq V_i\ a\ , \tag{1.1}$$

где $x_i\ t$ - введенный в период времени t объем i - ресурса, ед.,

$V_i\ a$ - max объем i-го ресурса для агента a, ед.

Применение ресурсов безусловно приводит к расходам, определяемым длительностью и размером используемого ресурса. Для каждого ресурса

$\forall i \in ir_a$ a произведенные на интервале [0,T] затраты естественно определять функционалом

$$L_i\ a\ = \int_0^T l_i\ x_i\ t\ ,a\ dt, \tag{1.2}$$

где l_i - некая функция, определяемая типом ресурса i, в обычных случаях $l_i = 0$, **при** $\mathrm{x} = 0$ и $l_i = 1$ **или** $l_i = \mathrm{x}$, в остальных случаях.

Стремления и планы как правило формулируются математически в виде проблем оптимизации этих либо других функционалов качества, либо свершения тех или иных установленных множеств. Стремления и планы имеют все шансы быть как у единичных агентов, так и у групп агентов.

В систему поступает большое число заказов разных типов. Отметим $z = \{z_i\}_{i=1}^{m}$ - множество различных типов заказов.

Рассмотрим ситуацию, в которой для $\forall z \in Z$ имеется агент или группа агентов, у которых имеются все нужные ресурсы для выполнения заказа.

Функция $iz_z Z: Z \to 2^q$ определяет набор ресурсов, который нужен для выполнения заказа z и $\{v_i\ z\ ,d_i\ z : i \in iz_z(z)\}$ - набор количества ресурсов и длительность их задействования при выполнении индивидуального количества заказов. У любого заказа имеется время поступления и максимальное время на исполнение, количество максимального времени на обработку заказа:

$$\mathrm{z} = \mathrm{z}(t_c, t_{max}, V, t_f), \tag{1.3}$$

где t_c - время поступления заказа, с.,

t_{max} —количество максимального времени на выполнение заказа, с.,

V — объем заказа, ед.,

t_f — максимальное время на обработку заказа, с.

Объем заказов определяется по формуле (1.4)

$$V = \sum_{i \in ir_z\ z} v_i\ z\ \ d_i\ z\ , \tag{1.4}$$

где $v_i\ z$ - количество i-го ресурса, ед.,

$d_i\ z$ - длительность задействования i-го ресурса, с.

У каждого заказа имеется определенная группа агентов, которая может его исполнить, у него есть следующие ограничения:

$$\forall a \in A \qquad \sum_{i \in ir_z} x_i v_i(a) \geq V_{min}, \tag{1.5}$$

где V_{min} - минимальный объем ресурсов, для выполнения заказа z, ед.,

x_i - объем i-го ресурса, ед.

За осуществление любого стандартного заказа установлена цена. Полагаем, что объем оплаты покупателем заказа пропорционален цене и размеру. Отметим, S_i - сумма оплаты i-го заказа.

У любого агента имеются собственные функции цены за использование тем либо другим ресурсом, а у любого заказа имеется своя максимальная цена. Кроме того, определяется цена взаимодействия агентов друг с другом.

Математическое моделирование является неотъемлемой частью построения мультиагентных систем. Далее будут рассмотрены способы построения математических моделей для их последующего применения в мультиагентных системах.

1.3 Задачи мультиагентных систем

В мультиагентной системе могут быть разобраны следующие задачи:

- расчет потенциального плана заказов;
- расчет наилучшего плана заказов. Следует создать такой план заказов, чтобы прибыль будет наибольшей:

$$f = \sum_{i=1}^{m} S_i - \sum_{a \in A} \sum_{i \in ir_a} L_i(a) \to max, \tag{1.6}$$

где S_i - сумма оплаты i-го заказа, ед.,

$L_i(a)$ - затраты агента a на выполнения i-го заказа, ед.

Как правило решение проблем оптимального управления сложно и не всегда корректно. Поэтому следует поставить задачу о нахождении почти оптимального уровня рентабельности.

Отметим C как сумму всех цен оплаты заказов, а D как сумму затрат каждого агента на выполнение каждого заказа:

$$C := \sum_{i=1}^{m} S_i, \tag{1.7}$$

$$D := \sum_{a \in A} \sum_{i \in ir_a}^{n} L_i(a). \tag{1.8}$$

- достижения данного уровня рентабельности. Уровень рентабельности задается отношением доходов к затратам:

$$P = \frac{C}{D} * 100\% \geq p_{fix} \tag{1.9}$$

где p_{fix} - заданный неизменимый уровень рентабельности, процентов.

- достижение максимального уровня рентабельности:

$$P = \frac{C}{D} * 100\% \rightarrow max \tag{1.10}$$

Решение такой задачи как правило обладает огромной стабильностью, и подобного рода постановки целесообразнее применять при адаптивном подборе стратегии управления мультиагентной системой.

1.4 Выбор языка программирования для создания программного обеспечения мультиагентной системы

На сегодняшний день ни один и разработанных языков программирования, полностью соответствующего всем необходимым требованиям технологии мультиагентных систем. Многоагентные системы, создающиеся на данный момент, в основном, применяют базовые языки, однако, их нельзя рассматривать как «агентно-ориентированные» [3]. Специальные языки, для работы с мультиагентными системами набирают популярность, и новые языки разрабатываются все чаще и чаще.

Перед началом анализа и сравнения агентских языков, выделим все основные требования. Более значимыми являются следующие:

Прежде чем приступать к сравнению языков, необходимо указать критерии, по которым язык будет выбираться:

- возможность переноса кода на другие платформы;
- возможность кроссплатформенного запуска;
- поддержка многопоточности;
- методы для создания общения агентов;
- способность выполнения вычисления символов;
- наличие функционала, ограничивающего возможность несанкционированного доступа;
- использование принципов ООП;

Java - один из популярнейших языков, из тех которые используют с целью разработки агентов. Синтаксис Java похож на C++, однако по идеологии скорее напоминающий Objective C. В язык Java встроена виртуальная машина, а также переводчик с языка Java в байт-код.

Язык Java учитывает разработку приложений, возможных для запуска и дополнения кода приложения на нескольких платформах. Программа, разработанная на Java, преобразуется в bytecode. Полученный код, используя интерпретатор, можно исполнять на любом устройстве, которое может использовать Java VM. Таким образом гарантируется кроссплатформенность Java – приложений. Байт-код приложения может быть получен из любых источников, в том числе через Web-страницу содержащую ссылку на данный код. Java VM функционирует в среде, которая вытесняет мультизадачности и использует потоки (threads). Средства для разработки потоковых процессов содержатся в Java в качестве классов.

Java является сильно типизированным объектно-ориентированным языком программирования. Сходство с C++ делает его крайне легким для изучения. Все данные в Java, разделяются на типы, которые представляют собой классы, а все функции языка считают методами этого класса. Вызов функций считается использованием метода объекта, к которому функция обращается [7]. Язык Java обладает поддержкой разработки дополнительных библиотек, а также использует Abstract Window Toolkit (AWT) для более

детальной разработки приложения. Важной частью также является наличие средств для обработки исключений (exceptions), а также контроль запущенной программы в реальном времени.

Язык Java используется для разработки двух видов приложений: автономных, включающих машинно-зависимый код, а также апплетов - приходящих из сети приложений и запускаемые в браузерах с поддержкой Java Web. Апплеты встраиваются в Web-страницы при помощи тега языка HTML, который создает и показывает ссылку, содержащую код, и запускаются сразу, вместе с открытием страницы в браузере [14].

Python - объектно-ориентированный язык программирования, переносимый и интерпретируемый, который был в CWI (Амстердаме). Язык содержит изящный (но не слишком упрощенный) синтаксис, в него включено некоторое число мощных типов данных. Python может расширяться, с помощью добавления различных модулей, которые выполнены на компилируемом языке, таких как C/C++ [2]. Подобные расширения могут определять как новые переменные и функции, так и новые типы объектов.

TeleScript. Одна из самых ранних рыночных разработок мобильного агента выполнена в среде TeleScript от General Magic. Эта технология создана на основе общедоступной сети (public network), позволяющая владельцам товара и клиентам налаживать связи для совершения бизнес-сделки.

TeleScript-технология оперирует такими понятиями: агенты, места, встречи, передвижения, возможности, соединения и разрешения. Ниже приведены пояснения:

1. Агенты. Приложение объясняется в качестве группы агентов, где каждый агент имеет свое конкретное место. Тем не менее, агент имеет возможность передвигаться на другие места, и по этой причине он имеет право находиться сразу в нескольких местах единовременно. Все агентские процедуры производятся параллельно. В модели электронного рынка на обычном месте постоянно имеется один, отведенный агент.

2. Места. TeleScript-технология анализирует всю сеть устройств как множество мест. Место - стационарная процедура на сервере, которая предлагает свои услуги агенту.

3. Встречи. Встреча дает агентам возможность вызывать процедуры взаимодействующего агента. Встречи является способом воздействия на агента с целью смены его места расположения. Для нахождения соседнего (co-located) агента, исполняется инструкция **meet**. Данная инструкция имеет требования (*petition*) - данные, подбирающие агента для создания встречи. **Meet**-инструкция используется для проведения сделок между продавцами и покупателями.

4. Передвижения. У агента есть способность менять свое расположение. Передвижение - характерный критерий, который наделяет агента умением приобрести что-либо, а после возвратиться на исходное местоположение. TeleScript также разрешает вычислительному пакету (computer package) – агенту двигаться от устройства к устройству. Чтобы выполнить передвижение агент исполняет инструкцию go. Она содержит ticket - сведения о адресе назначения пакета, и иных параметрах передвижения. При успешном передвижении, агент о этом информируется (его последующая инструкция производится уже на другом месте). В модели электронного рынка go дает способность агентам клиентов и владельцев товара менять расположение связанного агента с целью наиболее успешного взаимодействия.

5. Возможности. Данная технология наделяет единицам системы видеть возможности всех остальных единиц. Единицы не имею способности не открывать другим своих возможностей, равно как и называть ложные возможности. Технологией также используется учет возможностей при смене агентами текущего сетевого региона (network regions) - группой расположений, которые находятся на устройствах, и которые обладают одинаковыми возможностями. С целью выяснения возможностей другой единицы системы выполняется инструкция name. Итогом выполнения инструкции считается telename - значения, содержащие возможности единиц, находящихся пределах

единого сетевого региона. Такая возможность позволяет защищать единицы системы от попадания вирусов.

6. Соединения. Они дают возможность агентам меняться данными из любого месторасположения. Для создания связи агент осуществляет connect-инструкцию. Эта инструкция включает ряд параметров, например, цель (target) соединения. Благодаря данной инструкции, агенты имеют возможность передачи данных друг другу, находясь на приличной дистанции друг от друга.

7. Разрешения. Технология, позволяющая управлять назначением возможностей.

Язык программирования дает возможность программисту, который занимается разработкой программного обеспечения, устанавливать правила и порядки поведения агентов, и информацию, которую агенты разносят за период передвижения по сети. Он содержит в себе возможности, которые предлагают C/C++. Программное обеспечение возможно разработать полностью с использованием одного лишь TeleScript, но в основном программисты создают элементы мультиагентной системы при помощи TeleScript, а интерфейсы с пользователем, базами данных и так далее - на C или C++.

Telescript имеет данные отличительные черты:

- полноразмерность;
- чертами ООП;
- изменяемостью (dynamic). Агент способен передавать данные по мере смены своего расположения в сети. В том числе если при отправке элемента по адресу направления не определен, то его описательный класс также подлежит отправке по этому же адресу;
- постоянством (persistency). На всех этапах исполнения агент и данные, которые он переносит, хранится в памяти без возможности быть потерянными. Данная процедура решает проблему потери данных при крушении системы устройства;

- переносимость и безопасность. Устройство исполняет указания, присланные вместе с агентом с помощью engine-интерпретатора. Агент способен исполняться на всех устройствах с имеющимся интерпретатором;
- направленность на контакт (communication-centric). В данный язык программирования встроены указания, которые позволяют агенту осуществлять тяжелые коммуникационные задачи.

AgentSpeak. Подобием класса в этом языке будет считаться семейство, а членом семейства будет агент. Каждый агент владеет БД взаимоотношений с частной и общедоступными частями, группой услуг и группой намерений - процедур, известных только в случае их исполнения. Язык обеспечивает сохранение информации в пространстве функционирования агентов.

Любая услуга имеет один из последующих типов: query, achieve, told. Любое намерение устанавливается, используя имя и абстракцию. Если агент функционирует и его список намерений применяется в указанном случае, то производятся действия, которые связаны с данным списком намерений.

Передача сообщений может быть синхронная или асинхронная. Всем агентам дается индивидуальный почтовый ящик, где хранятся все поступающие на него послания. Кроме обмена посланиями типа агент-агент, также имеется связь агент-семейство агентов, что дает возможность агентам пользоваться сервисами обработки поступившей информации используя настройки семейства. Послания распределяются по пулу посланий, прикрепленных к своему семейству агентов. Послания, применяемые в данном языке программирования, обладают одним из следующих типов: оповещение, запрос с ожиданием ответа от цели и запрос без ожидания ответа от цели.

Всякий агент способен пребывать в одном из следующих состояний: готов, в ожидании, отключен.

Когда к агенту приходит послание, оно сохраняется в его личный почтовый ящик, после чего происходит обработка сообщения. Для начала выбираются намерения по вариации сообщения (achieve, query или told),

имеющие все шансы быть применёнными в указанном условии (множество подходящих намерений), дальше просматриваются все абстракции множества намерений, которые были взяты ранее (множество готовых к применению намерений). После для крайнего множества идет построение реализации. Изначально система делает выбор одного намерения из группы, возможных к применению и приступает к действию, отмеченному в свойствах этого намерения. Во время исполнения агент способен обладать множеством намерений – потоками, в виду чего агент можно назвать многопоточным процессом. Приватные и общедоступные сервисы агента действуют, соперничая за ресурсы, из-за того, что агент способен исполнять как приватные, так и общедоступные сервисы.

Этот язык схож с языками программирования Agent0 и PLACA, но у них есть различие в том, что здесь текущее положение агента подразумевается в качетсве взаимоотношения взглядов, целей и планов. Agent0 анализирует текущее положение, которое состоит из множеств возможностей, взглядов и целей.

Agent0 и PLACA объясняют агентно-ориентированное программирование подвидом ООП, однако разработчики AgentSpeak полагают то, что первое на самом деле считается масштабным расширением второго.

В таблицах 1.1 и 1.2 приведены сравнения языков программирования, которые могут использоваться для разработки программного обеспечения мультиагентной системы распределения заказов торговой компании.

Таблица 1.1 - Сравнение языков

Язык	**Переносимость кода**	**Доступность**	**Сетевые возможности**	**Параллельность**
Java	байт-код, виртуальная машина Java	Windows, Solaris SPARC /Intel, HP-UX, OS/2, Macintosh, Linux	APIs (библиотеки классов), Remote Method Invocation, сетевая сериализация	Threads синхронизованные с помощью мониторов
Python	интерпретируемый язык	Windows, DOS, Macintosh, UNIX	интерфейс к TCP/IP	

Язык	Переносимость кода	Доступность	Сетевые возможности	Параллельность
TeleScript	интерпретация скриптов	Solaris SPARC, HP-UX, OS IRIX	TCP/IP, UDP, сетевая сериализация	Множественные процессы, работающие в вытесняющей многозадачности
AgentSpeak	интерпретируемый	пока не реализован		аналог потоков - намерения

Таблица 1.2 - Сравнение языков по дополнительным параметрам

Язык	Символьные вычисления	Обеспечение безопасности	Объектная ориентация	Встроенные агентские свойства
Java	не поддерживаются	Есть	есть, без множественного наследования	Нет имеется сторонняя библиотека JADE, реализующая необходимый функционал взаимодействия агентов
TeleScript	не поддерживаются	встроенные в язык и библиотеку классов средства	да	агенты, места, маршруты, встречи, соединения, авторизация, полномочия, инструкции: go, meet, connect, permit, name
AgentSpeak			агентно-ориентированный язык	BDI (beliefs-desires-intentions) архитектура
Python			да	

Отталкиваясь от предоставленной информации, для разработки программного обеспечения мультиагентной системы будет использоваться язык java с подключением библиотеки JADE для работы с агентами.

Исходя из полученных сведений, можно сделать вывод, что существует верный и действенный способ автоматизировать процесс распределения заказов

в торговых компаниях, который в настоящее время остро нуждается в пересмотре методов распределения заказов [13]. Технология мультиагентных систем позволяет упростить и ускорить процесс назначения заказов по заказчикам путем использования агентов, взаимодействующих между собой по определенным правилам и условиям, что влечет за собой повышенную скорость обработки заказов, а, следовательно, и качество обслуживания клиентов.

ГЛАВА 2 РАЗРАБОТКА АГЕНТНОГО ПРИЛОЖЕНИЯ ДЛЯ ТОРГОВОЙ КОМПАНИИ НА ОСНОВЕ ПЛАТФОРМЫ JADE

Важные этапы создания мультиагентного приложения с помощью платформы JADE рассмотрим на примере приложения «Торговой компании». В данном приложении действуют агенты-продавцы и агенты-покупатели, которые делают заказы по поручению пользователя приложения. Любой агент-покупатель получает название заказа, который ему нужно сделать («необходимый заказ»), в качестве аргумента командной строки и время от времени запрашивает предложения у каждого известного ему агентов-продавцов. Сразу после того, как предложение получено, агент-покупатель принимает его и выдает заказ на покупку. В случае если несколько агентов-продавцов дают предложение агенту-покупателю, он выбирает среди них самое лучшее. После покупки агент-покупатель заканчивает свою работу. Любой агент-продавец создается, получая информацию из базы данных, в которой администратор базы данных может добавить новых заказов и информацию о них. Агент-продавец постоянно проводит проверку на поступление запросов от агентов-покупателей. При получении запроса на заказ каждый агент-продавец проверяет свой локальный журнал, если запрашиваемый заказ располагается в его журнале, отвечает на запрос, сообщая агенту-покупателю информацию: стоимость, доступное количество файлов, время доставки заказа. Сразу после исполнения, заказ автоматически удаляется из журнала.

2.1 Проектирование программного обеспечения

2.1.1 Разработка логической модели программного обеспечения

UML (Unified Modeling Language) был создан для специфицирования (формирования спецификации), записи компонентов программ их функций, бизнес-процессов, визуализации, построения.

UML считается легким и действенным способом для моделирования, его с легкостью можно действенно применять для проектирования разнообразных моделей, комплексных систем самого разного целевого применения. UML-

моделью называется графическое обозначение с целью создания абстрактной модели системы. Этот язык не является языком программирования, однако в средствах исполнения UML-моделей как интерпретируемого кода имеется возможность генерация кода [4].

Конструктивное применение языка UML полагается на представление определений моделирования комплексных систем индивидуальности процесса объектно-ориентированного анализа, создания проекта:

- основа абстрагирования - в модель включаются только детали, которые имеют определенное отношение к исполнению проектируемой системой собственных функций либо собственных прямых задач;
- основа многомодельности подтверждает то, что никакая единичная модель не сможет с необходимой точности делать описания различных деталей непростой системы. Общим понятием непростой системы являются: меняющееся представление (изложение процессов логики), разделенные на различные более отдельные представления и структурное;
- основа иерархического выстраивание моделей комплексных систем указывает рассматривать развитие проектировки модели на различных ступенях детализации или абстрагирования в установленных границах представлений.

UML может применяться с целью создания логических и концептуальных моделей комплексных систем различной целенаправленной задачи [6]. В создание включено проектирование модели, заключенной в графической аннотации. Поэтому обязательное соблюдение единых основ структурного проектирования: нисходящая разработка, точная формализация и строгая семантика, иерархическое создание модели.

Представление о модели сложной системы, отображающихся на UML языке в виде специальных графических диаграмм:

- диаграмма вариантов использования (use case diagram);
- диаграммы поведения (behavior diagrams);

- диаграмма деятельности (activity diagram);
- диаграмма кооперации (collaboration diagram);
- диаграммы взаимодействия (interaction diagrams);
- диаграммы реализации (implementation diagrams);
- диаграмма последовательности (sequence diagram);
- диаграмма компонентов (component diagram);
- диаграмма классов (class diagram);
- диаграмма состояний (statechart diagram);
- диаграмма развертывания (deployment diagram).

Диаграммы кооперации и диаграммы последовательностей также можно называть диаграммами взаимодействия.

В UML языке применяются такие инструменты как: включение (adornments), классификация, устройство увеличения (extensibility mechanisms) и установленные части (common divisions).

Язык UML не зависит от объектно-ориентированных языков программирования, и не зависит от используемой методологии разработки проекта. Поддерживается различными объектно-ориентированными языками.

Методология UML была применена при создание программного обеспечения для мультиагентной системы распределения заказов.

Диаграмма состояний (диаграмма конечного автомата) – это диаграмма, на которой показан конечный автомат с простыми состояниями, переходами и композитными состояниями.

Любая диаграмма состояний демонстрирует все состояния одного сегмента конкретного класса и разные алгоритмы его переходов из одного состояния в иное, то есть формирует все изменения состояний предмета как его реакцию на наружные действия.

Диаграмма состояний ПО мультиагентной системы представлена на рисунке 2.1

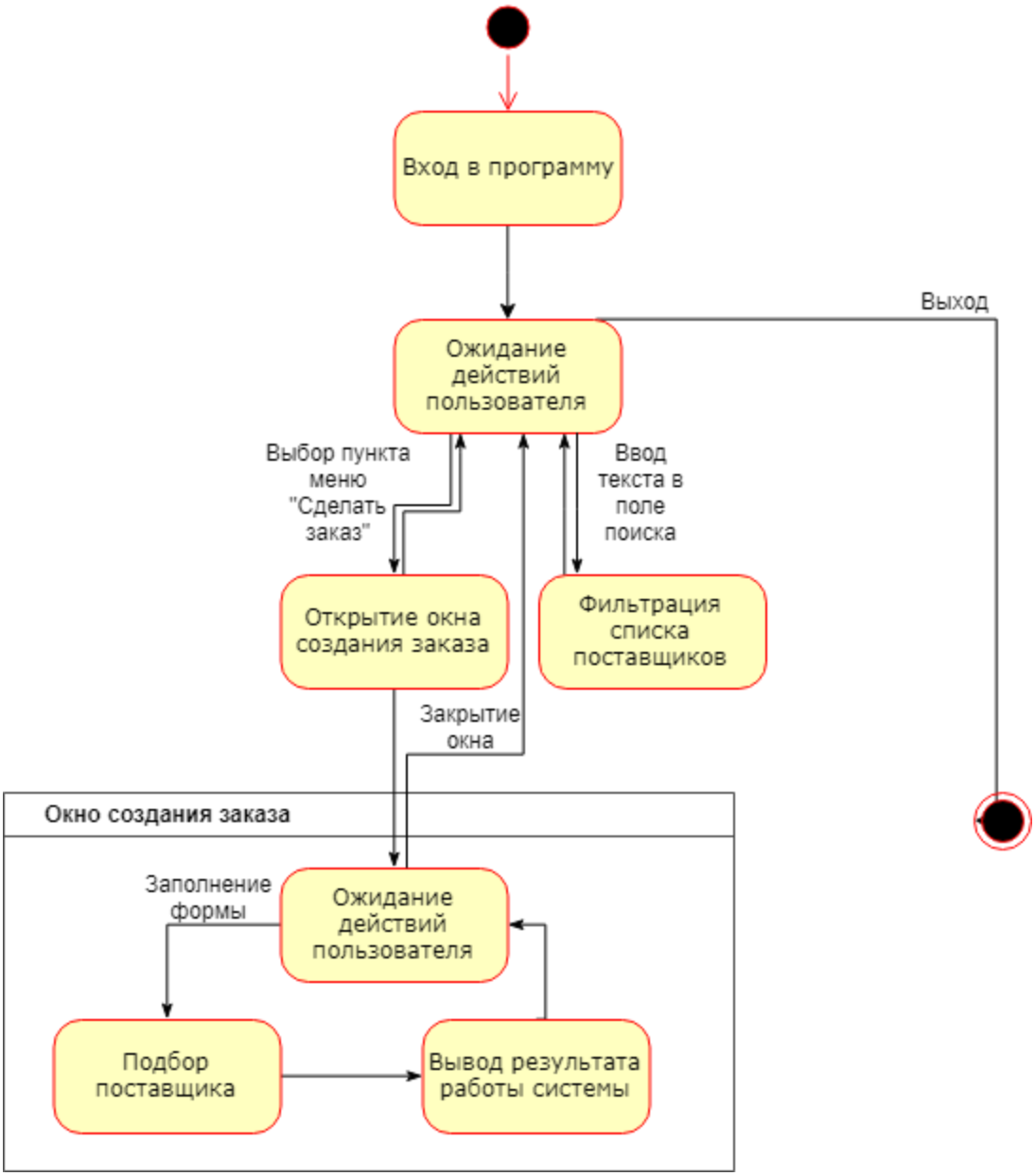

Рисунок 2.1 – Диаграмма состояний

Диаграммы состояний как правило используются с целью отображения действия других объектов, но в тот же период могут применяться для спецификации функциональности иных деталей моделей, таких как виды применения, подсистемы, актеры, способы и процедуры.

Диаграмма состояний - это схема особого типа, который предполагает некий автомат. Вершины схемы определяются как допустимые состояния автомата, которые показаны надлежащими графическими символами, а дуги показывают его переходы из одного состояния в иное положение.

Диаграммы состояний могут являться вложенными друг в друга с целью наиболее четкого понятия отдельных компонентов модели.

2.1.2 Создание базы данных торговой компании

При создании системы необходимо решить какую информацию база данных должна содержать. База данных имеет возможность хранить любую информацию, независимо от сферы, в которой она используется.

В данной работе база данных будет содержать список поставщиков, информация о которых будет использоваться при создании агентов-продавцов, и состоять из следующих таблиц:

- Продукция - хранит в себе список всей продукции, используемой поставщиками;
- Город - содержит список городов, в которых могут базироваться поставщики;
- Поставщики - содержит список всех поставщиков, которых поставляют товары, а также цену за единицу товара и время доставки.

Первым шагом будет создание новейшей таблицы в работе. Таблица – является собранием записей (строк таблицы), которая состоит из полей, которые отделенные от объекта (столбцов таблицы), хранящие данные. Таблицы – это основная модель отображения данных в реляционных БД. Без создания таблицы невозможно сделать базу, на основе таблиц в дальнейшем формируются отчеты, запросы.

Для создания таблицы необходимо сделать запрос на создание, используя язык SQL. Пример создания таблицы товаров показан на рисунке 2.2:

```
CREATE TABLE `product` (
  `id` int(11) NOT NULL,
  `name` varchar(65) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Рисунок 2.2 – SQL запрос на создание таблицы product

В таблице product созданы следующие поля:

- id – уникальный идентификационный номер для таблицы,
- name – текстовое значение, обозначающее наименование продукции.

```
ALTER TABLE `product`
  ADD PRIMARY KEY (`id`);
```

Рисунок 2.3 – Запрос на добавление первичного ключа полю id

Также для поля id необходимо установить значение AUTO_INCREMENT, благодаря которому значение id будет задаваться автоматически для каждой новой добавляемой строки, путем присвоения значению порядкового числа. На рисунке 2.4 показан запрос к базе данных:

```
ALTER TABLE `product`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT
```

Рисунок 2.4 – Запрос на добавление параметра AUTO_INCREMENT

Подобным образом будут созданы таблица city, хранящая список городов, и таблица distributor, содержащая список поставщиков с товарами, которые они могут продать. Ниже, в таблицах 2.1 и 2.2 будут показаны структуры таблиц city и distributor

Таблица 2.1 – Структура таблицы city

Название поля	Тип данных поля	Дополнительные параметры
id	Целое число	AUTO_INCREMENT PRIMARY_KEY
name	Строковое	

Таблица 2.2 – Структура таблицы distributor

Название поля	Тип данных поля	Дополнительные параметры
Id	Целое число	AUTO_INCREMENT PRIMARY_KEY
Name	Строковое	
City	Целое число	SECONDARY_KEY city.id
days_of_delivery	Целое число	
product	Целое число	SECONDARY_KEY product.id
product_quantity	Целое число	
price_for_product_unit	Целое число	

Диаграмма «сущность-связь» базы данных, которая включает в себя таблицы с описанием полей и типов данных, а также связями между таблицами показана на рисунке 2.5:

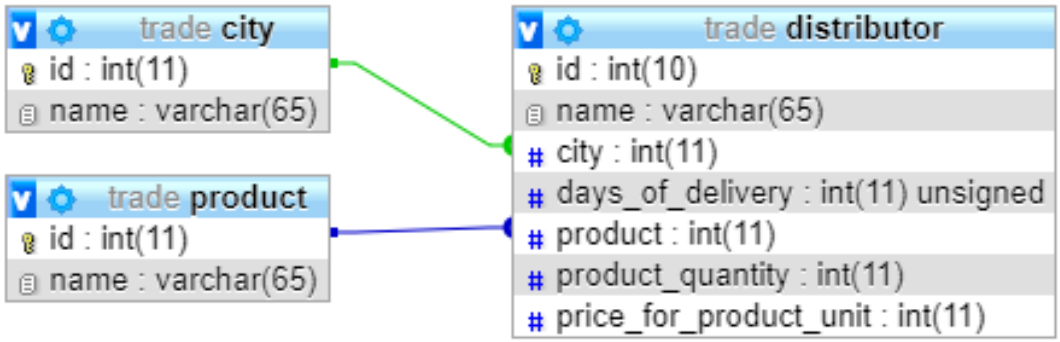

Рисунок 2.5 – Физическая модель данных

Общение с базами данных происходит с помощью запросов, направленных к этим базам данных. Добавление и удаление таблиц, изменение, получение и удаление информации из таблицы происходит с использованием специальных запросов.

Для разработки приложения «Торговая компания» нужны запросы для получения информации из таблиц, например, для выдачи списка наименований товаров. Ниже, на рисунке 2.6 показан фрагмент программного кода приложения с запросом к БД:

```
String query = "SELECT product.name "
             + "FROM product";
```

Рисунок 2.6 – Фрагмент кода приложения с запросом выборки

Также, необходим запрос на изменение информации в базе данных после завершения процесса покупки. На рисунке 2.7 представлен фрагмент программного кода приложения:

```
String query = "UPDATE distributor "
             + "SET distributor.product_quantity = '"+o[3]+"' "
             + "WHERE distributor.name = '"+agentName+"' and distributor.product = "
             + "(SELECT product.id FROM product WHERE product.name ='"+title+"') ";
```

Рисунок 2.7 – Фрагмент кода приложения с запросом изменения

Хранение информации в базах данных позволяет свободно распоряжаться данной информацией между пользователями, у которых имеется доступ, а также иметь постоянный доступ к данным, при наличии соединение с БД.

2.1.3 Сервис «желтых страниц»

Платформа JADE предоставляет возможность быстро узнавать агентам-покупателям о свободных агентах-продавцах при помощи сервиса «yellow pages».

Агент Directory Facilitator (DF) – менеджер директорий, который предоставляет работу сервиса «yellow pages», в котором агенты показывают данные о сервисах, которые они предоставляют. Агент находит других агентов при помощи DF, которые дают нужные сервисы, после между агентами происходят переговоры. Несколько DF способны быть в рамках одной системы, к примеру, которые дают информацию о разных группах сервисов либо о сервисах разных групп агентов. Образец пользования DF продемонстрирован ниже на рисунке 2.8.

Платформа JADE использует агента DF как принято изначально (с локальным именем«df») [15]. С целью предоставления единого распределенного каталога «yellow pages», некоторое количество DF агентов запускаются и объединяются, даже те, которые по умолчанию находятся на платформе.

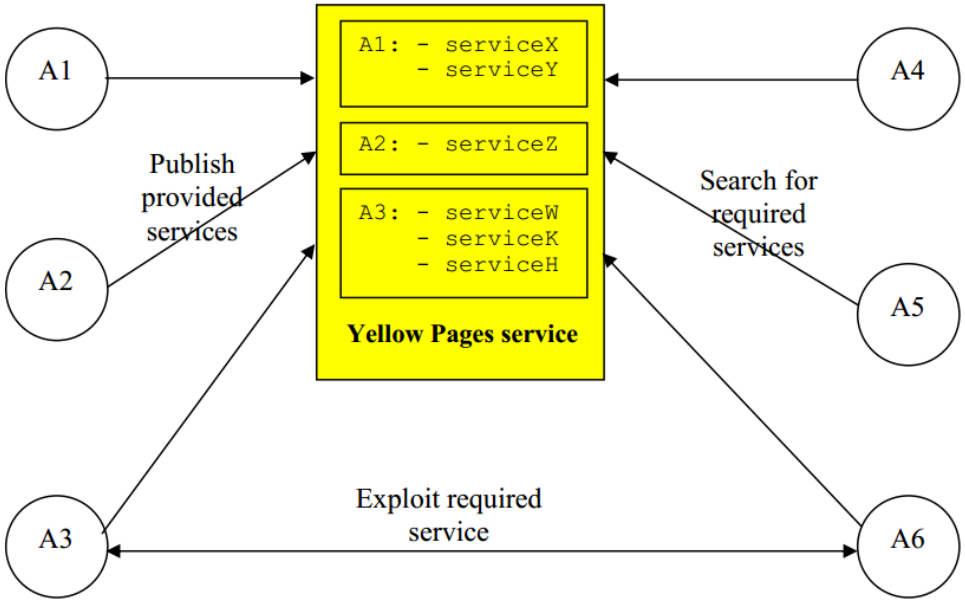

Рисунок 2.8 – Сервис «yellow pages»

Так как DF является агентом, существует вероятность действовать с ним, при помощи обмена ACL-сообщениями, то есть обычным способом, применяя, в согласии с спецификациями FIPA, подходящий язык содержания (SLO) и надлежащую онтологию (онтологию FIPA-agent-management). JADE дает класс jade.domain.DFService, при помощи которого возможна публикация и выполнение сервисов при помощи вызовы методов.

2.1.4 Поведение агентов мультиагентных систем

Агент активно действует ходе работы в рамках своего «поведения». В отдельном потоке Java выполняются все агенты . Из комбинации режимов создается логика агента . Поведение агента является последовательностью действий, которые необходимо совершить (аналогично методу Java-класса), и создается как объект класса, наследуемого от jade.core.behaviours.Behaviour. С помощью метода addBehaviour () класса Agent формируется поведение агента. Поведение добавляется в какой угодно момент: при запуске агента (в методе Setup ()) или внутри иного хода работы. Любой класс, который наследуется от класса Behaviour, обязан переопределить метод action(), который на самом деле определяет операции, которые задают поведение агента, также метод done(),

который возвращает булево значение, указывающее на окончание поведения, которое необходимо удалить из пула режимов работы агента [16].

Агент способен делать в одно время сразу некоторое количество моделей поведения. Поэтому необходимо заметить, что составление плана режимов агента считается не упреждающим , как к примеру в потоках Java, а кооперативным [10]. Значит, то что, когда поведение готовится к осуществлению, метод action() вызван и выполняется до того, пока режим не закончит свою работу. Поэтому программист обязан определять момент, когда агенту следует переключается от осуществления одного режима к осуществлению другого. Данный подход обладает рядом положительных моментов:

- поддерживает работу с единственным потоком Java для агента (что считается весьма значительным в условиях, где ограниченное количество ресурсов, к примеру, которыми обладают мобильные телефоны);
- дает возможность для более лучшего переключения между режимами работы, чем переключение между потоками Java;
- ликвидирует все без исключения трудности синхронизации параллельных режимов работы с целью доступа к одним и тем же ресурсам (с целью сделать лучше эффективность работы), так как все режимы производятся одним и тем же потоком Java;
- что дает возможность сохранить состояние агента в постоянном хранилище с целью дальнейшего возобновления (сохранямость агента) либо передачи его иному контейнеру для удаленного исполнения (мобильность агента).

Когда происходит формирование агента, то для этого делается вызов метода setup(), для инициализация агента. Затем, в нескончаемом цикле избирается последующий режим из пула активных и происходит его выполнение. Потом, в зависимости от вида режима, он удаляется из пула активных, или останется в нем и выполнится вновь, но только когда дойдет его

очередь. Если активные режимы отсутствуют (к примеру, в случае если агент ожидает уведомление) поток агента «переходит в спящий режим» (это дает возможность экономии ресурсов процессора). Для удаления агента вызывается метод takeDown().

Используя показанный способ планирования, режим, который продемонстрирован ниже, не дает возможности запуска иного режима, так как его метод action() никогда не заканчивается [18].

Всего существует три вида поведения агентов мультиагентных систем:

1) One-shot (одноразовый режим), в нем метод action() исполняется всего один единственный раз, после этого режим перестанет быть активным. jade.core.behaviours.OneShotBehaviour уже имеет метод done(), возвращающий значение true, и может быть с легкостью модифицирован для создания одноразового поведения.

```
public class MyOneShotBehaviour extends OneShotBehaviour {
    public void action() {
    // perform operation X
    }
}
```

Рисунок 2.9 – Пример one-shot behaviour

Выполнение операции X происходит всего один единственный раз.

2) Cyclic (циклический режим), который постоянно продолжает быть активным и не может закончиться. Метод action() осуществляет однообразные операции каждый раз, когда происходит его вызов. jade.core.behaviours.CyclicBehaviour уже имеет метод done(), который возвращает значение false, и может быть с легкостью модифицирован для создания циклического поведения.

```
public class MyCyclicBehaviour extends CyclicBehaviour {
 public void action() {
    // perform operation Y
      }
}
```

Рисунок 2.10 – Пример cyclic behaviour

Операция Y производится неоднократно, агент выполняет циклическое поведение до конца работы.

3) Generic (общий режим), осуществляющий разнообразные операции, которые зависят от значения некой переменной. Режим остается действующим вплоть до того, пока исполняется условие, данное в методе done().

```
public class MyThreeStepBehaviour extends Behaviour {
 private int step = 0;
      public void action() {
       switch (step) {
             case 0:
             // perform operation X
                   step++;
                   break;
             case 1:
             // perform operation Y
                   step++;
                   break;
                   case 2:
             // perform operation Z
                   step++;
                   break;
                }
             }
             public boolean done() {
                   return step == 3;
            }
       }
```

Рисунок 2.11 – Пример generic behaviour

Операции X, Y и Z производятся друг за другом, а потом поведение заканчивается. JADE учитывает вероятность комбинации простых режимов работы агента с целью формирования наиболее сложных режимов.

При создании данной программы будет использоваться generic и cyclic behaviour.

На платформе JADE имеются два готовых класса (в пакете jade.core.behaviours), с помощью которых с легкостью возможно создавать режимы работы, которые осуществляют определенные операции в установленные моменты времени.

1) WakerBehaviour, в данном режиме методы action() и done() выполнены таким способом, что абстрактный метод handleElapsedTimeout () осуществляется по окончанию установленного периода времени, который указан в конструкторе. Уже после выполнения метода handleElapsedTimeout () поведение заканчивается.

Операция X осуществляется через 10 секунд после того, как на экран выводится сообщение «Adding waker behaviour».

```
public class MyAgent extends Agent {
      protected void setup() {
            System.out.println("Adding waker behaviour");
            addBehaviour(new WakerBehaviour(this, 10000) {
                  protected void handleElapsedTimeout() {
                  // perform operation X
                  }
            } );
        }
      }
```

Рисунок 2.12 – Пример WakerBehaviour

2) TickerBehaviour, в данном режиме методы action() и done() выполнены таким способом, что абстрактный метод onTick() производится неоднократно, через конкретный интервал времени (указанный в конструкторе). Режим TickerBehaviour никогда не заканчивается.

```
public class MyAgent extends Agent {
      protected void setup() {
            addBehaviour(new TickerBehaviour(this, 10000) {
                  protected void onTick() {
                  // perform operation Y
                  }
               } );
            }
         }
```

Рисунок 2.13 – Пример TickerBehaviour

Операция Y производится время от времени каждые 10 секунд.

При разработке агента-покупателя в данной программе будет использоваться tickerBehaviour для периодичного выполнения поиска подходящий агентов-продавцов.

2.1.5 Способы взаимодействия агентов в мультиагентных системах

Одной из более значимых способностей считается способность коммуникации среди агентов, которую дает JADE. С помощью Message Transport System происходит обмен оповещениями в JADE, который является асинхронным. Параметры уведомлений отвечает стандарту FIPA. Для того чтобы ускорить работу, внутри платформы уведомления пересылаются в виде Java-объектов, а при обмене между платформами (либо с иными мультиагентными системами) – в виде XML-строки [20]. При отправке уведомления оно оказывается в «почтовом ящике» , то есть попадает в очередь уведомлений агента-адресата, после чего агент получает извещение. Уведомление принимается из «почтового ящика» агента и подвергается обработке любым из его режимов. С целью подбора подходящего вида уведомления из очереди разработчик способен применять фильтры. Образец обмена уведомлениями представлен на рисунке 2.14.

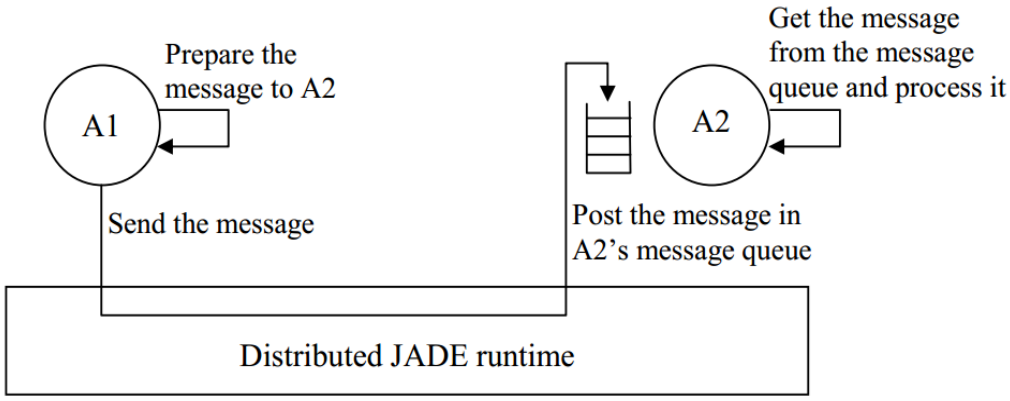

Рисунок 2.14 – Образец обмена уведомлениями среди агентов

Между агентами JADE происходит обмен уведомлениями, такие уведомления обладают параметрами языка ACL, конкретного FIPA, как

международный стандарт взаимодействия агентов. Данный параметр содержит следующие поля:

- адресант уведомления;
- перечень получателей;
- коммуникативные действия («пожелания»), которые указывают цель отправки уведомлений. Уведомления бывают таких видов:

1) REQUEST (запрос), если отправитель хочет, чтобы получатель совершил определенное действие;

2) INFORM (информирование), если отправитель хочет, чтобы получатель был уведомлен о каком-либо прецеденте;

3) QUERY_IF, если отправитель хочет понять, сделано или нет установленное требование; CFP (извещение о предложении);

4) PROPOSE (предложение);

5) ACCEPT_PROPOSAL (принятие предложения);

6) REJECT_PROPOSAL (отклонение предложения), если отправитель и получатель ведут диалог, иные виды коммуникативных действий;

- содержание, то есть подленные сведения, содержащиеся в уведомлении, к примеру, действия, которые станут осуществляться согласно запросу REQUEST; прецендент, о котором отправитель хочет оповещать в послание INFORM и так далее;
- язык, на котором продемонстрировано содержание (отправитель и получатель обязаны владеть способностью кодировать/декодировать выражения в соответствие с синтаксисом языка);
- онтология, то есть словарь символов, которые используются в содержании, и их смысл (у отправителя и получателя должно быть одинаковое понимание смысла символов);

- поля, применяемые с целью управления некоторым количеством параллельных переговоров, которые определяют промежуток времени с целью получения ответа, например: conversation-id, reply-with, in-reply-to, reply-by.

На рисунке 2.15 продемонстрировано объявление уведомления.

```
ACLMessage msg = myAgent.receive(mt);
    if (msg != null) {
        Object[] offer = msg.getContent().split("-");
```

Рисунок 2.15 – Пример получения уведомления

Уведомление в JADE создается как объект класса jade.lang.acl.ACLMessage, который предоставляет методы get() и set() с целью установки значений всех полей при получении и посылке уведомлений.

2.1.6 Формы для представления выходных данных

Для ввода и просмотра информации в подходящем для пользователя виде, отвечающему документу, который ему привычен, для этого служат формы. При выводе информации при помощи таких форм, становится возможно использовать специализированные ресурсы для оформления.

При создании приложения «Торговая компания» были созданы форма для вывода информации по поставщикам, а также форма для создания заказа. Для создания форм использовались средства библиотеки javax.swing. На рисунке 2.16 изображен список элементов формы, служащей для вывода информации о поставщиках:

```
private javax.swing.JTable distributorTable;
private javax.swing.JMenuItem exitMenuItem;
private javax.swing.JLabel jLabel1;
private javax.swing.JMenu jMenu1;
private javax.swing.JMenuBar jMenuBar1;
private javax.swing.JMenuItem jMenuItem2;
private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JPopupMenu.Separator jSeparator2;
private javax.swing.JMenuItem newOrderMenuItem;
private javax.swing.JTextField searchField;
private javax.swing.JComboBox<String> searchType;
```

Рисунок 2.16 – Элементы формы ActionWindow

Среди данных элементов имеются: таблица, необходимая для отображения информации из базы данных (JTable), меню, которое находится в верхней части окна (JMenuBar), текстовый ярлык, служащая с целью формирования подписей на окне (JLabel), текстовое поле для ввода(JTextField), перечень которых выпадает (JComboBox) и т. д.

2.2 Разработка классов-агентов мультиагентной системы

Агент JADE формируется при поддержки определения класса, который наследуется от класса jade.core.Agent, и исполнения метода setup():

```
public class DistributorAgent extends Agent {
    @Override
    protected void setup() {
        System.out.println("Hello. Distributor-Agent "+getAID().getName()+" is ready.");
```

Рисунок 2.17 – Начало реализации метода setup()

Метод setup() осуществляет инициализацию агента. Затем агент работает по заранее заданному плану реакций на происходящие.

Любой агент обладает индивидуальным «ID агента», являющийся образцом класса jade.core.AID. Метод getAID() класса Agent дает возможность приобрести ID агента. У объекта AID есть уникальное неповторимое наименование и адрес. ID каждого агента JADE обладает форматом: ::<agent_name>@<platform_name>, по этой причине агент с именем «United Trade», который располагается в системе tradeinc.com/trade, будет обладать уникальным и неповторимым именем UnitedTrade@tradeinc.com. Адрес, который имеется в AID, – это адрес системы, на которой расположен агент. Данный адрес применяется только лишь в тогда, когда агенту надо контактировать с иными агентами, которые находятся в другой системе.

Создание агента происходит через программный код приложения:

```
JadeHandler.distributor = JadeHandler.agentContainer.createNewAgent(previousDistributor,"salescomp.DistributorAgent",arr);
JadeHandler.distributor.start();
```

Рисунок 2.18 – Пример кода создания агента-продавца

Для того чтобы привести в действие откомпилированного агента на выполнение, следует запустить среду JADE. Одним из вариантов запуска является запуск через программный код приложения:

```
rt = Runtime.instance();
```

Рисунок 2.19 – Запуск среды JADE через программный код

Во время запуска платформы JADE производится активация ядра JADE. Процедура заканчивается уведомлением, о готовности к работе контейнера «MainContainer». Среда JADE активирует агента, созданного пользователем, который выводит собственное обращение с названием, отмеченным в свойствах формирования агента. Название платформы присваивается автоматически в соответствии с адресом расположения активной среды JADE.

Даже если у пользовательского агента нет поставленных задач на момент запуска, то он все равно продолжает работать. Для чтобы работа была закончена, нужно использовать метод doDelete (). Перед окончанием работы агента используется метод takeDown (), выполняющий операции по очистке агента.

Агенты приобретают аргументы, которые указаны в командной строке, при запуске. Данные аргументы представляются в виде массива Object[] при помощи метода getArguments () класса Agent. Агенту CustomerAgent надо приобрести в качестве аргумента командной строки название заказа, который он должен сделать и количество заказа. Для получения ожидаемого результата заменим агента.

2.3. Реализация методов поведения классов-агентов

Агент Customer обязан время от времени просить у агента продавца товар, который ему нужно купить. Это возможно добиться, если изменить режим TickerBehaviour так, что всякий «шаг» прибавляет иной режим, который,

по сути, осуществляет запрос к агенту-продавцу. Метод setup() класса CustomerAgent следует изменить следующим способом.

```
public class CustomerAgent extends Agent {
    protected void setup(){
      System.out.println("Hello, Customer-Agent " + getAID().getName() + " is ready!");
      Object[] args = getArguments();

      if (args != null && args.length >0){
          productTypeName = (String) args[0];
          productQuantity = (int) args[1];
          System.out.println("Trying to buy "+productTypeName+ ". Amount: "+productQuantity);

          addBehaviour(new TickerBehaviour(this,tickTimer)
          {
              @Override
              protected void onTick(){
                myAgent.addBehaviour(new RequestPerformer());
              }
          });
      }
      else
      {
          System.out.println("No product type specified!");
          doDelete();
      }
    }

    @Override
    protected void takeDown(){
        System.out.println("Customer-agent " + getAID().getName() +" terminating.");
    }
```

Рисунок 2.20 – Разработка CustomerAgent

Агент Distributor ждет прихода запросов от агентов-покупателей и обслуживает их. Запросы бывают разных типов: запрос о представлении предложения на товар и запрос на осуществление заказа. С целью реализации запросов разных типов агенту Distributor нужно задать два циклических поведения: первое для обслуживания запросов на предложение, а второе для обслуживания заказов. Разработка агента DistributorAgent на платформе JADE представлена на рисунке 2.21.

```
public class DistributorAgent extends Agent {
    private final List<Object[]> catalogue = new ArrayList<>();

    @Override
    protected void setup() {
        System.out.println("Hello. Distributor-Agent "+getAID().getName()+" is ready.");

        addBehaviour(new OfferRequestsServer());
        addBehaviour(new PurchaseOrdersServer());
    }

    @Override
    protected void takeDown() {
        try {
            DFService.deregister(this);
        }
        catch (FIPAException fe) {
            fe.printStackTrace();
        }
        System.out.println("Distributor-agent "+getAID().getName()+" terminating.");
    }
```

Рисунок 2.21 – Разработка DistributorAgent

Агенту Distributor следует задать одноразовое поведение, которое реализует обновление журнала заказов, легкодоступных с целью продажи в тех случаях, когда пользователь добавляет новый товар из базы данных.

2.4 Дополнение классов-агентов методами взаимодействия агентов

В данном приложении применяется уведомление CFP с целью оповещения о том, что агент-продавец DistributorAgent выслал агенту-покупателю CustomerAgent предложение на приобретение товара. Сообщение PROPOSE имеет предложение продавца, а уведомление ACCEPT_PROPOSAL сохранит заказ покупателя. Уведомление REFUSE агента-продавца информирует о том, что товара, о котором запросил агент-покупатель, не имеется в каталоге. В двух видах уведомлений, которые отправляются агенту-покупателю, находится название товара. Содержанием уведомления PROPOSE должна быть информация о товаре. Далее показан пример, который демонстрирует, как формируется и оправляется уведомление CFP.

```
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
for (int i = 0; i < distributorAgent.length; ++i) {
    cfp.addReceiver(distributorAgent[i]);
}
cfp.setContent(productTypeName+"-"+productQuantity);
cfp.setConversationId("type-distribution");
cfp.setReplyWith("cfp"+System.currentTimeMillis());

myAgent.send(cfp);

mt = MessageTemplate.and(MessageTemplate.MatchConversationId("type-distribution"),
MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
```

Рисунок 2.22 – Пример отправки сообщения

Как только уведомления были доставлены, среда исполнения JADE автоматически перемещает их в очередь уведомлений получателя. Благодаря методу receive() агент способен получать уведомления из своей очереди уведомлений. Данный метод возвращает первое уведомление, которое находится в очереди, после чего убирает его из этого места, или ничего не возвращает, если в очереди уведомлений они отсутствуют. После чего очередь уведомлений сразу же заканчивается.

```
ACLMessage reply = myAgent.receive(mt);
```

Рисунок 2.23 – Метод receive()

Зачастую при разработке желаемого поведения следует создавать получение уведомлений от иных агентов, к примеру, в режимах PurchaseOrdersServer и OfferRequestsServer [11]. Поведение агента DistributorAgent, надо принимать и обрабатывать уведомления от агента покупателя, который пересылает заявки на заказ и предложение. Подобное поведение обязано функционировать непрерывно, то есть цикличное поведение, но при этом каждый раз при исполнении метода action() следует делать проверку, что уведомление было принято и подвергнуто обработке. Далее показан режим OfferRequestsServer.

```
private class OfferRequestsServer extends CyclicBehaviour {
    @Override
    public void action() {
        MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
        ACLMessage msg = myAgent.receive(mt);
            if (msg != null) {
                Object[] offer = msg.getContent().split("-");

                String offerTitle = offer[0].toString();
                int offerQuantity = Integer.parseInt(offer[1].toString());
                int price = 0,days = 0,quantity = 0;
                boolean isFound = false;

                ACLMessage reply = msg.createReply();

                if(catalogue!=null){
                    for (int i=0;i<catalogue.size();i++){
                        Object[] o = catalogue.get(i);
                        String name = o[0].toString();

                        if ((offerTitle.equalsIgnoreCase(name))&&(offerQuantity<=(int)o[3]))
                        {
                            System.out.println();
                            price = (int)o[1];
                            days = (int)o[2];
                            quantity = (int)o[3];
                            isFound = true;
                            break;
                        }
                    }
                    if (isFound) {
                        System.out.println("Sending proposal");
                        reply.setPerformative(ACLMessage.PROPOSE);
                        reply.setContent(price+"-"+days+"-"+quantity);
                    }
                }
            else {
                    reply.setPerformative(ACLMessage.REFUSE);
                    reply.setContent("not-available");
                }
                myAgent.send(reply);
            }
    }
}
```

Рисунок 2.24 – Внутренний класс OfferRequestsServer

В этом фрагменте поведение OfferRequestsServer включенный в описание класса как внутренний класс класса DistributorAgent. Что в разы упрощает реализацию, так как дает доступ к перечню товаров с целью реализации на прямую, однако это никак не считается неотъемлемым.

Метод createReply () класса ACLMessage по умолчанию формирует обновленные параметры ACLMessageproperly для агента адресата и все свойства, которые используются с целью контроля сеанса (conversation-id, reply-with, in-reply-to). При разработке паттерна, поток агента поступает в постоянный цикл, что очень загружает процессор. Чтобы избежать загруженности процессора, метод action() входящий в поведение OfferRequestsServer необходимо осуществлять только, если придет новое

уведомление. Поэтому допускается применять метод block() класса поведений. Данный метод отмечает поведение как «заблокированное», таким образом указывая что агент не собирается делать последующих выполнений. Если в очередь уведомлений агента добавится новое уведомление, то все режимы, которые были заблокированы, вновь станут доступными для исполнения и получат шанс делать обработку приобретенных уведомлений.

Так как состояния PurchaseOrdersServer и OfferRequestsServer считаются цикличными, их метод action(), стартующие с метода myAgent.receive(), создают следующую сложность: нет возможности понять, правильно ли поступает OfferRequestsServer делая выбор из потока уведомлений только на те уведомления, которые содержат запрос о находится данный товар в наличии, а поведение PurchaseOrdersServer – выбирая только уведомления, содержащие заказы-поставки?

Для того чтобы найти решение данной проблемы следует изменить код, и использовать подходящие «шаблоны» для метода receive(). При указанном шаблоне метод receive() будет отвечать уведомлением, которое соответствует этому шаблону, не беря во внимание все неподходящие уведомления. Такие шаблоны созданы как экземпляры класса jade.lang.acl.MessageTemplate, предоставляющий группу методов, позволяющие быстро и легко формировать шаблоны. Для запроса о доступности товара применяется уведомление CFP, и уведомление ACCEPT_PROPOSAL – для отправки предложения-заказа. По этой причине необходимо заменить метод action() класса OfferRequestServer таким образом, чтобы вызов myAgent.receive() пропускал все уведомления, за исключением CFP.

```
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt);
    if (msg != null) {
        //do stuff
```

Рисунок 2.25 – Пример проверки типа уведомления

Режим RequestPerformer предполагает пример поведения, который проводит «сложные» переговоры. В этом случае, «переговоры» – это очередь

уведомлений, которыми меняются некоторому числу агентов с четко определенными паттерном отношений по времени и причинам. Поведение RequestProposal отправляет CFP-уведомление некоторому числу агентов, которые известны всем агентам-продавцам, вернуть все без исключения ответы, но при получении ответа с типом PROPOSE, то позже высылается уведомление ACCEPT_PROPOSAL агенту-продавцу, сделавшему предложение. Всякий раз, когда совершается обмен уведомлениями, нужно определить управляющие поля в уведомлениях, которые участвуют в обмене. Это даст возможность свободно и однозначно формировать шаблоны, которые соответствуют возможным ответам.

```
case 2:
    ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);

    order.addReceiver(bestSeller);
    order.setContent(productTypeName+"-"+productQuantity);
    order.setConversationId("type-distribution");
    order.setReplyWith("order"+System.currentTimeMillis());

    myAgent.send(order);

    mt = MessageTemplate.and(MessageTemplate.MatchConversationId("type-distribution"),
    MessageTemplate.MatchInReplyTo(order.getReplyWith()));

    step = 3;
break;
```

Рисунок 2.26 – Фрагмент кода с отправкой уведомления ACCEPT_PROPOSAL

JADE дает фундамент для осуществления обмена уведомлениям, используя протоколы, которые, в свою очередь, заложены в jade.protopackage.

Помимо метода receive(), класс Agent включает в себя метод blockingReceive(), являющийся вызов, который блокируется, так как ответ от него не будет содержать управление вплоть до момента появления уведомлений в очереди агента. Перегруженная версия данного метода принимает MessageTemplate как аргумент. Отметим, что метод blockingReceive() по факту не дает потоку агента совершать что-либо. По этой причине, в случае вызова каким-либо режимом метода blockingReceive(), исполнение остальных режимов останавливается, до возврата методом blockingReceive() не вернет управление. Чтобы переговоры между агентами

правильно функционировали, нужно применить blockingReceive() в методах setup() и takeDown().

2.5 Применение сервиса «Желтых страниц» для классов-агентов

Агент, стремящийся сделать легкодоступными несколько вариантов услуг, обязан обеспечить DF, содержащий AID этого агента, перечень всевозможных языков и онтологий, о которых знают иные агенты с целью общения с ним, также перечень сервисов с целью публикации. Для любого сервиса, который публикуется предоставляется описание, в котором находятся вид сервиса, его название, языки и онтологии, которые нужны для его применения, и еще некие параметры, уникальные для этого сервиса. Классы ServiceDescription, DFAgentDescription и Property, поступающие в структуру пакета jade.domain.FIPAAgentManagement, целесообразно предполагаются как несколько перечисленных абстракций. Чтобы сделать публикацию сервиса, агенту следует сформировать описание, которое является единицей класса DFAgentDescription, и обратиться к статическому методу register() класса DFService. Как правило регистрация сервиса совершается в методе setup(), как представлено далее на примере агента-продавца товара [21].

```
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("type-distribution");
sd.setName("JADE-type-distribution");
dfd.addServices(sd);
try {
    DFService.register(this, dfd);
}
catch (FIPAException fe) {
    fe.printStackTrace();
}
```

Рисунок 2.27 – Использование сервиса «yellow pages» класса DistributorAgent

Агент, которому надо отыскать некие сервисы, обязан обеспечить агенту DF представление нужного шаблона. Итогом поиска будет считается перечень

всех описаний, удовлетворяющих шаблону, объявленных ранее. Описание отвечает шаблону, если все поля, которые были объявлены в шаблоне, содержатся с этими данными, что и в описании. Применение статического метода search() класса DFService возможно показать на примере, где агент-покупатель товара быстро отыскивает каждого агента, которые предоставляются сервисом «type-distribution».

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("type-distribution");

template.addServices(sd);
    try {
        DFAgentDescription[] result = DFService.search(myAgent, template);
        distributorAgent = new AID[result.length];
        for (int i = 0; i < result.length; ++i) {
            distributorAgent[i] = result[i].getName();
            System.out.println(distributorAgent[i].getName());
        }
    }
    catch (FIPAException fe) {
            fe.printStackTrace();
    }
myAgent.addBehaviour(new RequestPerformer());
```

Рисунок 2.28 – Применение статического метода search() в CustomerAgent

Отметим, то что пополнение перечня существующих агентов-продавцов совершается постоянно перед попыткой приобрести необходимый товар, потому что агенты-продавцы быстро образуются и пропадают. Класс DFService дает возможность автоматически получать уведомления от DF о создании агента, который дает выбранный сервис (методы searchUntilFound() и createSubscriptionMessage()).

В данной главе показан процесс разработки программного обеспечения мультиагентной системы распределения заказов, описаны поведения агентов, а также сформирован пользовательский интерфейс для упрощенного восприятия получаемой информации.

ГЛАВА 3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Демонстрация работы разработанного приложения

Разработанное приложение предназначено для упрощения процесса закупки товаров для пользователей, при помощи мультиагентной системы, которая подбирает наиболее подходящего для пользователя поставщика товара.

Пользовательский интерфейс разработанного приложения представлен на рисунке 3.1:

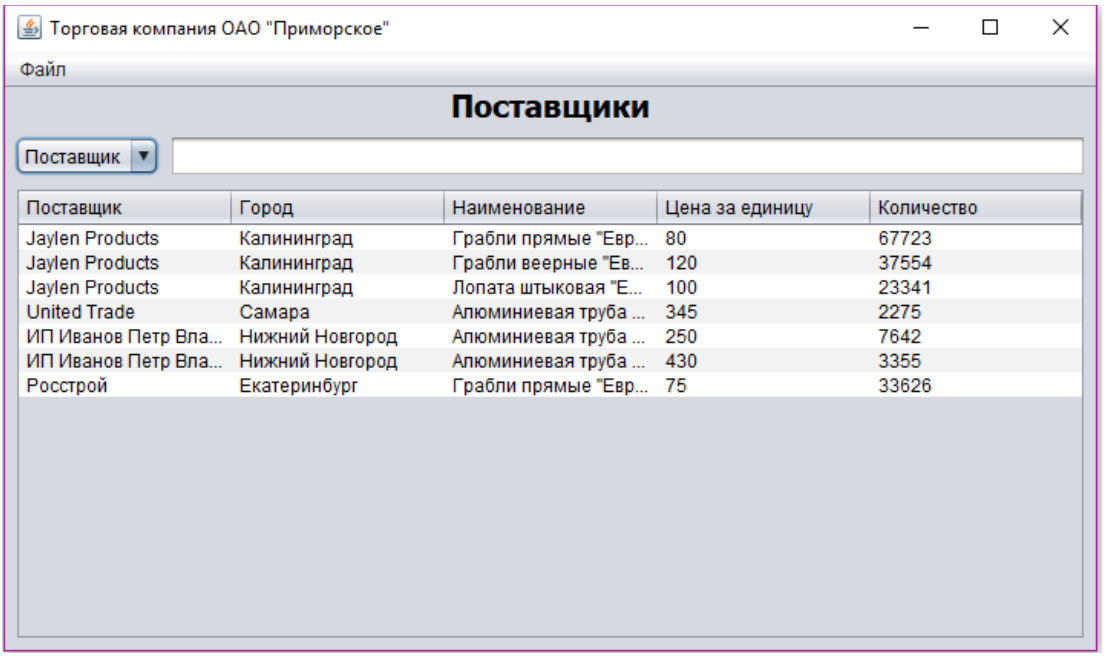

Рисунок 3.1 – Стартовый экран приложения «Торговая компания»

В центре окна показана таблица со всеми поставщиками, а также со всей информацией по товарам, которые они продают. Текстовое поле над таблицей служит для фильтрации таблицы по полю, указанному в выпадающем списке в левой части окна (рисунок 3.2).

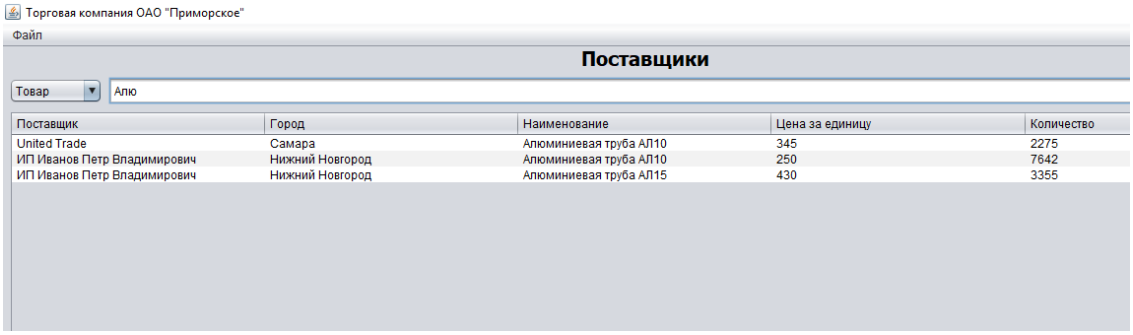

Рисунок 3.2 – Демонстрация работы фильтра по полю

Для создания заказа необходимо перейти в меню в верхней части окна и выбрать пункт «Сделать заказ», либо использовать комбинацию клавиш ALT+O (рисунок 3.3).

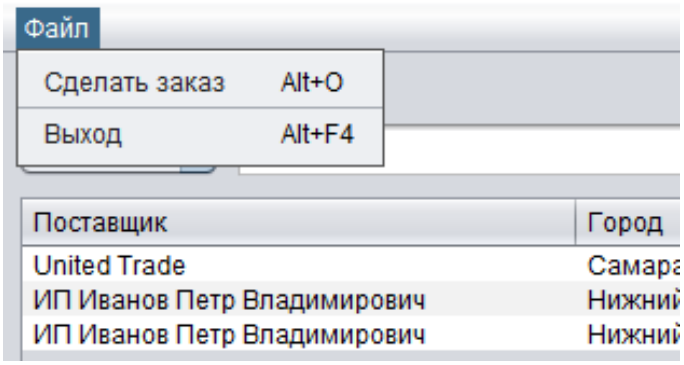

Рисунок 3.3 – Выпадающее меню

Чтобы сделать заказ необходимо указать наименование товара, который нужно заказать, а также его количество. На рисунке 3.4 показан пример создания заказа товара с наименованием «Алюминиевая труба АЛ10» в количестве 355 штук.

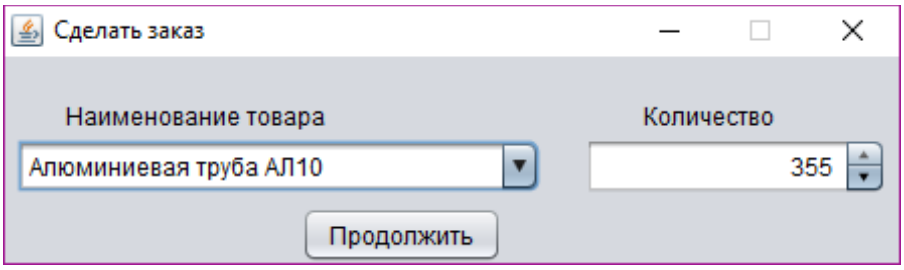

Рисунок 3.4 – Окно «Сделать заказ»

После совершения заказа появляется предупреждение, благодаря которому пользователь может уточнить, правильно был составлен заказ или нет (рисунок 3.5). При нажатии кнопки «NO» фокус возвращается на предыдущее окно, чтобы пользователь мог изменить заказ.

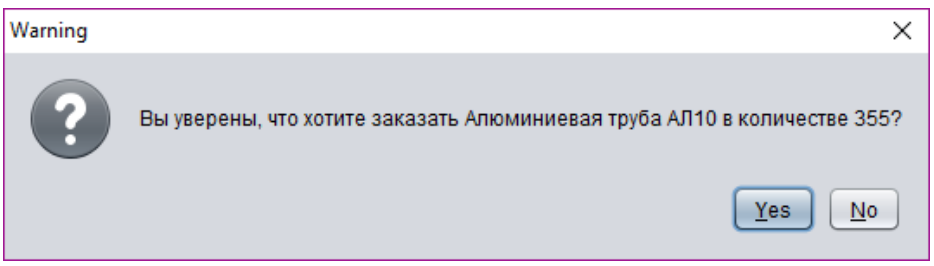

Рисунок 3.5 – Окно подтверждения совершения заказа

При подтверждении создания заказа появляется окно, с просьбой подождать (рисунок 3.6). Агенту нужно время для совершения первой итерации поиска агентов.

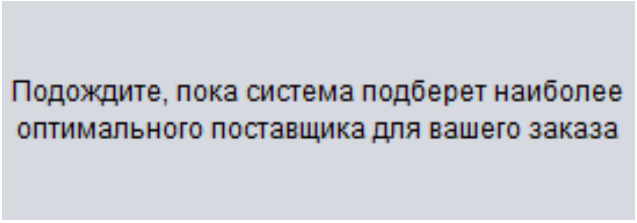

Рисунок 3.6 – Предупреждающее окно

В случае успешного подбора агента-продавца появляется окно с результатом работы. На рисунке 3.7 представлено окно с выполненным заказом «Алюминиевая труба АЛ10» в количества 355.

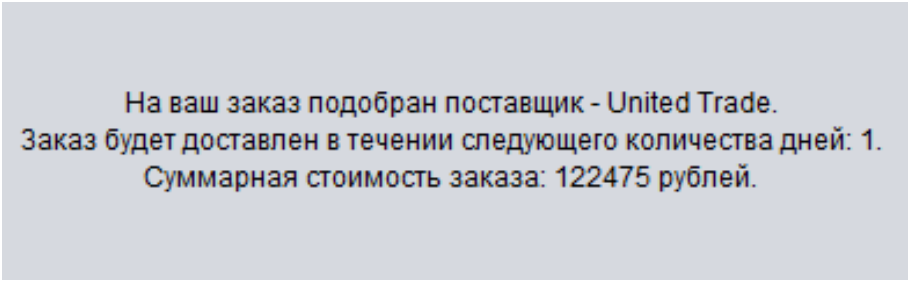

Рисунок 3.7 – Окно результатов поиска поставщика

Если не найдены агенты, содержащие искомый товар или необходимое количество товара, то выдастся сообщение с просьбой подождать, когда появится агент с искомым товаром или же с подходящим количеством товара (рисунок 3.8).

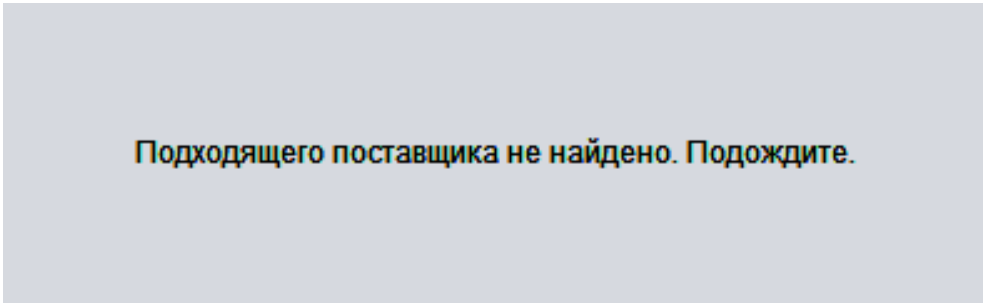

Рисунок 3.8 – Сообщение с просьбой ожидания

Разработанное программное обеспечение мультиагентной системы обладает простым и не нагроможденным интерфейсом, что увеличивает

скорость обучения работы в приложении, а также повышает производительность, поскольку нет необходимость загружать вычурные и зачастую ненужные графические интерфейсы с изображениями и движущимися объектами

3.2 Тестирования разработанного приложения

Тестирование разработанного приложения проводится с целью выявления ошибок, которые необходимо исправить, а также с целью выявления быстродействия программного продукта. При низком быстродействии необходимо оптимизировать работу процессов.

После выполнения пятисот запусков программы, среднем временем запуска программы при наличии стабильного соединения с сервером, который хранит базу данных, является 0.84354 секунды. Данный результат не нуждается в оптимизации и является хорошим показателем.

Одним из примененных методов тестирования было применено ручное тестирование, в ходе которого разработчик пытается сымитировать действия предполагаемого пользователя с целью выявления ошибок. Для выявления возможных ошибок необходимо провести тестирование вводимых пользователем данных. Ниже, в таблице 3.1 показаны проводимые действия и результат работы программы.

Таблица 3.1 - Результаты тестирования программы

Действия пользователя	Результат программы
При создании заказа попытка ввода отрицательного числа или буквенного значения	Поле ввода числа ограничивает вводимые значения числами с минимальным значением 1
При создании заказа попытка ввода числа, заведомо больше, чем максимальное доступное число данного товара среди всех поставщиков	Программа принимает заказ, сообщает пользователю о том, что подходящего заказа на данный момент нет и ожидает появления в базе данных поставщика, который удовлетворит заказ
При создании заказа попытка ввода названия товара, заведомо не существующего среди всех поставщиков	Программа принимает заказ, сообщает пользователю о том, что подходящего заказа на данный момент нет и ожидает появления в базе данных поставщика, который удовлетворит заказ

По результатам проведенного тестирования, ошибок выполнения программы обнаружено не было.

Еще одним способом тестирования, которому подвергалось разработанное программное обеспечение было системное тестирование на соответствие требованием к программному обеспечению. После внедрения программного обеспечения в систему торговой компании ОАО «Приморское» доступ к системе был предоставлен группе пользователей в количестве двадцати двух человек. В ходе проведенного тестирования в течении двух недель, ошибок выявлено не было.

Зачастую разработчик не замечает, или не желает замечать небольшие совершенные им ошибки, которые в последствии могут испортить впечатление о программном обеспечении и ухудшить качество работы пользователя. Для того, чтобы обнаружить данные ошибки до начала эксплуатации программного обеспечения, необходимо проводить разного рода тестирования. Благодаря примененным методам тестирования, которым подверглось программное обеспечение мультиагентной системы распределения заказов, было обнаружено отсутствие ошибок в разработанном приложении. Пользователи, участвовавшие в системном тестировании, положительно отзывались об опыте пользования приложения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы была проанализирована необходимая литература, при помощи которой были определены основные требования, предъявляемые к разработке мультиагентной системе.

Для выполнения поставленной цели в бакалаврской работе был проведен анализ предметной области, рассмотрены основные требования к системе проектирования приложения, методические рекомендации по разработке приложения, структурная организация, а также была изучения информационная система, используемая организацией в своей работе.

Была подробно исследована предметная область, на основе анализа которой были определены требования к функциональным характеристикам разрабатываемой мультиагентной системе.

Выбран комплекс технических и программных средств реализации, а также был спроектирован программный продукт. Система реализована в трехзвенной архитектуре «клиент-сервер» на языке Java с применением библиотеки JADE. В качестве системы базы данных применена СУБД MySQL. К плюсам данной разработки можно отнести то, что данную мультиагентную систему легко эксплуатировать и поддерживать, а также такая система недорого обойдется в обслуживании.

Созданное программное обеспечение было протестировано и успешно внедрено в рабочую систему компании. Использование приложения значительно повысило качество и скорость работы по подбору поставщиков.

Задачи, выполненные в ходе работы, позволили в итоге создать мультагентную систему для торговой компании.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Бессмертный И.А. Интеллектуальные системы. Учебник и практикум. / И.А. Бессмертный, А.Б. Нугуманова, А.В. Платонов - М.:ЮРАЙТ, 2017. - 244 с.

2. Васильев А.Н. Python на примерах. Практический курс по программированию. / А.Н. Васильев - СПб.:Наука и Техника, 2017. - 432 с.

3. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес – СПб.:Питер, 2016. – 366 с.

4. Гома Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений. / Х. Гома - М.:ДМК Пресс, 2016. - 700 с.

5. Зобнин Б. Мультиагентные системы / Б. Зобнин, А. Вожегов. - LAP Lambert Academic Publishing, 2014. - 156 с.

6. Ларман К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. / К. Ларман - М.:Вильямс, 2013. - 736 с.

7. Лафоре Р. Структуры данных и алгоритмы в Java. / Р. Лафоре - СПб.:Питер, 2017. - 704 с.

8. Лихтенштейн В.Е. Мультиагентные системы. Самоорганизация и развитие. / В.Е. Лихтенштейн, В.А. Конявский, Г.В. Росс, В.П. Лось - М.:Финансы и статистика, 2018. - 264 с.

9. Макаров В. Л. Социальное моделирование — новый компьютерный прорыв (агент-ориентированные модели). / В. Л. Макаров, А. Р. Бахтизин. — М.: Экономика, 2013. — 295 с.

10. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. / Р. Мартин - СПб.:Питер, 2018. - 352 с.

11. Пароджаров С. Многоагентные системы. Взаимодействие. / С. Пароджаров - СПб.: LAP Lambert Academic Publishing, 2012. - 200 с.

12. Советов Б.Я. Интеллектуальные системы и технологии. / Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской - М.:Academia, 2013. - 320 с.

13. Федорова Г. Н. Информационные системы. / Г. Н. Федорова - СПб.:Academia, 2013. - 208 с.

14. Хорстманн К. Java. Библиотека профессионала. Том 2. Расширенные средства программирования / К. С. Хорстманн – М.:Вильямс, 2017. – 976 с.

Электронные ресурс

15. Java Agent DEvelopment Framework [Электронный ресурс] URL: http://jade.tilab.com/.

16. Агентная платформа JADE: [Электронный ресурс] URL: http://www.studfiles.ru/preview/2947243/#2947243

17. Портал искусственного интеллекта. Многоагентные системы: [Электронный ресурс] / М., 2015. URL: http://www.aiportal.ru/articles/multiagent-systems/multiagent-systems.html/.

Литература на иностранном языке

18. Bai Q. Multi-agent and Complex Systems (Studies in Computational Intelligence) / Quan Bai, Fenghui Ren, Katsuhide Fujita, Minje Znang. - Luxembourg:Springer, 2016. - 210 с.

19. Choulier D. Developing multiagent systems for design activity analysis/ D. Coulier, A Fougèresa, E. Ostrosi // Sciencedirect, 2015. Vol. 59. P. 201-213. doi:10.1016/j.cad.2014.10.007

20. Kaliaev A. Multiagent Approach for Building Distributed Adaptive Computing System / A. Kaliaev // Procedia Computer Science, 2013. Vol. 18. P. 2193-2202. doi:10.1016/j.procs.2013.05.390

21. Sandita A.V. Developing A Multi-Agent System in JADE for Information Management in Educational Competence Domains. / A.V. Sandita, C.I. Popirlan // Procedia Economics and Finance, 2015. Vol. 23. P. 478-486. doi:10.1016/S2212-5671(15)00404-9

22. Weiss G. Multiagent Systems (Intelligent Robotics and Autonomous Agents series) second edition Edition / Gerhard Weiss. - Cambridge:The MIT Press, 2013 - 920 с. - 2nd Edition

ПРИЛОЖЕНИЕ А

Листинг программы

Основная форма ActionWindow.java

```
package salescomp;

import jade.wrapper.StaleProxyException;

import javax.swing.JOptionPane;
import javax.swing.RowFilter;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ActionWindow extends javax.swing.JFrame {

    private static final String URL = "jdbc:mysql://localhost:3306/trade";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    private static Connection con;
    private static Statement stmt;
    private static ResultSet rs;

    DefaultTableModel dm;

    public ActionWindow() throws StaleProxyException {
        initComponents();
        distributorTable.getTableHeader().setResizingAllowed(false);
        distributorTable.getTableHeader().setReorderingAllowed(false);
        tableCreator();
    }


    private int ColumnSelector(){
        int searchBoxIndex = searchType.getSelectedIndex();

        if (searchBoxIndex == 0){
            return 0;
        }else{
            return 2;
        }
    }

    private void filter(String query){
        dm = (DefaultTableModel) distributorTable.getModel();
        TableRowSorter<DefaultTableModel> tr = new TableRowSorter<DefaultTableModel>(dm);

        distributorTable.setRowSorter(tr);
        tr.setRowFilter(RowFilter.regexFilter("(?i)" + query, ColumnSelector()));
    }

    private void tableFiller(){
        DefaultTableModel model = (DefaultTableModel) distributorTable.getModel();
        model.setRowCount(0);

        String query = "SELECT distributor.name, city.name,
product.name,distributor.price_for_product_unit,distributor.product_quantity,
distributor.days_of_delivery "
                    + "FROM distributor,city,product "
```

```
+ "WHERE distributor.city = city.id  and distributor.product = product.id "
                    + "ORDER BY distributor.name";

        try {
            List<String> args = new ArrayList<>();
            con = DriverManager.getConnection(URL, USER, PASSWORD);
            stmt = con.createStatement();
            rs = stmt.executeQuery(query);

            if (rs != null){
                while(rs.next())
                {
                    args.add(rs.getString(1));  //name
                    args.add(rs.getString(2));  //city
                    args.add(rs.getString(3));  //product
                    args.add(rs.getString(4));  //price_for_product_unit
                    args.add(rs.getString(5));  //product_quantity

                    model.addRow(new
Object[]{args.get(0),args.get(1),args.get(2),args.get(3),args.get(4)});
                    args.clear();
                }
            }
        } catch (SQLException sqlEx) {
            sqlEx.printStackTrace();
        } finally {
            try { con.close(); } catch(SQLException se) { /*can't do anything */ }
            try { stmt.close(); } catch(SQLException se) { /*can't do anything */ }
            try { rs.close(); } catch(SQLException se) { /*can't do anything */ }
        }
    }

    private void tableCreator() throws StaleProxyException{
        DefaultTableModel model = (DefaultTableModel) distributorTable.getModel();
        model.setRowCount(0);


            String query = "SELECT distributor.name, city.name,
      product.name,distributor.price_for_product_unit,distributor.product_quantity,
      distributor.days_of_delivery "
                        + "FROM distributor,city,product "
                        + "WHERE distributor.city = city.id  and distributor.product =
      product.id "
                        + "ORDER BY distributor.name";

            try {
                List<String> args = new ArrayList<>();
                String previousDistributor = "";
                List<String> agentArgs = new ArrayList<>();
                con = DriverManager.getConnection(URL, USER, PASSWORD);
                stmt = con.createStatement();
                rs = stmt.executeQuery(query);

                if (rs != null){
                    while(rs.next())
                    {
                        args.add(rs.getString(1));  //name
                        args.add(rs.getString(2));  //city
                        args.add(rs.getString(3));  //product
                        args.add(rs.getString(4));  //price_for_product_unit
                        args.add(rs.getString(5));  //product_quantity

                        model.addRow(new
      Object[]{args.get(0),args.get(1),args.get(2),args.get(3),args.get(4)});

                        args.clear();

                        if((previousDistributor.equalsIgnoreCase(rs.getString(1))) ||
      (previousDistributor.equalsIgnoreCase(""))){}
                        else
                        {
                            Object[] arr = new String[agentArgs.size()];
                            agentArgs.toArray(arr);
```

```
                                if(arr != null){
                                jadeRuntimeChecker();
                                JadeHandler.distributor =
        JadeHandler.agentContainer.createNewAgent(previousDistributor,"salescomp.DistributorAgent
        ",arr);
                                JadeHandler.distributor.start();
                            }
                            agentArgs.clear();
                        }
                        previousDistributor = rs.getString(1);
                        agentArgs.add(rs.getString(3)); //product
                        agentArgs.add(rs.getString(4)); //price for product unit
                        agentArgs.add(rs.getString(6)); //days of delivery
                        agentArgs.add(rs.getString(5)); //product quantity

                        if(rs.isLast()){
                            Object[] arr = new String[agentArgs.size()];
                            agentArgs.toArray(arr);

                            if(arr != null){
                                jadeRuntimeChecker();
                                JadeHandler.distributor =
        JadeHandler.agentContainer.createNewAgent(rs.getString(1),"salescomp.DistributorAgent",ar
        r);
                                JadeHandler.distributor.start();
                            }
                            agentArgs.clear();
                        }
                    }
                }
            } catch (SQLException sqlEx) {
                sqlEx.printStackTrace();
            } finally {
                //close connection ,stmt and resultset here
                try { con.close(); } catch(SQLException se) { /*can't do anything */ }
                try { stmt.close(); } catch(SQLException se) { /*can't do anything */ }
                try { rs.close(); } catch(SQLException se) { /*can't do anything */ }
            }
        }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        jSeparator1 = new javax.swing.JSeparator();
        jMenuItem2 = new javax.swing.JMenuItem();
        searchType = new javax.swing.JComboBox<>();
        searchField = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jScrollPane2 = new javax.swing.JScrollPane();
        distributorTable = new javax.swing.JTable();
        jMenuBar1 = new javax.swing.JMenuBar();
        jMenu1 = new javax.swing.JMenu();
        newOrderMenuItem = new javax.swing.JMenuItem();
        jSeparator2 = new javax.swing.JPopupMenu.Separator();
        exitMenuItem = new javax.swing.JMenuItem();

        jMenuItem2.setText("jMenuItem2");

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Торговая компания ОАО \"Приморское\"");
        setMinimumSize(new java.awt.Dimension(720, 400));
        addWindowFocusListener(new java.awt.event.WindowFocusListener() {
            public void windowGainedFocus(java.awt.event.WindowEvent evt) {
                formWindowGainedFocus(evt);
            }
            public void windowLostFocus(java.awt.event.WindowEvent evt) {
            }
        });

        searchType.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] {
"Поставщик", "Товар" }));

        searchField.setToolTipText("Поиск");
```

```
    searchField.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyReleased(java.awt.event.KeyEvent evt) {
        searchFieldKeyReleased(evt);
    }
});

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 20)); // NOI18N
jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel1.setText("Поставщики");

distributorTable.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "Поставщик", "Город", "Наименование", "Цена за единицу", "Количество"
    }
) {
    boolean[] canEdit = new boolean [] {
        false, false, false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
distributorTable.setAutoscrolls(false);
distributorTable.setEnabled(false);
jScrollPane2.setViewportView(distributorTable);

jMenu1.setText("Файл");
jMenu1.setActionCommand("File");


newOrderMenuItem.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEven
t.VK_O, java.awt.event.InputEvent.ALT_MASK));
        newOrderMenuItem.setText("Сделать заказ");
        newOrderMenuItem.setActionCommand("Order");
        newOrderMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                newOrderMenuItemActionPerformed(evt);
            }
        });
        jMenu1.add(newOrderMenuItem);
        jMenu1.add(jSeparator2);


exitMenuItem.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK
_F4, java.awt.event.InputEvent.ALT_MASK));
        exitMenuItem.setText("Выход");
        exitMenuItem.setActionCommand("exit");
        exitMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                exitMenuItemActionPerformed(evt);
            }
        });
        jMenu1.add(exitMenuItem);

        jMenuBar1.add(jMenu1);

        setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
    .addComponent(jLabel1, javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE,
    700, Short.MAX_VALUE)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(searchType, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)

    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(searchField)))
                    .addContainerGap())
            );
            layout.setVerticalGroup(
                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jLabel1)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(searchType, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(searchField, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jScrollPane2, javax.swing.GroupLayout.DEFAULT_SIZE, 296,
    Short.MAX_VALUE)
                    .addContainerGap())
            );

            pack();
        }// </editor-fold>//GEN-END:initComponents

        private void newOrderMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
    FIRST:event_newOrderMenuItemActionPerformed
            NewOrderWindow nw = new NewOrderWindow();
            nw.setVisible(true);
            nw.setResizable(false);
        }//GEN-LAST:event_newOrderMenuItemActionPerformed

    private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_exitMenuItemActionPerformed
            if("exit".equals(evt.getActionCommand())){
                int dialogButton = JOptionPane.YES_NO_OPTION;
                int result = JOptionPane.showConfirmDialog (null, "Вы действительно хотите
выйти?","Внимание",dialogButton);

                if(result == JOptionPane.YES_OPTION){
                    System.exit(NORMAL);
                }
            }
    }//GEN-LAST:event_exitMenuItemActionPerformed

    private void searchFieldKeyReleased(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_searchFieldKeyReleased
        String query = searchField.getText();
        filter(query);
    }//GEN-LAST:event_searchFieldKeyReleased

    private void formWindowGainedFocus(java.awt.event.WindowEvent evt) {//GEN-
FIRST:event_formWindowGainedFocus
        tableFiller();
    }//GEN-LAST:event_formWindowGainedFocus

    static void jadeRuntimeChecker() throws StaleProxyException{
    if (!JadeHandler.isMainContainerRunning()){
        new JadeHandler();
       }
    }

    /**
     * @param args the command line arguments
     * @throws jade.wrapper.StaleProxyException
     */
```

```
        public static void main(String args[]){
            try {
                for (javax.swing.UIManager.LookAndFeelInfo info :
    javax.swing.UIManager.getInstalledLookAndFeels()) {
                    if ("Nimbus".equals(info.getName())) {
                        javax.swing.UIManager.setLookAndFeel(info.getClassName());
                        break;
                    }
                }
            } catch (ClassNotFoundException ex) {

    java.util.logging.Logger.getLogger(ActionWindow.class.getName()).log(java.util.logging.Le
    vel.SEVERE, null, ex);
            } catch (InstantiationException ex) {

    java.util.logging.Logger.getLogger(ActionWindow.class.getName()).log(java.util.logging.Le
    vel.SEVERE, null, ex);
            } catch (IllegalAccessException ex) {

    java.util.logging.Logger.getLogger(ActionWindow.class.getName()).log(java.util.logging.Le
    vel.SEVERE, null, ex);
            } catch (javax.swing.UnsupportedLookAndFeelException ex) {

    java.util.logging.Logger.getLogger(ActionWindow.class.getName()).log(java.util.logging.Le
    vel.SEVERE, null, ex);
            }

            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
                    try {
                        new ActionWindow().setVisible(true);
                    } catch (StaleProxyException ex) {
                        Logger.getLogger(ActionWindow.class.getName()).log(Level.SEVERE,
    null, ex);
                    }
                }
            });
        }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JTable distributorTable;
    private javax.swing.JMenuItem exitMenuItem;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JPopupMenu.Separator jSeparator2;
    private javax.swing.JMenuItem newOrderMenuItem;
    private javax.swing.JTextField searchField;
    private javax.swing.JComboBox<String> searchType;
    // End of variables declaration//GEN-END:variables
}
```

Форма создания заказа NewOrderWindow.java

```
package salescomp;

import jade.wrapper.StaleProxyException;

import java.awt.event.ActionListener;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.*;

import org.jdesktop.swingx.autocomplete.AutoCompleteDecorator;
```

```
import static salescomp.ActionWindow.jadeRuntimeChecker;

public class NewOrderWindow extends javax.swing.JFrame {

    private static final String URL = "jdbc:mysql://localhost:3306/trade";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    private static Connection con;
    private static Statement stmt;
    private static ResultSet rs;

    public NewOrderWindow() {
        initComponents();

    }

    @SuppressWarnings("unchecked")
    private void initComponents() {
        productNameComboBox = new JComboBox(new Object[]{});
        AutoCompleteDecorator.decorate(productNameComboBox);

        String query = "SELECT product.name "
                     + "FROM product";

        try {
            con = DriverManager.getConnection(URL, USER, PASSWORD);
            stmt = con.createStatement();
            rs = stmt.executeQuery(query);

            if (rs != null){
                while(rs.next())
                {
                    productNameComboBox.addItem(rs.getString(1));
                }
            }
        }catch (SQLException sqlEx) {
            sqlEx.printStackTrace();
        } finally {
                    try { con.close(); } catch(SQLException se) { /*can't do anything */ }
                    try { stmt.close(); } catch(SQLException se) { /*can't do anything */ }
                    try { rs.close(); } catch(SQLException se) { /*can't do anything */ }
                }

                productNameTextField = new javax.swing.JTextField();
                productQuantitySpinner = new javax.swing.JSpinner();
                jLabel1 = new javax.swing.JLabel();
                jLabel2 = new javax.swing.JLabel();
                continueButton = new javax.swing.JButton();

                setTitle("Сделать заказ");
                setMaximumSize(new java.awt.Dimension(450, 131));
                setMinimumSize(new java.awt.Dimension(450, 131));

                productNameTextField.setToolTipText("Введите наименование товара");

                productQuantitySpinner.setModel(new javax.swing.SpinnerNumberModel(1, 1, null,
       1));

                jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
                jLabel1.setText("Наименование товара");

                jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
                jLabel2.setText("Количество");

                continueButton.setText("Продолжить");
                continueButton.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(java.awt.event.ActionEvent evt) {
                        try {
                            continueButtonActionPerformed(evt);
                        } catch (StaleProxyException ex) {
```

```
                    Logger.getLogger(NewOrderWindow.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(productNameComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 260, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 9,
Short.MAX_VALUE)
                .addComponent(productQuantitySpinner,
javax.swing.GroupLayout.PREFERRED_SIZE, 150, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
            .addGroup(layout.createSequentialGroup()
                .addGap(147, 147, 147)
                .addComponent(continueButton)
                .addGap(0, 0, Short.MAX_VALUE))
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGap(129, 129, 129))
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 176,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 178,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
            layout.setVerticalGroup(
                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel1)
                        .addComponent(jLabel2))
                    .addGap(3, 3, 3)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(productQuantitySpinner,
    javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(productNameComboBox,
    javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
    javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(continueButton)))
            );

            pack();
        }

        private void continueButtonActionPerformed(java.awt.event.ActionEvent evt) throws
    StaleProxyException {
            if(productNameComboBox.getSelectedItem().toString() != null){
                int dialogButton = JOptionPane.YES_NO_OPTION;
                int dialogResult = JOptionPane.showConfirmDialog (null, "Вы уверены, что
    хотите заказать "+productNameComboBox.getSelectedItem().toString()+" в количестве
    "+productQuantitySpinner.getValue()+"?","Warning",dialogButton);
                if (dialogResult == JOptionPane.YES_OPTION) { //The ISSUE is here
```

```
                    Object[] args = new Object[2];
                    args[0] = productNameComboBox.getSelectedItem().toString();
                    args[1] = productQuantitySpinner.getValue();
                    jadeRuntimeChecker();
                    JadeHandler.distributor =
    JadeHandler.agentContainer.createNewAgent("customer","salescomp.CustomerAgent",args);
                    JadeHandler.distributor.start();
                }
            }
        }

        private javax.swing.JButton continueButton;
        private javax.swing.JLabel jLabel1;
        private javax.swing.JLabel jLabel2;
        private javax.swing.JTextField productNameTextField;
        private javax.swing.JComboBox productNameComboBox;
        private javax.swing.JSpinner productQuantitySpinner;
    }
```

Класс агента-покупателя CustomerAgent.java

```
    package salescomp;

    import jade.core.Agent;
    import jade.core.behaviours.*;
    import jade.core.AID;
    import jade.domain.DFService;
    import jade.domain.FIPAAgentManagement.*;
    import jade.domain.FIPAException;
    import javax.swing.*;

    import jade.lang.acl.ACLMessage;
    import jade.lang.acl.MessageTemplate;

    import java.awt.Dimension;
    import java.awt.Toolkit;

    public class CustomerAgent extends Agent {

    public static JFrame searchResultFrame;

    private JWindow window;
    private String productTypeName;
    private int productQuantity;
    private final int tickTimer = 10000;

    private AID[] distributorAgent = {};

    @Override
    protected void setup(){
      System.out.println("Hello, Customer-Agent " + getAID().getName() + " is ready!");
      Object[] args = getArguments();

      if (args != null && args.length >0){
          productTypeName = (String) args[0];
          productQuantity = (int) args[1];
          System.out.println("Trying to buy "+productTypeName+ ". Amount:
"+productQuantity);

          window = new JWindow();

          window.setBounds(100, 100, 275, 100);
          Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
          window.setLocation(dim.width/2-window.getSize().width/2, dim.height/2-
window.getSize().height/2);

          JLabel waitMessage = new JLabel("<html><div style='text-align:
center;'>Подождите, пока система подберет наиболее<br/>оптимального поставщика для вашего
заказа</html>",SwingConstants.CENTER);
          window.getContentPane().add(waitMessage);

          window.setVisible(true);
          window.repaint();
          window.validate();
```

```
            addBehaviour(new TickerBehaviour(this,tickTimer)
            {
                @Override
                protected void onTick(){
                  if (window.isVisible()){
                      window.setVisible(false);
                      searchResultFrame = new JFrame();

                      searchResultFrame.setVisible(true);
                      searchResultFrame.setSize(400,150);
                      searchResultFrame.setLocation(dim.width/2-
searchResultFrame.getSize().width/2, dim.height/2-searchResultFrame.getSize().height
                      window.dispose();
                  }
                  System.out.println("Trying to buy "+productTypeName+ ". Amount:
"+productQuantity);

                   DFAgentDescription template = new DFAgentDescription();
                   ServiceDescription sd = new ServiceDescription();
                   sd.setType("type-distribution");

                   template.addServices(sd);
                      try {
                          DFAgentDescription[] result = DFService.search(myAgent,
template);
                          distributorAgent = new AID[result.length];
                          for (int i = 0; i < result.length; ++i) {
                              distributorAgent[i] = result[i].getName();
                              System.out.println(distributorAgent[i].getName());
                          }
                      }
                      catch (FIPAException fe) {
                              fe.printStackTrace();
                      }
                   myAgent.addBehaviour(new RequestPerformer());
                }
            });
        }
        else
        {
            System.out.println("No product type specified!");
            doDelete();
        }
    }

    @Override
    protected void takeDown(){
        System.out.println("Customer-agent " + getAID().getName() +" terminating.");
    }

    private class RequestPerformer extends Behaviour {
        private AID bestSeller;
        private int bestCoeff;
        private int repliesCnt = 0;
        private MessageTemplate mt;
        private int step = 0;

        @Override
        public void action() {
            switch (step) {
                case 0:
                    ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
                    for (int i = 0; i < distributorAgent.length; ++i) {
                        cfp.addReceiver(distributorAgent[i]);
                    }
                    cfp.setContent(productTypeName+"-"+productQuantity);
                    cfp.setConversationId("type-distribution");
                    cfp.setReplyWith("cfp"+System.currentTimeMillis());
```

```
                    myAgent.send(cfp);

                    mt = MessageTemplate.and(MessageTemplate.MatchConversationId("type-
distribution"),
                    MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
                    step = 1;
                break;
                case 1:
                    ACLMessage reply = myAgent.receive(mt);
                    if (reply != null) {
                        // Получение ответа
                        if (reply.getPerformative() == ACLMessage.PROPOSE) {
                            Object[] offer = reply.getContent().split("-");

                            int price = Integer.parseInt(offer[0].toString());
                            int days = Integer.parseInt(offer[1].toString());
                            int quantity = Integer.parseInt(offer[2].toString());
                            int coeff = price*quantity*days;

                            if (bestSeller == null || coeff < bestCoeff) {
                                bestCoeff = coeff;
                                bestSeller = reply.getSender();
                            }
                        }
                        repliesCnt++;
                        if (repliesCnt >= distributorAgent.length) {
                            step = 2;
                        }
                    }
                    else {

                        block();
                    }
                break;
                case 2:
                    ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);

                            order.addReceiver(bestSeller);
                            order.setContent(productTypeName+"-"+productQuantity);
                            order.setConversationId("type-distribution");
                            order.setReplyWith("order"+System.currentTimeMillis());

                            myAgent.send(order);

                            mt = MessageTemplate.and(MessageTemplate.MatchConversationId("type-
        distribution"),
                            MessageTemplate.MatchInReplyTo(order.getReplyWith()));

                            step = 3;
                        break;
                        case 3:
                            reply = myAgent.receive(mt);
                            if (reply != null) {
                                if (reply.getPerformative() == ACLMessage.INFORM) {
                                    System.out.println(productTypeName+" successfully
        purchased.");
                                    myAgent.doDelete();
                                }
                            step = 4;
                            }
                            else {
                            block();
                            }
                        break;
                    }
                }
                @Override
                public boolean done() {
                    return ((step == 2 && bestSeller == null) || step == 4);
                }
            }
        }
```

Класс агента-продавца DistributorAgent.java

```
package salescomp;

import jade.core.Agent;
import jade.core. behaviours.*;

import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;

import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import java.util.*;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import static salescomp.CustomerAgent.searchResultFrame;

public class DistributorAgent extends Agent {
    private final List<Object[]> catalogue = new ArrayList<>();
    private static final String URL = "jdbc:mysql://localhost:3306/trade";
    private static final String USER = "root";
    private static final String PASSWORD = "";

    private static Connection con;
    private static Statement stmt;

    private String agentName;
    @Override
    protected void setup() {
        System.out.println("Hello. Distributor-Agent "+getAID().getName()+" is ready.");

        agentName = getName().substring(0,getName().length()-24);


Object[] args;
        args = getArguments();

        if((args.length/4)>1){
            for (int k=0; k<(args.length/4);k++){
                updateCatalogue((String) args[k*4], Integer.parseInt((String)
args[1+k*4]),Integer.parseInt((String) args[2+k*4]),Integer.parseInt((String)
args[3+k*4]));
            }
        }else{
            updateCatalogue((String) args[0], Integer.parseInt((String)
args[1]),Integer.parseInt((String) args[2]),Integer.parseInt((String) args[3]));
        }

         // Register the service in the yellow pages
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType("type-distribution");
        sd.setName("JADE-type-distribution");
        dfd.addServices(sd);
        try {
            DFService.register(this, dfd);
        }
        catch (FIPAException fe) {
            fe.printStackTrace();
        }

        addBehaviour(new OfferRequestsServer());
        addBehaviour(new PurchaseOrdersServer());
    }
```

```
@Override
    protected void takeDown() {
        try {
            DFService.deregister(this);
        }
        catch (FIPAException fe) {
            fe.printStackTrace();
        }
        System.out.println("Distributor-agent "+getAID().getName()+" terminating.");
    }

    public void updateCatalogue(final String name, final int price, final int days, final
int quantity) {
        addBehaviour(new OneShotBehaviour() {
            public void action() {
                Object[] args = {name,price,days,quantity};
                if (args!=null){
                    catalogue.add(args);
                    System.out.println(name+" added. Price: "+price+" Days of delivery:
"+days+" Avaliable quantity: "+quantity);
                }
            }
        });
    }

    private class OfferRequestsServer extends CyclicBehaviour {
        @Override
        public void action() {
            MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
            ACLMessage msg = myAgent.receive(mt);
                if (msg != null) {
                    //do stuff
                    Object[] offer = msg.getContent().split("-");

                    String offerTitle = offer[0].toString();
                    int offerQuantity = Integer.parseInt(offer[1].toString());
                    int price = 0,days = 0,quantity = 0;
                    boolean isFound = false;


                       ACLMessage reply = msg.createReply();

                       if(catalogue!=null){
                           for (int i=0;i<catalogue.size();i++){
                               Object[] o = catalogue.get(i);
                               String name = o[0].toString();

                               if
    ((offerTitle.equalsIgnoreCase(name))&&(offerQuantity<=(int)o[3]))
                               {
                                   System.out.println();
                                   price = (int)o[1];
                                   days = (int)o[2];
                                   quantity = (int)o[3];
                                   isFound = true;
                                   break;
                               }
                           }
                           if (isFound) {
                               System.out.println("Sending proposal");
                               reply.setPerformative(ACLMessage.PROPOSE);
                               reply.setContent(price+"-"+days+"-"+quantity);
                           }
                           else{
                               searchResultFrame.add(new JLabel("<html><div style='text-
    align: center;'>Подходящего поставщика не найдено.
    Подождите.</html>",SwingConstants.CENTER));
                               searchResultFrame.setResizable(false);
                               searchResultFrame.repaint();
                               searchResultFrame.validate();

                           }
                       }
```

```
else {
                        reply.setPerformative(ACLMessage.REFUSE);
                        reply.setContent("not-available");
                    }
                    myAgent.send(reply);
                }
            }
        }

    private class PurchaseOrdersServer extends CyclicBehaviour {
        @Override
        public void action() {
            MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
            ACLMessage msg = myAgent.receive(mt);
            if (msg != null) {
                Object[] offer = msg.getContent().split("-");

                String title = offer[0].toString();
                int offerQuantity = Integer.parseInt(offer[1].toString());

                ACLMessage reply = msg.createReply();

                boolean isFound = false;

                if(catalogue!=null){
                    for (int i=0;i<catalogue.size();i++){
                        Object[] o = catalogue.get(i);
                        String name = o[0].toString();

                        if (title.equalsIgnoreCase(name)){
                            isFound = true;
                            o[3] = (int)o[3] - offerQuantity;
                            System.out.println(o[3]);

                                String query = "UPDATE distributor "
                                        + "SET distributor.product_quantity = '"+o[3]+"'
        "
                                        + "WHERE distributor.name = '"+agentName+"' and
        distributor.product = "
                                        + "(SELECT product.id FROM product WHERE
        product.name ='"+title+"') ";

                                try {
                                    con = DriverManager.getConnection(URL +
        "?useUnicode=true&characterEncoding=UTF8",USER, PASSWORD);
                                    stmt = con.createStatement();

                                    stmt.executeUpdate(query);
                                }
                                catch (SQLException sqlEx) {
                                    sqlEx.printStackTrace();
                                } finally {
                                    try { con.close(); } catch(SQLException se) { /*can't do
        anything */ }
                                    try { stmt.close(); } catch(SQLException se) { /*can't do
        anything */ }
                                }

                                searchResultFrame.add(new JLabel("<html><div style='text-
        align: center;'>На ваш заказ подобран поставщик - "+agentName+".<br/> Заказ будет
        доставлен в течении следующего количества дней: "+o[2].toString()+".<br/>"
                                        + "Суммарная стоимость заказа:
        "+(int)o[1]*offerQuantity+" рублей.</html>",SwingConstants.CENTER));
                                searchResultFrame.setResizable(false);
                                searchResultFrame.repaint();
                                searchResultFrame.validate();

                                if ((int)o[3] == 0){
                                    catalogue.remove(i);
                                }
```

```
            }
                                }
                            }

                            if (isFound) {
                                reply.setPerformative(ACLMessage.INFORM);
                                System.out.println(title+" sold to agent
        "+msg.getSender().getName()+" by "+getName());
                            }
                            else {
                                reply.setPerformative(ACLMessage.FAILURE);
                                reply.setContent("not-available");
                            }
                            myAgent.send(reply);
                        }
                        else {
                            block();
                        }
                    }
                }
            }
```

Класс запуска JADE JadeHandler.java

```
        package salescomp;

        import jade.core.ProfileImpl;
        import jade.core.Runtime;
        import jade.wrapper.AgentContainer;
        import jade.wrapper.AgentController;
        import jade.wrapper.StaleProxyException;

        public class JadeHandler {

            public static Runtime rt;

            public static AgentContainer agentContainer;
            public static AgentController distributor = null;


    ProfileImpl p = new ProfileImpl(false);
    public JadeHandler() throws StaleProxyException{
        rt = Runtime.instance();
        JadeHandler.agentContainer = rt.createMainContainer(p);

    }

    public static boolean isMainContainerRunning(){
        boolean isRunning;
        try {
                agentContainer.getState();
                isRunning = true;

         } catch (Exception eMC) {
                isRunning = false; //
                agentContainer = null;
                try {
                   Runtime.instance().shutDown();
                } catch (Exception ex) {
                }
         }
        return isRunning;
    }

    public static void jadeShutdown(){
        rt.shutDown();
    }
}
```