МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Российский государственный гуманитарный университет»
(ФГБОУ ВО «РГГУ»)

ИНСТИТУТ ИНФОРМАЦИОННЫХ НАУК И ТЕХНОЛОГИЙ БЕЗОПАСНОСТИ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ СИСТЕМ И БЕЗОПАСНОСТИ
Кафедра информационных технологий и систем

МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ В ГУМАНИТАРНОЙ СФЕРЕ

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

09.03.03 Прикладная информатика
Код и наименование направления подготовки

Прикладная информатика в гуманитарной сфере
Наименование направленности (профиля)

Уровень высшего образования: *бакалавриат*

Форма обучения: *очная*

РПД адаптирована для лиц
с ограниченными возможностями
здоровья и инвалидов

Москва 2023

Мультиагентные системы в гуманитарной сфере
Рабочая программа дисциплины

Составитель(и):
старший преподаватель кафедры информационных технологий и систем Е.П. Охапкина

Ответственный редактор
канд. с.-х. наук, доцент, заведующий кафедрой
информационных технологий и систем Н.Ш. Шукенбаева

УТВЕРЖДЕНО
Протокол заседания кафедры ИТС
№ 8 от 15.04.2023 г.

ОГЛАВЛЕНИЕ

1. Пояснительная записка

1.1. Цель и задачи дисциплины

Цель дисциплины: приобретение знаний, навыков и умений в области проектирования и разработки мультиагентных систем.

Задачи: сформировать у студентов понятия о роли и месте мультиагентного подхода к решению задач в области информатизации и автоматизации систем управления, о его достоинствах и ограничениях. Сформировать знания об основных видах агентных архитектур и стратегиях управления мультиагентными коллективами и познакомить с ними на практике. Предоставить информацию о назначении и основных характеристиках существующих мультиагентных систем и их функциональных возможностях. Сформировать у студентов навыки самостоятельной разработки мультиагентных систем.

1.2. Перечень планируемых результатов обучения по дисциплине, соотнесенных с индикаторами достижения компетенций

Компетенция	Индикаторы компетенций	Результаты обучения
ПК-3 Способен проектировать информационные системы по видам обеспечения	ПК-3.1 Знает модели жизненного цикла информационных систем, основные технологии, стадии и этапы их проектирования	**знать** методологию, методы и модели формирования МАС; о базовых ситуациях, режимах и моделях взаимодействия, коммуникации, кооперации; программные языки и инструментальные средства реализации искусственных агентов.
	ПК-3.2 Умеет применять технологии проектирования информационных систем по видам обеспечения	**уметь** осуществлять синтез искусственных агентов различных классов и выбор эффективных архитектур МАС для конкретных, специфических приложений
	ПК-3.3 Владеет навыками проектирования информационных систем или их частей по видам обеспечения	**владеть** методами моделирования поведения и действий агентов.
ПК-6 Способен настраивать, эксплуатировать и сопровождать информационные системы и сервисы	ПК-6.1 Знает методы настройки, порядок и мероприятия по эксплуатации и сопровождению информационных систем и сервисов	**знать** общие принципы построения, основные свойства и архитектуры автономных агентов

	ПК-6.2 Умеет организовывать настройку, эксплуатацию и сопровождение информационных систем и сервисов	**уметь** программировать агентов с использованием языков ориентированного программирования, библиотек агентов и агентских сред; разработки структур коммуникации агентов на основе стандарта ACL (Agents Communication); применять восходящее и нисходящее проектирование МАС.
	ПК-6.3 Владеет навыками управления конфигурацией ИС и сервисов в процессе эксплуатации, решения проблем и консультирования пользователей информационных систем и сервисов	**владеть** вопросами о причинах появления и основных направлениях развития теории агентов и МАС как стратегической области информатики и искусственного интеллекта; важнейшими способами разработки агентов (поведенческая, деятельностная, логическая, лингвистическая, программистская и пр.) и формализмах описания мультиагентных систем различных классах

1.3. Место дисциплины в структуре образовательной программы

Дисциплина «Мультиагентные системы в гуманитарной сфере» относится к части, формируемой участниками образовательных отношений блока дисциплин учебного плана.

Для освоения дисциплины необходимы знания, умения и владения, сформированные в ходе изучения дисциплин Программирование C#, Программирование C++, Архитектура вычислительных систем.

В результате освоения дисциплины формируются знания, умения и владения, необходимые для изучения следующих дисциплин и прохождения практик: Методы информационного поиска в задачах информатизации гуманитарной сферы, Информационно-поисковые системы и машины в гуманитарной сфере.

2. Структура дисциплины

Общая трудоёмкость дисциплины составляет 3 з.е., 108 академических часов.

Структура дисциплины для очной формы обучения

Объем дисциплины в форме контактной работы обучающихся с педагогическими работниками и (или) лицами, привлекаемыми к реализации образовательной программы на иных условиях, при проведении учебных занятий:

Семестр	Тип учебных занятий	Количество часов
4	Лекции	16
4	Практические занятия	26
	Всего:	42

Объем дисциплины (модуля) в форме самостоятельной работы обучающихся составляет 66 академических часов.

3. Содержание дисциплины

№	Наименование раздела дисциплины	Содержание
1.	Тема 1. Введение в многоагентные системы	Понятие агента. Окружение агентов. Архитектура агентов. Архитектуры, основанные на знаниях. Архитектуры на основе планирования. Языки программирования агентов: требования, классификация. Многоагентные системы. Коммуникация между агентами.
2.	Тема 2. Архитектура многоагентных систем	Архитектура взаимодействия системы агентов: одноуровневая архитектура взаимодействия агентов, иерархическая архитектура взаимодействия агентов. Протоколы и языки координации. Проблемы построения многоагентных систем.
3.	Тема 3. Мультиагентное моделирование	Язык программирования Java и библиотека Jade. Примеры реализации мультиагентных систем. Постановка задачи. Концептуальный анализ поведения агентов.

4. Образовательные технологии

Для проведения учебных занятий по дисциплине используются различные образовательные технологии. Для организации учебного процесса может быть использовано электронное обучение и (или) дистанционные образовательные технологии.

5. Оценка планируемых результатов обучения

5.1 Система оценивания

Форма контроля	**Макс. количество баллов**	
	За одну работу	**Всего**
Текущий контроль: Практическая работа № 1, защита отчета Практическая работа № 2, защита отчета Практическая работа № 3, защита отчета	 20 баллов 20 баллов 20 баллов	 20 баллов 20 баллов 20 баллов
Промежуточная аттестация *зачет с оценкой*		40 баллов
Итого за семестр		100 баллов

Полученный совокупный результат конвертируется в традиционную шкалу оценок и в шкалу оценок Европейской системы переноса и накопления кредитов (European Credit Transfer System; далее – ECTS) в соответствии с таблицей:

100-балльная шкала	Традиционная шкала		Шкала ECTS
95 – 100	отлично	зачтено	A
83 – 94			B
68 – 82	хорошо		C
56 – 67	удовлетворительно		D
50 – 55			E
20 – 49	неудовлетворительно	не зачтено	FX
0 – 19			F

5.2 Критерии выставления оценки по дисциплине

Баллы/ Шкала ECTS	Оценка по дисциплине	Критерии оценки результатов обучения по дисциплине
100-83/ A,B	отлично/ зачтено	Выставляется обучающемуся, если он глубоко и прочно усвоил теоретический и практический материал, может продемонстрировать это на занятиях и в ходе промежуточной аттестации. Обучающийся исчерпывающе и логически стройно излагает учебный материал, умеет увязывать теорию с практикой, справляется с решением задач профессиональной направленности высокого уровня сложности, правильно обосновывает принятые решения. Свободно ориентируется в учебной и профессиональной литературе. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции, закреплённые за дисциплиной, сформированы на уровне – «высокий».
82-68/ C	хорошо/ зачтено	Выставляется обучающемуся, если он знает теоретический и практический материал, грамотно и по существу излагает его на занятиях и в ходе промежуточной аттестации, не допуская существенных неточностей. Обучающийся правильно применяет теоретические положения при решении практических задач профессиональной направленности разного уровня сложности, владеет необходимыми для этого навыками и приёмами. Достаточно хорошо ориентируется в учебной и профессиональной литературе. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции, закреплённые за дисциплиной, сформированы на уровне – «хороший».
67-50/ D,E	удовлетво-рительно/ зачтено	Выставляется обучающемуся, если он знает на базовом уровне теоретический и практический материал, допускает отдельные ошибки при его изложении на занятиях и в ходе промежуточной аттестации. Обучающийся испытывает определённые затруднения в применении теоретических положений при решении практических задач профессиональной направленности стандартного уровня сложности, владеет необходимыми для этого базовыми навыками и приёмами. Демонстрирует достаточный уровень знания учебной литературы по дисциплине. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции, закреплённые за дисциплиной, сформированы на уровне – «достаточный».
49-0/ F,FX	неудовлет-ворительно/ не зачтено	Выставляется обучающемуся, если он не знает на базовом уровне теоретический и практический материал, допускает грубые ошибки при его изложении на занятиях и в ходе промежуточной аттестации. Обучающийся испытывает серьёзные затруднения в применении теоретических положений при решении практических задач профессиональной направленности стандартного уровня сложности, не владеет необходимыми для этого навыками и приёмами. Демонстрирует фрагментарные знания учебной литературы по дисциплине. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции на уровне «достаточный», закреплённые за дисциплиной, не сформированы.

5.3 Оценочные средства (материалы) для текущего контроля успеваемости, промежуточной аттестации обучающихся по дисциплине

Контрольные вопросы экзамена

1. Приведите несколько наиболее емких определений агента.
2. Перечислите свойства интеллектуального агента.
3. Как вы понимаете необходимость для агента сохранения баланса между внутренним и внешним окружением и какими действиями этот баланс может быть достигнут?

4. Понятия реактивности и проактивности. Возможно ли сохранить разумный баланс между ними?
5. Приведите наиболее общую классификацию агентов согласно Stan Franklin and Art Graesser.
6. Каковы характеристики "интеллектуального" агента?
7. Что означает понятие "целеориентированный" агент?
8. Опишите взаимодействие агента с внешней средой при условии, что этот агент хранит историю своих состояний, приведите схему.
9. Достоинства и недостатки рефлективной, делиберативной (логической) и гибридной агентной архитектуры.
10. Архитектура агента Belief-Desire-Intention. Опишите последовательность его действий.
11. Уровневые архитектуры потоков событий агентов, перечислить, охарактеризовать.
12. Понятие МАС, требования к Мультиагентным системам (МАС).
13. Назовите наиболее известные стандарты для создания МАС и их особенности.
14. Назовите основные объекты для стандартизации в МАС.
15. Перечислить и кратко охарактеризовать известные вам агентные платформы.
16. Модель агентной платформы. Возможности агентных платформ.
17. Платформа FIPA-OS, структура, характеристики.
18. Структура и характеристики платформы JADE.
19. Изобразить архитектуру агента в JADE, привести описание его компонентов.
20. Описать типы поведения агента в JADE.
21. Перечислить и охарактеризовать языки коммуникации агентов, привести пример.
22. Язык ACL и сервис обмена сообщениями в JADE.
23. Специальные агенты в JADE, их назначение и специфика работы.
24. Привести блок-схему жизненного цикла агента в JADE.
25. Перечислите и детализируйте свойства внешней среды агента.
26. Привести схему простого реактивного агента с пояснениями.
27. Привести схему агента с целью с пояснениями.
28. В чем разница между логическим целеориентированным и ориентированным на выгоду агентами?
29. Привести схему обучающегося агента с пояснениями, перечислить его достоинства и недостатки.
30. Назовите последовательность шагов агента, решающего задачи.

31. Какими способами может быть выполнен поиск решения в пространстве состояний?
32. Как обычно оценивается производительность при решении задачи поиска в пространстве состояний?
33. Логика первого порядка в рассуждениях агента. Атомарные высказывания, предложения – привести примеры.
34. Как формально должна быть построена задача в логике первого порядка и как затем выглядит процесс получения решения?
35. Интеллектуальные способы планирования решения задач в мультиагентных системах.
36. Определение онтологии. Предназначение онтологий в МАС.
37. Структура онтологий, где онтологии применяются.

6. Учебно-методическое и информационное обеспечение дисциплины

6.1 Список источников и литературы

Источники
Основные

1. Федеральный закон Российской Федерации от 27 июля 2006 г. N 149-ФЗ «Об информации, информационных технологиях и о защите информации».
2. ГОСТ 34.003-90. Автоматизированные системы. Термины и определения.

3. ГОСТ 34.201-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем.

4. ГОСТ 34.601-90. Автоматизированные системы. Комплекс стандартов на автоматизированные системы. Стадии создания.

Литература

Основная

1. Федунец Н. И. Алгоритмы формирования управляющих воздействий в распределенных мультиагентных системах[Текст] / Н. И. Федунец, М. А. Приходько // Программные продукты и системы. - 2010. - N 4. - С. 81-84. - Библиогр.: с. 84 (2 назв.). - ил.: 1 рис.
2. Лихачев В. Е. Модель мультиагента прогнозирования автоматизированных систем управления / В. Е. Лихачев // Программные продукты и системы. - 2010. - N 1. - С. 95-98. - Библиогр.: с. 98 (2 назв.).
3. Мультиагентная система распределения производственных ресурсов в тяжелом машиностроении[Текст] / М. В. Андреев [и др.] // Программные продукты и системы. - 2010. - N 3. - С. 100-104. - Библиогр.: с. 104 (4 назв.). - ил.: 3 рис.
4. Лавренков Ю. Н. Анализ характеристик канала передачи информации на основе нейронной сети [Текст] / Ю. Н. Лавренков, Л. Г. Комарцова // Прикладная информатика. - 2014. - № 3 (51). - С. 79-99. - Библиогр.: с. 99 (17 назв.). - 18 рис.
5. Колмыков В. В. Сравнительный анализ статистической модели и нейронной сети обратного распространения в задаче прогнозирования[Текст] / В. В. Колмыков // Прикладная информатика. - 2010. - N 6 (30). - С. 111-119. - Библиогр.: с. 119 (1 назв.). - 3 табл.

Дополнительная

1. Гаранина Н. О. (Институт систем информатики им. А. П. Ершова СО РАН). Экспоненциальное улучшение временной сложности проверки моделей для мультиагентных систем с абсолютной памятью [Текст] / Н. О. Гаранина // Программирование. - 2012. - № 6. - С. 20-31. - Библиогр.: с. 31 (20 назв.).
2. Лебедев П. В. VMASTER – среда для разработки и верификации вероятностных мультиагентных систем[Текст] / П. В. Лебедев // Программные продукты и системы. - 2013. - № 2. - С. 201-207. - Библиогр.: с. 207 (9 назв.). - ил.: 2 табл. Ссылка на ресурс: http://www.swsys.ru/index.php?page=article&id=3494.
3. Зайцев И. Д. Верификация мультиагентных систем с помощью цепей Маркова: оценка вероятности нахождения агентами оптимального решения[Текст] / И. Д. Зайцев // Программные продукты и системы.-2013.-№ 4.-С. 96-100. - Библиогр.: с. 100 (9 назв.).Ссылка на ресурс: http://www.swsys.ru/index.php?page=article&id=3664
4. Беркинблит Михаил Борисович. Нейронные сети: экспериментальное учебное пособие / М. Б. Беркинблит. - Москва : МИРОС, 1993. - 95, [1] с. : рис. ; 20 см. - Библиогр. в конце кн. - ISBN 5-7084-0026-9 : 150.00.
5. Иванов А. И. Нейросетевая защита биометрических данных пользователя, а также его личного криптографического ключа при локальной и дистанционной аутентификации / А. И. Иванов, В. А. Фунтиков, О. В. Ефимов // Вопросы защиты информации. - 2008. - N 2. - С. 25-28. - Библиогр.: с. 10-11 (10 назв.).
6. Сидоркина И. Г. Прогнозирование на основе выделения границы реализаций динамических систем[Текст] / И. Г. Сидоркина, А. В. Егошин, Д. С. Шумков // Программные продукты и системы. - 2010. - N 4. - С. 117-119. - Библиогр.: с. 119 (4 назв.).

7. Бугорский В. Н. Использование нейронных сетей в работе трейдера[Текст] / В. Н. Бугорский, А. Г. Сергиенко // Прикладная информатика. - 2011. - N 1 (31). - С. 17-28. - Библиогр.: с. 28 (7 назв.). - 4 рис.

6.2 Перечень ресурсов информационно-телекоммуникационной сети «Интернет».

Национальная электронная библиотека (НЭБ) www.rusneb.ru
ELibrary.ru Научная электронная библиотека www.elibrary.ru
Электронная библиотека Grebennikon.ru www.grebennikon.ru
Cambridge University Press

6.3 Профессиональные базы данных и информационно-справочные системы

Доступ к профессиональным базам данных: https://liber.rsuh.ru/ru/bases

Информационные справочные системы:
1. Консультант Плюс
2. Гарант

7. Материально-техническое обеспечение дисциплины

№п/п	Наименование специальных помещений и помещений для самостоятельной работы	Оснащенность специальных помещений и помещений для самостоятельной работы	Перечень лицензионного программного обеспечения. Реквизиты подтверждающего документа		
			Наименование ПО	Лицензия/сертификат/заказ	Дата лицензии
1.	Лаборатория информатики – ауд. № 203	1 компьютер преподавателя, 12 компьютеров обучающихся, маркерная доска, проектор	Windows 7 Microsoft office 2010 Pro Microsoft Visual Professional 2019 Mozilla Firefox 52.8.1 ESR Matlab Mathcad Education - University edition Kaspersky Endpoint Security	68526624 49420326 63202190 свободный доступ 647526 2996385 17E0-181226-094912-873-979	без даты 08.12.2011 без даты свободный доступ без даты 14.06.2019 26.12.2018

8. Обеспечение образовательного процесса для лиц с ограниченными возможностями здоровья и инвалидов

В ходе реализации дисциплины используются следующие дополнительные методы обучения, текущего контроля успеваемости и промежуточной аттестации обучающихся в зависимости от их индивидуальных особенностей:

- для слепых и слабовидящих: лекции оформляются в виде электронного документа, доступного с помощью компьютера со специализированным программным обеспечением; письменные задания выполняются на компьютере со специализированным программным обеспечением или могут быть заменены устным ответом; обеспечивается индивидуальное равномерное освещение не менее 300 люкс; для выполнения задания при необходимости

предоставляется увеличивающее устройство; возможно также использование собственных увеличивающих устройств; письменные задания оформляются увеличенным шрифтом; экзамен и зачёт проводятся в устной форме или выполняются в письменной форме на компьютере.

- для глухих и слабослышащих: лекции оформляются в виде электронного документа, либо предоставляется звукоусиливающая аппаратура индивидуального пользования; письменные задания выполняются на компьютере в письменной форме; экзамен и зачёт проводятся в письменной форме на компьютере; возможно проведение в форме тестирования.
- для лиц с нарушениями опорно-двигательного аппарата: лекции оформляются в виде электронного документа, доступного с помощью компьютера со специализированным программным обеспечением; письменные задания выполняются на компьютере со специализированным программным обеспечением; экзамен и зачёт проводятся в устной форме или выполняются в письменной форме на компьютере.

При необходимости предусматривается увеличение времени для подготовки ответа.

Процедура проведения промежуточной аттестации для обучающихся устанавливается с учётом их индивидуальных психофизических особенностей. Промежуточная аттестация может проводиться в несколько этапов.

При проведении процедуры оценивания результатов обучения предусматривается использование технических средств, необходимых в связи с индивидуальными особенностями обучающихся. Эти средства могут быть предоставлены университетом, или могут использоваться собственные технические средства.

Проведение процедуры оценивания результатов обучения допускается с использованием дистанционных образовательных технологий.

Обеспечивается доступ к информационным и библиографическим ресурсам в сети Интернет для каждого обучающегося в формах, адаптированных к ограничениям их здоровья и восприятия информации:

- для слепых и слабовидящих: в печатной форме увеличенным шрифтом, в форме электронного документа, в форме аудиофайла.
- для глухих и слабослышащих: в печатной форме, в форме электронного документа.
- для обучающихся с нарушениями опорно-двигательного аппарата: в печатной форме, в форме электронного документа, в форме аудиофайла.

Учебные аудитории для всех видов контактной и самостоятельной работы, научная библиотека и иные помещения для обучения оснащены специальным оборудованием и учебными местами с техническими средствами обучения:

- для слепых и слабовидящих: устройством для сканирования и чтения с камерой SARA CE; дисплеем Брайля PAC Mate 20; принтером Брайля EmBraille ViewPlus;
- для глухих и слабослышащих: автоматизированным рабочим местом для людей с нарушением слуха и слабослышащих; акустический усилитель и колонки;
- для обучающихся с нарушениями опорно-двигательного аппарата: передвижными, регулируемыми эргономическими партами СИ-1; компьютерной техникой со специальным программным обеспечением.

9. Методические материалы

9.1 Планы практических занятий

Практическая работа № 1

Установка IDE NetBeans, подключение библиотеки JADE, создание простого агентного приложения

Постановка задачи:

Создать простое агентное приложение с использованием JADE

Задачи:

1. Познакомиться с понятием агента
2. Познакомиться с интегрированными средами разработки (IDE) NetBeans, Eclipse, IntelliJ IDEA. Поместить обзоры по данным IDE в отчет по лабораторной работе.
3. Установить ПО NetBeans, Jade, JDK
4. Подключить библиотеку JADE к интегрированной среде разработке NetBeans
5. Создать 2х агентов на платформе JADE
6. Подготовить в печатном и электронном виде отчет по лабораторной работе №1 (обязательно наличие: титульного листа, цели, задач, описания IDE, хода выполнения работы, выводов).
7. При составлении отчета привести комментарии ко всем строкам кода.

Установка ПО:

Для создания агента понадобится многоагентная система **JADE, Java Developer Kit** и IDE в которой будем разрабатывать приложение – **NetBeans.** JDK можно скачать сразу вместе с NetBeans.

Сначала необходимо распаковать JADE в любую папку (например на флешку или в свою паку в Моих документах). Теперь мы можем создать своего первого агента на платформе Java Agent Development Framework. Сначала необходимо создать Java-проект нашего агента в среде NetBeans и подключить JADE к этому проекту.

Подключение JADE:

Создаем JAVA проект, который назовем, к примеру, JavaAgent. При создании проекта не забудьте выключить галочку на **Create Main Class**, это отключит создание главного класса, т.к. мы будем создавать все классы вручную.

Создаем комплект библиотек для подключение через меню Service(или Tools) – Libraries, далее «New Library» т.е. создаем свой набор библиотек, назовем его, к примеру, «Jade». Далее добавляем библиотеки JADE кнопочкой «Add JAR/Folder» и указываем путь к библиотекам JADE. Это четыре библиотеки: http.jar, iiop.jar, jade.jar, jadeTools.jar.

Затем подключаем их к проекту, для этого в свойствах проекта выбираем libraries и выбираем «Add Library…» Результат действий:

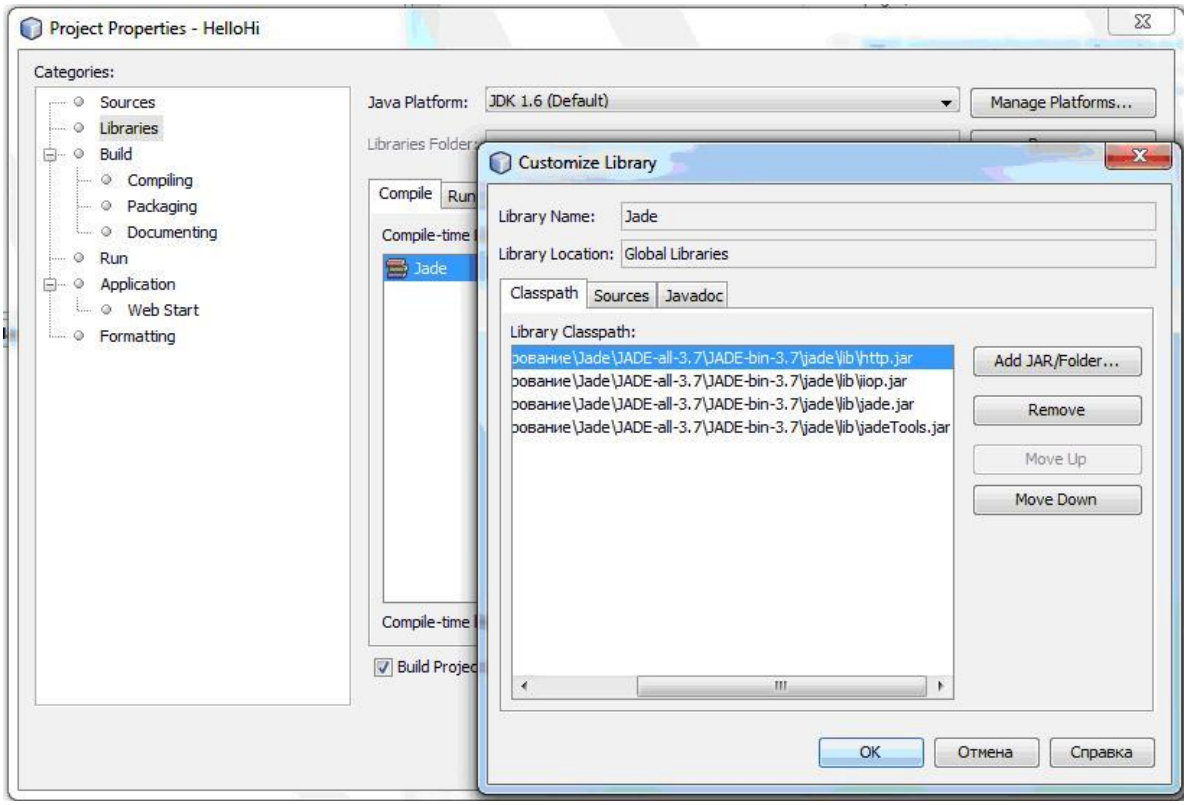


Рис. 1. Подключение библиотеки JADE

Разработка Агентов:

Создадим простейший пример многоагентной системы, где будут 2 агента, один посылает сообщение, второй отвечает на это сообщение.

Теперь необходимо создать первого агента (Агент А), который будет отвечать на запрос, для этого щелкаем правой кнопкой мыши по Source Packages и выбираем New -> Java Packages, назовем его A, теперь создаем в этом пакете Java Class - его назовем AMain. Делаем тоже самое и для Агента B (Пакет B, Java-класс BClass)

На данный момент пока просто вставьте код для AMain и BClass, который указан в приложениях к лабораторной работе. Перед тем как запустить нашего агента необходимо настроить параметры запуска проекта. Это можно сделать выбрав в свойствах проекта в свойстве «Run».

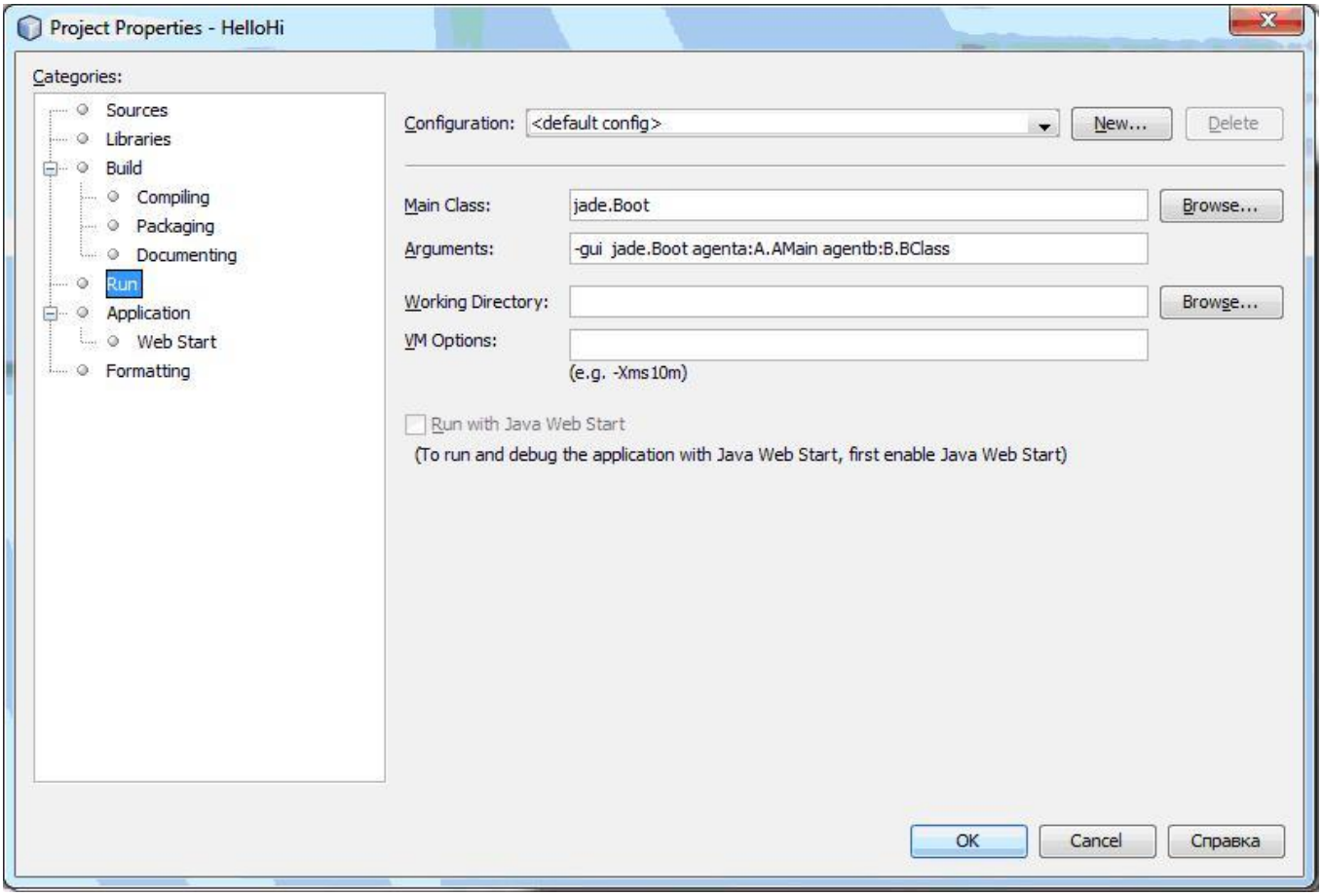


Рис. 2. Параметры запуска агентов

Для этого вписываем главный класс как jade.Boot, а в аргументах указываем GUI системы jade и те агенты которые запускаем, agenta из пакета A, класса AMain и agentb из пакета B класса BClass В итоге получаем (-gui jade.Boot agenta:A.AMain agentb:B.BClass):

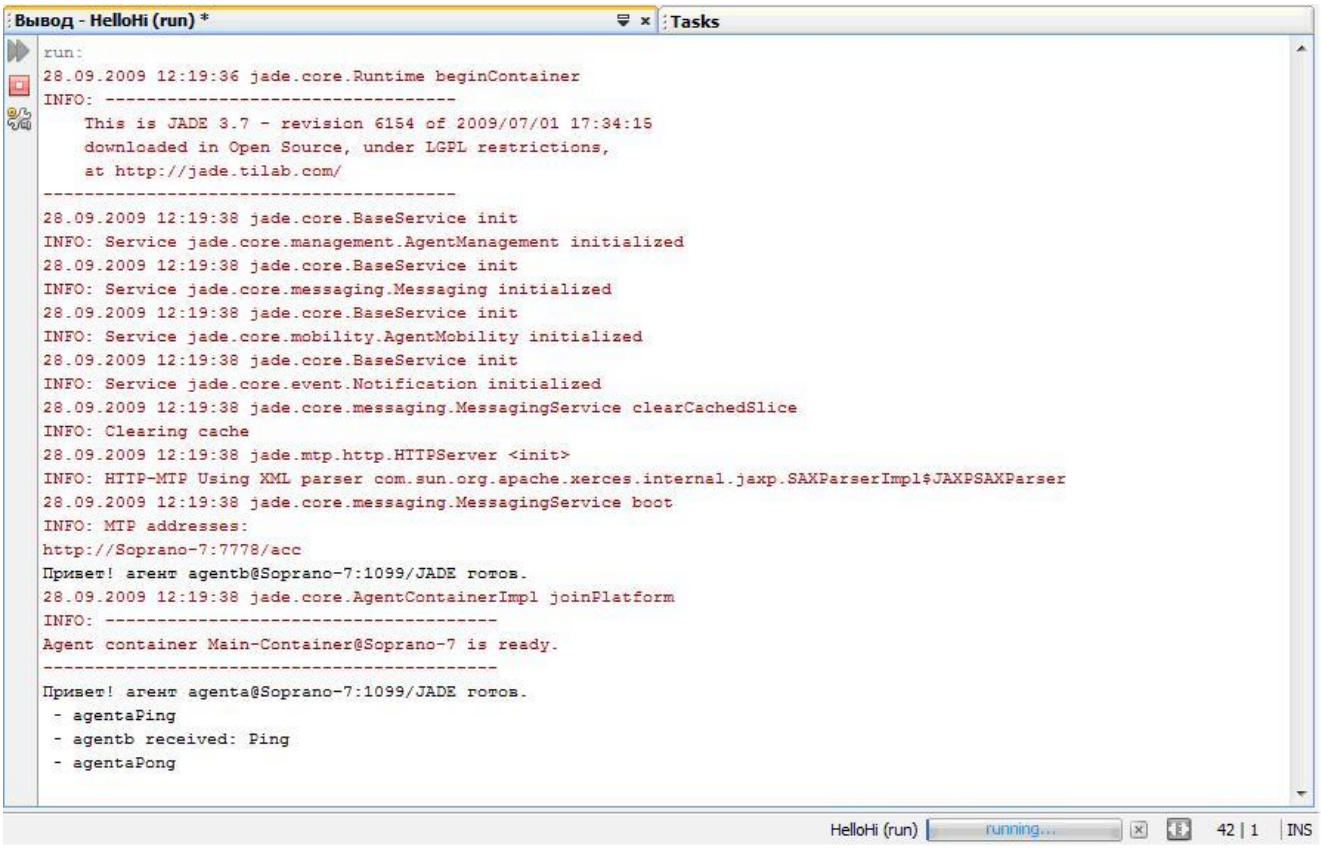


Рис.3. Вывод диалога между созданными агентами

Теперь можно запускать приложение!

Графический интерфейс платформы администрирования с использованием RMA агента выглядит следующим образом:

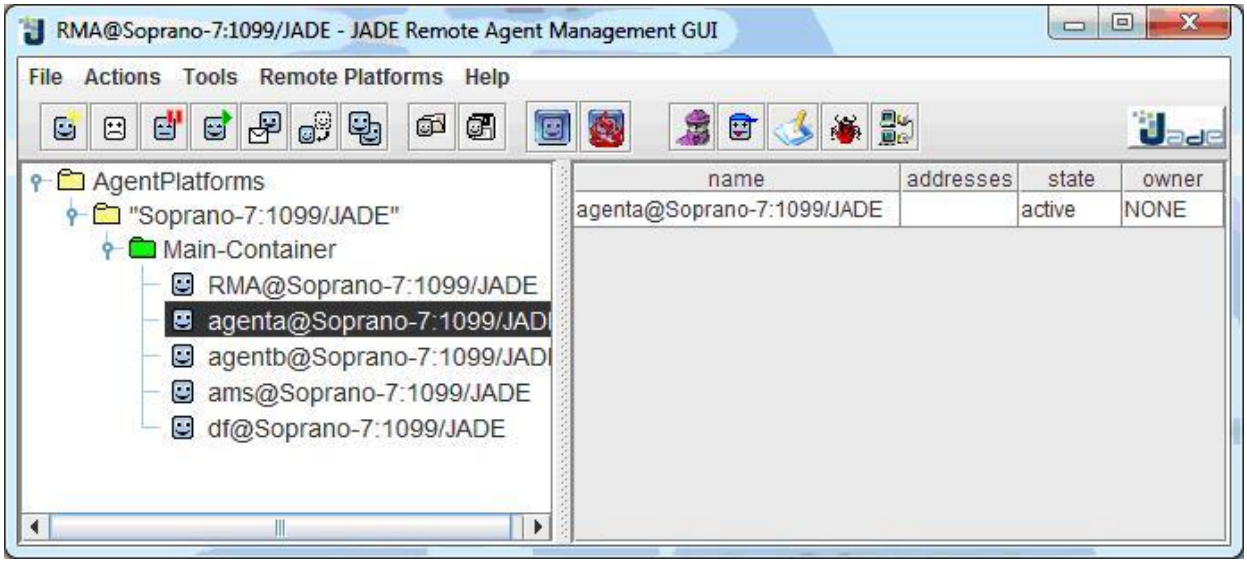


Рис. 4 Графический интерфейс JADE

Приложение

Пакет A (Агент A):

```
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package A;

import jade.core.Agent;
import jade.core.AID;
import jade.domain.AMSService;
import jade.domain.FIPAAgentManagement.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;

/**
*
* @author TRSteep
*/
public class AMain extends Agent {

public void setup() {
System.out.println("Привет! агент "+getAID().getName()+" готов.");
addBehaviour(new CyclicBehaviour(this) // Поведение агента исполняемое в цикле
{

public void action() {
ACLMessage msg = receive();
if (msg != null) {
```

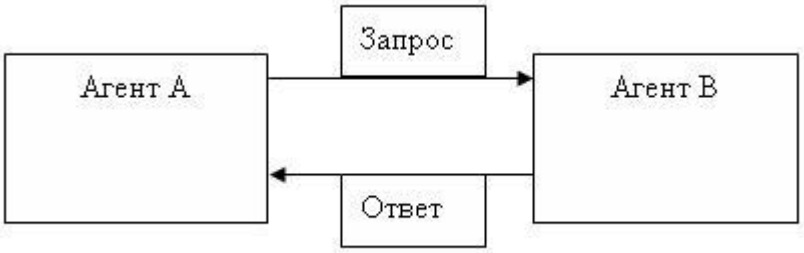
```
System.out.println( " – " + myAgent.getLocalName()
+ " received: "
+ msg.getContent() );
} //Вывод на экран локального имени агента и полученного сообщения
block();
//Блокируем поведение, пока в очереди сообщений агента не появится хотя бы одно
сообщение
}
});
AMSAgentDescription [] agents = null;
try {
SearchConstraints c = new SearchConstraints();
c.setMaxResults(new Long(-1));
agents = AMSService.search(this, new AMSAgentDescription(), c);
} catch (Exception e) {
System.out.println( "Problem searching AMS: " + e);

e.printStackTrace();
}

for(int i=0; i<4; i++) {
AID agentID = agents[i].getName();
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(agentID); // id агента которому отправляем сообщение
msg.setLanguage("English"); //Язык
msg.setContent("Ping"); //Содержимое сообщения

send(msg); //отправляем сообщение
}
}
}
```

Пакет B (Агент B):

```
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/

package B;

import jade.core.Agent;
import jade.core.behaviours.*;
import jade.lang.acl.*;
/**
*
* @author TRSteep
*/
public class BClass extends Agent {

protected void setup() {
System.out.println("Привет! агент "+getAID().getName()+" готов.");
addBehaviour(new CyclicBehaviour(this) {
```

```
public void action() {
ACLMessage msg = receive();
if (msg != null) {
System.out.println(" – " +
myAgent.getLocalName() +
" received: "
+ msg.getContent() );
//Вывод на экран локального имени агента и полученного сообщения
ACLMessage reply = msg.createReply();
reply.setPerformative( ACLMessage.INFORM );
reply.setContent( "Pong");
//Содержимое сообщения
send(reply); //отправляем сообщения
}
block();
}
});
}

}
```

Диаграмма взаимодействия межу агентами:



Практическая работа № 2

Разработка агентного приложения «Матчинг заказов и ресурсов»

Постановка задачи:

Создать агентное приложение «Матчинг заказов и ресурсов»

Задачи:

8. Создать простейшее агентное приложение HelloWorldAgent в качестве закрепления материала из лабораторной работы № 1
9. Познакомиться с теоретическими сведениями, рекомендованными преподавателем:

   http://translatedby.com/you/jade-programming-for-beginners/into-ru/trans/ (а именно раздел 5 Агент коммуникации - класс ACLMessage на стр. 12 данного руководства).
10. Самостоятельно найти информацию о различных языках, используемых в технологии интеллектуальных агентов, в соответствии с классификацией, записанной на практическом занятии.

11. Создать проект, содержащий агентное приложение «Матчинг заказов и ресурсов» используя библиотеку JADE (ВНИМАНИЕ: если в процессе создания NetBeans выдает сообщения об ошибках, игнорируйте их, продолжая выполнять задание).

12. Подготовить в печатном и электронном виде отчет по лабораторной работе №2 (обязательно наличие: титульного листа, цели, задач, информация о языках агентов, хода выполнения работы, выводов).
13. При составлении отчета привести комментарии ко всем строкам кода.

1. Задание на закрепление материала лабораторной работы №1. Создание агента HelloWorldAgent

Пользуясь знаниями, полученными в ходе лабораторной работы № 1, создайте и запустите агента HelloWorldAgent. Код агента приведен ниже.

/* **Код класса HelloWorldAgent.java** */

```
import jade.core.Agent;
public class HelloWorldAgent extends Agent
{
        protected void setup()
{
        System.out.println("Hello World! My name is "+getLocalName());
 }
}
```

В качестве параметров запуска можно использовать следующие данные:

jade.Boot Peter:HelloWorldAgent

Для создания агентного приложения используется стандартный инструмент, входящий в состав JADE, – *Remote Agent Management GUI (RAM)* (Рис. 1). Прежде всего, необходимо выбрать контейнер с агентами (на рисунке по умолчанию выбран главный контейнер *Main-Container*).

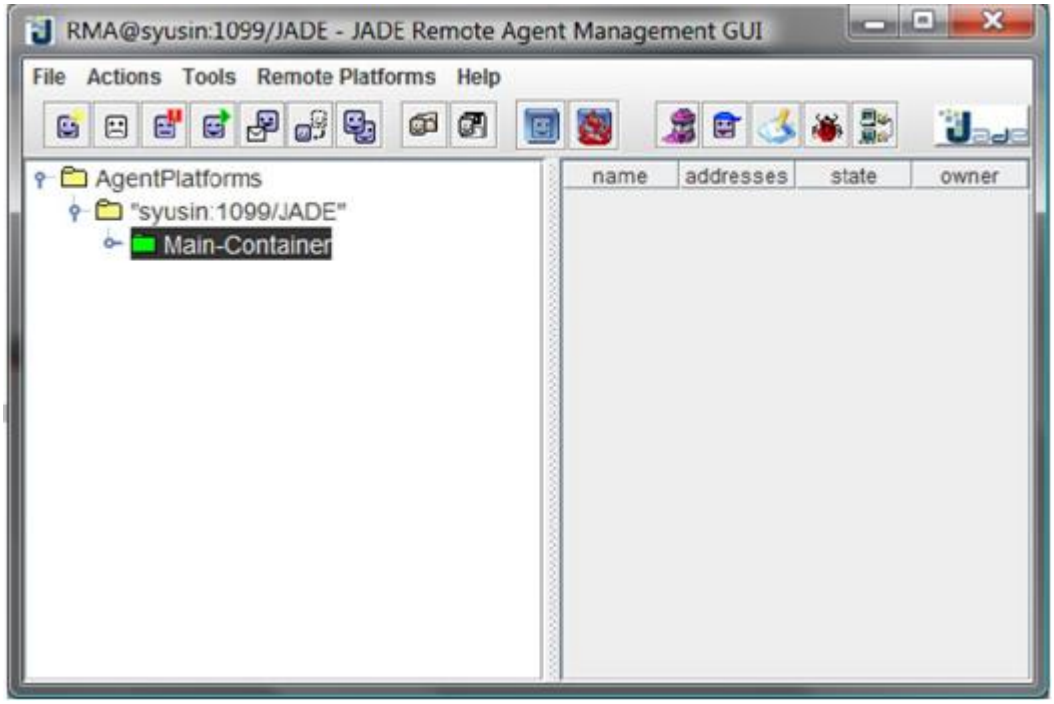


Рис. 1 Окно Remote Agent Management GUI (RAM)

Перед созданием агентов обязательно надо указать контейнер, иначе агенты не будут созданы (Рис.2).

Рис. 2 Ошибка создания агента: не выбран контейнер

После выполнения действий агент остается в системе в активном состоянии: он присутствует в списке агентов контейнера *Main-Container* (Рис.3)

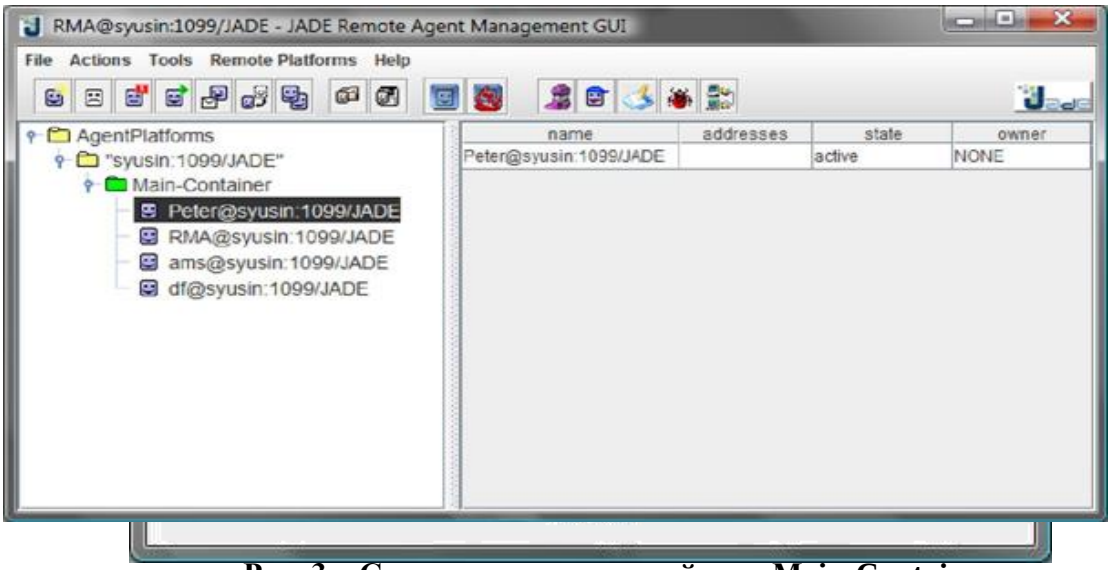


Рис. 3 – Список агентов контейнера Main-Container

Для того чтобы удалить агента из системы (а также из списка агентов контейнера), в код класса агента необходимо добавить вызов метода *doDelete()*:

```
public class HelloWorldAgent extends Agent
{
 protected void setup()
{
        System.out.println("Hello World! My name is "+
        getLocalName()); // Make this agent terminate
        doDelete()
}
}
```

3. Разработка приложения «Матчинг заказов и ресурсов»

- Данный пример демонстрирует следующую функциональность:
  создание и регистрация в системе агентов различных типов;
- рассылка сообщений между агентами;

- осуществление матчинга по заданным условиям;
- вывод результата матчинга.

В качестве модели данных используется отношение Заказ – Ресурс, наиболее часто встречающееся в сфере производства и логистики. Ресурс и заказ имеют активных агентов, которые осуществляют матчинг (поиск соответствия) для создания связей между заказом и подходящим ресурсом. Агент заказа рассылает сообщения всем агентам ресурсов,

зарегистрированным в системе. Сообщение содержит тип (информационное), ссылку на онтологию (в виде строки-названия), а также контент – название города, куда необходимо доставить заказ. Каждый ресурс может доставить заказ только в определенный город (пункт доставки), название которого задается пользователем при создании агента ресурса. Далее происходит матчинг – выполнить заказ могут только те ресурсы, пункт доставки которых совпадает с пунктом, указанным в сообщении, полученном от агента заказа.

Описание классов агентов заказа и ресурсов

Для реализации этой модели были определены два типа агентов – агент заказа (*OrderAgent)* и агент ресурса (*ResourceAgent*). Они являются расширением типа *Agent* в JADE. Ниже приведен код класса агента заказа

OrderAgent.java и код класса агента ресурса *ResourceAgent.java*.

***ВНИМАНИЕ:* IP адреса агентов ресурсов в AID[] resources должны совпадать с IP адресом ВАШЕГО компьютера.**

/* **Код класса OrderAgent.java** */

```
import
jade.core.Agent;
import jade.core.*;
import
jade.core.behaviours.*;
import jade.lang.acl.*;

public class OrderAgent extends Agent {

 protected void setup() {
        addBehaviour(new SimpleBehaviour(this)
    {
     private boolean finished = false;
     public String DestinationPoint = "Moscow";

     AID[] resources = {new
                        AID("Resource1@192.168.56.1:1099/JAD
                        E"), new
                        AID("Resource2@192.168.56.1:1099/JAD
                        E"), new
                        AID("Resource3@192.168.56.1:1099/JAD
                        E"), new
                        AID("Resource4@192.168.56.1:1099/JAD
                        E"), new
                        AID("Resource5@192.168.56.1:1099/JAD
```

```
                    E")};

    public void action()
    {
      System.out.println(getLocalName() +  " is active");
      ACLMessage msg = new
      ACLMessage(ACLMessage.INFORM);
      msg.setOntology("TestOntology");
      msg.setContent(DestinationPoint);
      for (int i=0; i <
        resources.length; i++)
        msg.addReceiver(resources[i
        ]);
      send(msg);
      finished =
      true;

    }

    public boolean done()
    {
      return finished;
    }
   });
  }
}
```

Примечание: у агента заказа определен пункт доставки – Moscow.

/* Код класса ResourceAgent.java */ import jade.core.Agent;
import jade.core.behaviours.*;
import jade.lang.acl.*;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

```
public class ResourceAgent extends Agent {

 protected void setup() {
       addBehaviour(new SimpleBehaviour(this)
   {
    private boolean finished =
    false; public String
    DestinationPoint;

    public void action()
    {
      System.out.println(getLocalName() + " is active");
      System.out.println(getLocalName() + " input destination point:");
      BufferedReader buff = new BufferedReader(new
```

```
        InputStreamReader(System.in)); try
        {
          DestinationPoint =  buff.readLine().toUpperCase();
        }
        catch (IOException E)  {}

        MessageTemplate m1 =
        MessageTemplate.MatchPerformative(ACLMessage.INFORM);
        MessageTemplate m2 =
        MessageTemplate.MatchOntology("TestOntology"); MessageTemplate
        m3 = MessageTemplate.and(m1, m2);
        ACLMessage msg = blockingReceive(m3,
        120000); if (msg != null)
        {
          System.out.println(getLocalName() + ": message from " +
          msg.getSender().getLocalName() + " was received");
           if
             (DestinationPoint.equals(msg.getContent().toUpper
             Case())) System.out.println(getLocalName() + ":
             order was accepted");
           else
             System.out.println(getLocalName() +  ": order was rejected");
        }
        else
           System.out.println(getLocalName() + ": empty message was
        received"); finished = true;
      }

      public boolean done()
      {
        return finished;
      }
     });

   }
}
```

Агент ресурса при создании и инициализации запрашивает у пользователя пункт доставки, а затем в течение определенного времени ждет поступления сообщения с определенными параметрами (сообщения фильтруются по типу и по ссылке на онтологию).

Примечание: метод *blockingReceive(m3, 120000)* использован для указания интервала времени (в миллисекундах), в течение которого агент должен ожидать входящее сообщение. После получения сообщения агент ресурса проверяет, совпадает ли его пункт доставки с пунктом, указанным в сообщении. Если совпадает, то агент принимает заказ, в противном случае, заказ отклоняется.

Создание агентного приложения «Матчинг заказов и ресурсов»

После запуска *Remote Agent Management GUI* открывается соответствующее окно (Рис.4).

Рис. 4 Окно Remote Agent Management GUI (RAM)

В окне RAM следует создать одного агента заказа и необходимое количество агентов ресурсов.

ВНИМАНИЕ: Вводить пункт доставки у агента *ResourceN* (где N – номер агента) необходимо в NetBeans в окне ввода (внизу под программным кодом).

Чтобы создать экземпляр агента типа *ResourceAgent* с именем *Resource1*, необходимо выполнить команды *Actions -> Start New Agent*. Далее следует указать имя экземпляра (*Resource1*), а также тип создаваемого агента, который должен совпадать с именем jar-файла (Рис. 5). После нажатия кнопки <OK> агент создается и регистрируется в системе, при этом его статус

устанавливается в *Active*. Характеристики агента, выбранного в списке агентов данного контейнера, отображаются в правом окне RAM (Рис. 6).

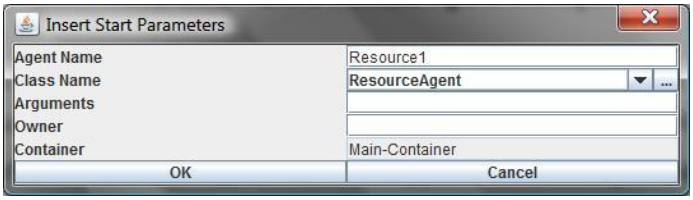


Рис. 5 Инициализация агента

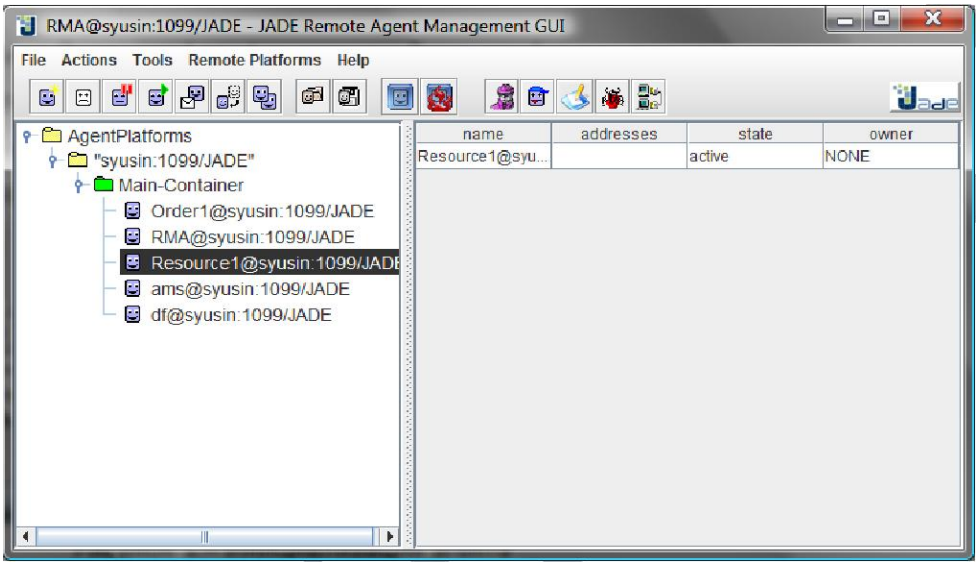


Рис. 6 Просмотр состояния агента в RAM

При создании агента ресурса в открывшемся окне командной строки пользователю будет предложено ввести название пункта доставки для данного ресурса (Рис.7). Можно вводить любое значение, но следует помнить, что заказ должен быть доставлен в пункт

Moscow. Поэтому для успешного осуществления матчинга необходимо, чтобы хотя бы для одного агента ресурса был определен пункт доставки Moscow.

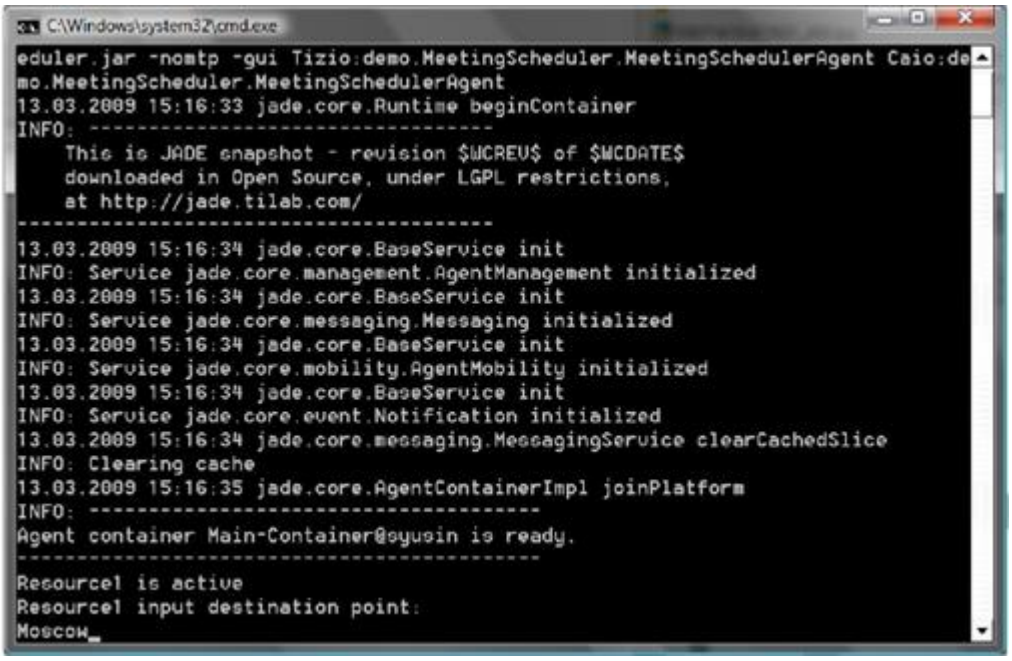


Рис. 7 Ввод наименования пункта доставки для агента ресурса

Создать еще четырех агентов ресурсов с именами *Resource2*, *Resource3*, *Resource4*, *Resource5*. Для каждого из них задать названия пунктов доставки (Рис. 8). Таким образом, создается ресурсная база, на основании которой можно выбирать наиболее подходящий вариант для выполнения заказа. Для агентов ресурсов программно установлено время ожидания сообщения – 2 минуты. По истечении 2-х минут агент просматривает очередь сообщений и,

если она пуста, выводится сообщение *ResourceX: empty message was received*.

Поэтому необходимо инициализировать всех агентов ресурсов и агента заказа в течение 2-х минут.

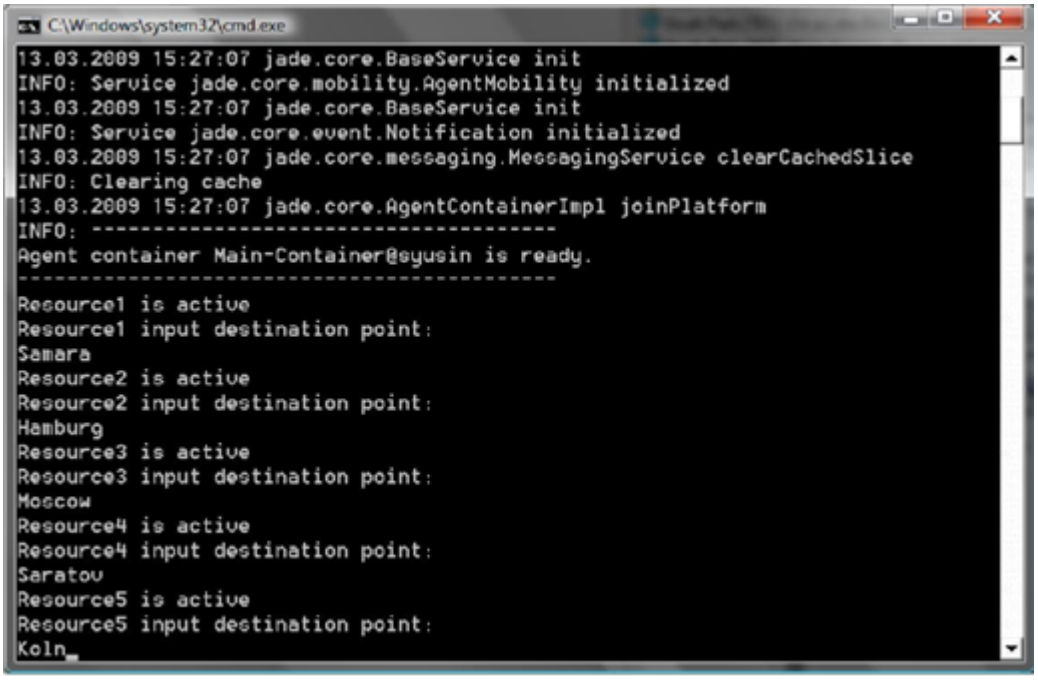


Рис. 8 Инициализация пяти агентов ресурсов

Создание агента заказа типа *OrderAgent* показано на Рис. 9.

Рис. 9 Инициализация агента заказа

Т.к. в классе агента *OrderAgent.java* указан пункт доставки Moscow, агент заказа будет выбирать ресурс с таким же пунктом доставки.

После создания агента заказа он сразу же рассылает сообщения агентам ресурсов. Каждый агент реагирует на это сообщение, при этом возможны два варианта ответа: подтверждение возможности выполнения заказа или отклонение его. Если несколько агентов смогут принять заказ, то все они ответят положительно. Ответы агентов ресурсов приведены на Рис. 10.

Сообщение типа *ResourceX: message from Order1 was received* означает, что агент ресурса получил сообщение от агента заказа, а сообщения типа

ResourceX: order was accepted или *ResourceX: order was rejected* означают соответственно прием или отклонение заказа. Из полученного списка сообщений следует, что только *Resource3* подтвердил возможность выполнения заказа, т.к. только у этого ресурса пункт доставки совпадает с пунктом доставки, указанным в заказе (Рис. 10). Остальные агенты ресурсов отклонили заказ. Следует отметить, что ответы ресурсов поступали не в том порядке, в котором они были инициализированы в системе. Следовательно, пересылка сообщений и ответов между агентами выполнялась в случайном порядке.

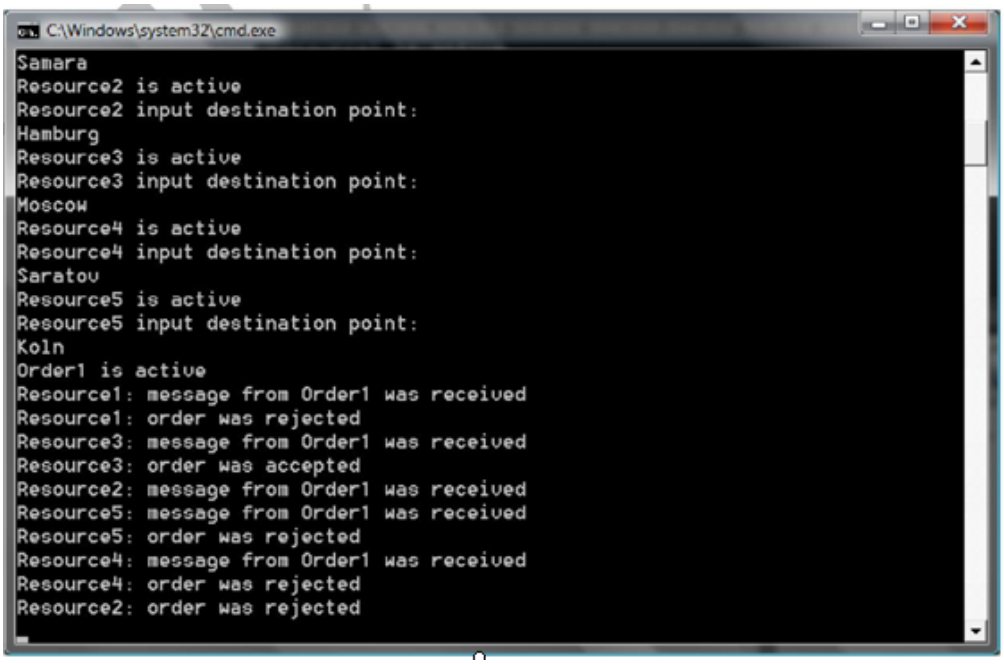


Рис. 10 Ответы агентов ресурсов на сообщение агента заказа

Практическая работа № 3

Разработка агентного приложения «Торговля книгами»

Постановка задачи:

Создать агентное приложение «Торговля книгами»

Задачи:

14. Познакомиться с теоретическими сведениями, приведенными в лабораторной работе и необходимыми для целостного понимания процесса создания агентного приложения со сложным поведением.

15. Создать проект, содержащий агентное приложение «Торговля книгами» используя библиотеку JADE.

16. Подготовить в печатном и электронном виде отчет по лабораторной работе №3 (обязательно наличие: титульного листа, цели, задач, хода выполнения работы, выводов).
17. При составлении отчета привести комментарии ко всем строкам кода.

18. **РАЗРАБОТКА АГЕНТНОГО ПРИЛОЖЕНИЯ «ТОРГОВЛЯ КНИГАМИ» НА ОСНОВЕ ПЛАТФОРМЫ JADE**

Основные этапы разработки мультиагентного приложения с использованием платформы JADE рассмотрим на примере приложения «Торговля книгами». В приложении действуют агенты-книготорговцы и агенты-покупатели, приобретающие книги по поручению своих владельцев.

Каждый агент-покупатель получает название книги, которую необходимо купить («требуемая книга»), в качестве аргумента командной строки и периодически запрашивает предложения у всех известных ему агентов- продавцов. Как только предложение получено, агент-покупатель принимает его и выдает заказ на покупку. Если несколько агентов-продавцов предоставляют предложение агенту-покупателю, он выбирает из них лучшее (с самой низкой ценой). После покупки агент-покупатель завершает свою работу.

Каждый агент-продавец имеет минимальный интерфейс, с помощью которого пользователь может добавить названия новых книг и цены на них в свой локальный каталог книг для продажи. Агент-продавец непрерывно проверяет поступление запросов от агентов-покупателей. При получении запроса на книгу каждый агент-продавец проверяет свой локальный каталог и, если запрашиваемая книга находится в его каталоге, отвечает на запрос, сообщая агенту-покупателю цену. После выполнения заказа купленная книга автоматически удаляется из каталога.

Коды классов, реализующих данное приложение, находятся в папке *\jade\src\examples\bookTraiding*.

19. **Класс «Агент»**

20. ***Создание агента***

Агент JADE создается с помощью определения класса, наследуемого от класса *jade.core.Agent*, и реализации метода *setup()*:

```
import jade.core.Agent;

public class BookBuyerAgent extends Agent { protected void
setup() {

// Printout a welcome message
```

```
System.out.println("Hello! Buyer-agent "+getAID().getName()+" is ready.");
}
}
```

Метод *setup()* выполняет инициализацию агента. Далее агент функционирует в рамках своего поведения (Класс «Поведение агента»).

21. ***Идентификация агента***

Каждый агент имеет определенный «идентификатор агента», представляющий собой экземпляр класса *jade.core.AID*. Метод *getAID()* класса *Agent* позволяет получить идентификатор агента. Объект *AID* содержит глобальное уникальное имя, а также адрес. Идентификатор агента JADE имеет структуру: *<имя_агента>@<имя_платформы>*, поэтому агент с именем *Peter*, находящийся на платформе *P1*, будет иметь глобальное уникальное имя *Peter@P1*. Адрес, содержащийся в *AID*, – это адрес платформы, где находится агент. Этот адрес используется только в тех случаях, когда агент должен общаться с другими агентами, которые находятся на другой платформе. *AID* агента может быть получен по его идентификатору следующим образом:

```
String nickname = "Peter";
AID id = new AID(nickname, AID.ISLOCALNAME);
```

Константа *ISLOCALNAME* указывает, что первый параметр представляет собой локальный идентификатор на платформе, а не глобальный уникальный идентификатор этого агента.

22. ***Запуск агента***

Для того чтобы запустить откомпилированого агента на выполнение, необходимо запустить среду JADE и указать имя агента (Start New Agent).

Например, может быть получен следующий результат:

```
C:\jade>java –classpath <JADE-classes> jade.Boot buyer:BookBuyerAgent 5-mag-
2008 11.06.45 jade.core.Runtime beginContainer

INFO: ----------------------------------

This is JADE snapshot - revision 5995 of 2007/09/03 09:45:22 downloaded in
Open Source, under LGPL restrictions,

at http://jade.tilab.com/

----------------------------------------

5-mag-2008 11.06.51 jade.core.BaseService init

INFO: Service jade.core.management.AgentManagement initialized 5-mag-
2008 11.06.51 jade.core.BaseService init

INFO: Service jade.core.messaging.Messaging initialized 5-mag-
2008 11.06.52 jade.core.BaseService init

INFO: Service jade.core.mobility.AgentMobility initialized 5-mag-2008
11.06.52 jade.core.BaseService init

INFO: Service jade.core.event.Notification initialized
```

5-mag-2008 11.06.52 jade.core.messaging.MessagingService clearCachedSlice INFO:
Clearing cache

5-mag-2008 11.06.53 jade.mtp.http.HTTPServer <init> INFO:
HTTP-MTP Using XML parser
com.sun.org.apache.xerces.internal.parsers.SAXParser

5-mag-2008 11.06.54 jade.core.messaging.MessagingService boot INFO: MTP
addresses: http://NBNT2004130496.telecomitalia.local:7778/acc

5-mag-2008 11.06.54 jade.core.AgentContainerImpl joinPlatform INFO: ---------
---

Agent container Main-Container@NBNT2004130496 is ready.

Hello! Buyer-agent buyer@NBNT2004130496:1099/JADE is ready.

При запуске среды JADE выполняется инициализация ядра сервисов JADE. Процесс завершается сообщением о том, что контейнер с именем *«Main- Container»* готов к работе. Когда среда JADE запускает пользовательского агента, он выводит свое приветствие с именем *«Buyer»*, указанным при создании агента. Имя платформы *«NBNT2004130496: 1099/JADE»* назначается автоматически в соответствии с адресом хоста и порта, на котором работает среда JADE.

23. ***Завершение работы агента***

Даже если пользовательский агент ничего не делает после вывода приветствия, он продолжает работать. Чтобы завершить его работу, необходимо вызвать метод *doDelete ()*. Непосредственно перед завершением работы агента вызывается метод *takeDown ()*, который выполняет операции по очистке агента.

24. ***Передача аргументов агенту***

Агенты могут получать при запуске аргументы, заданные в командной строке. Эти аргументы могут быть получены как массив *Object* с помощью метода *getArguments ()* класса *Agent*. Агент *BookBuyerAgent* должен получить в качестве аргумента командной строки название книги, которую необходимо купить. Для достижения этой цели изменим агента (заметим, что список известных агентов-продавцов для отправки запросов фиксирован.

```
import jade.core.Agent; import
jade.core.AID;

public class BookBuyerAgent extends Agent {

// The title of the book to buy private String
targetBookTitle;

// The list of known seller agents

private AID[] sellerAgents = {new AID("seller1", AID.ISLOCALNAME), new
AID("seller2", AID.ISLOCALNAME)};

// Put agent initializations here protected
void setup() {
```

```
// Printout a welcome message
System.out.println("Hello! Buyer-agent "+getAID().getName()+" is ready.");
// Get the title of the book to buy as a start-up argument
Object[] args = getArguments();
if (args != null && args.length > 0) { targetBookTitle = (String)
args[0]; System.out.println("Trying to buy "+targetBookTitle);
}
else {
// Make the agent terminate immediately
System.out.println("No book title specified"); doDelete();
}
}
// Put agent clean-up operations here protected
void takeDown() {
// Printout a dismissal message
System.out.println("Buyer-agent "+getAID().getName()+" terminating.");
}
}
```

Аргументы командной строки задаются внутри скобок и разделяются пробелами.

```
C:\jade>java jade.Boot buyer:BookBuyerAgent(The-Lord-of-the-rings)
...
...
5-mag-2008 11.11.00 jade.core.AgentContainerImpl joinPlatform INFO: ---------
----------------------------
Agent container Main-Container@NBNT2004130496 is ready.
--------------------------------------------
Hello! Buyer-agent buyer@NBNT2004130496:1099/JADE is ready.
    Trying to buy The-Lord-of-the-Rings
```

25. Класс «Поведение агента»

Активность агента выполняется в рамках *«поведения»* (*режима работы*). Каждый агент исполняется в отдельном потоке Java. Логика агента строится из комбинации режимов.

Поведение представляет собой последовательность действий, которые должен выполнить агент (аналогично методу Java-класса), и реализуется как объект класса, наследуемого от *jade.core.behaviours.Behaviour*. Поведение агента определяется с помощью метода *addBehaviour ()* класса *Agent*. Поведение может быть добавлено в любой момент: при запуске агента (в методе *Setup ()*) или внутри другого режима работы.

Каждый класс, наследуемый от класса *Behaviour*, должен переопределить метод *action()*, который фактически определяет операции, задающие поведение агента, а также метод *done()*, который возвращает булево значение, указывающее на завершение того или иного поведения, которое должно быть удалено из пула режимов работы агента.

26. *Планирование и исполнение режимов работы агента*

Агент может выполнять одновременно несколько моделей поведения. Однако важно заметить, что планирование режимов агента является не упреждающим (как в потоках Java), а кооперативным. Это означает, что, когда поведение планируется к исполнению, метод *action()* вызван и работает до тех пор, пока режим завершится. Поэтому программист должен определять, когда агент переключается от выполнения одного режима к выполнению следующего. Такой подход имеет ряд преимуществ:

- поддерживает работу с одним потоком Java для агента (что является весьма важным в условиях ограниченных ресурсов, какими обладают, например, сотовые телефоны);
- обеспечивает более быстрое переключение между режимами работы, чем переключение между потоками Java;
- устраняет проблемы синхронизации параллельных режимов работы для доступа к одним и тем же ресурсам (что ускоряет производительность), т.к. все режимы выполняются одним и тем же потоком Java;
- позволяет сохранять состояние агента в постоянном хранилище для последующего возобновления (сохранямость агента) или передачи его другому контейнеру для удаленного исполнения (мобильность агента).

При создании агента вызывается метод *setup()*, в котором происходит инициализация агента. Далее в бесконечном цикле выбирается следующий режим из пула активных и исполняется. Затем, в зависимости от типа режима, он либо удаляется из пула активных, либо остается в нем и выполняется снова, когда до него дойдет очередь. При отсутствии активных режимов (например, если агент ждет сообщение) поток агента «засыпает» (что позволяет экономить ресурсы процессора). При удалении агента вызывается метод *takeDown()*.

В соответствии с описанным механизмом планирования, режим, представленный ниже, не позволяет запустить другой режим, т.к. его метод *action()* никогда не завершается.

```
public class OverbearingBehaviour extends Behaviour { public void
action() {

while (true) {

// do something

}

}

public boolean done() { return
true;

}

}
```

27. *Типы режимов агента*

Различают три типа поведения агентов.

1) Одноразовый (*one-shot*) режим, у которого метод *action()* выполняется только один раз, после чего режим становится неактивным. *jade.core.behaviours.OneShotBehaviour* уже реализует метод *done()*, возвращающий значение *true*, и может быть легко модифицирован для реализации одноразового поведения.

```
public class MyOneShotBehaviour extends OneShotBehaviour { public void
action() {

// perform operation X

}

}
```

Операция X будет выполнена только один раз.

2) Циклический (*cyclic*) режим, который никогда не завершается и всегда продолжает оставаться активным. Метод *action()* выполняет одинаковые операции каждый раз, когда он вызывается. *jade.core.behaviours.CyclicBehaviour* уже реализует метод *done()*, возвращающий значение *false*, и может быть легко модифицирован для реализации циклического поведения.

```
public class MyCyclicBehaviour extends CyclicBehaviour { public void
action() {

// perform operation Y

}

}
```

Операция Y выполняется многократно, до завершения работы агента, выполняющего циклическое поведение.

3) Общий (*generic*) режим, который выполняет различные операции в зависимости от значения некоторой переменной. Режим остается активным до тех пор, пока выполняется условие, заданное 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;

}

}
```

Операции X, Y и Z выполняются одна за другой, а затем поведение завершается.

JADE предусматривает возможность комбинации простых режимов работы агента для создания более сложных режимов.

28. ***Планирование операций в заданных временных точках***

JADE предлагает два готовых класса (в пакете *jade.core.behaviours*), с помощью которых можно легко реализовывать режимы работы, выполняющие определенные операции в заданные моменты времени.

1) Режим *WakerBehaviour*, у которого методы *action()* и *done()* реализованы таким образом, что абстрактный метод *handleElapsedTimeout ()* выполняется по истечении заданного интервала времени (указанного в конструкторе). После исполнения метода *handleElapsedTimeout ()* поведение завершается.

```
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

}

} );

}

}
```

Операция X выполняется через 10 секунд после вывода на экран сообщения «Adding waker behaviour».

2) Режим *TickerBehaviour*, у которого методы *action()* и *done()* реализованы таким образом, что абстрактный метод *onTick()* выполняется многократно, через определенный интервал времени (указанный в конструкторе). Режим *TickerBehaviour* никогда не завершается.

```
public class MyAgent extends Agent { protected
void setup() {

addBehaviour(new TickerBehaviour(this, 10000) { protected
void onTick() {

// perform operation Y

}

} );

}

}
```

Операция Y выполняется периодически каждые 10 секунд.

29. *Режимы работы агентов в приложении «Торговля книгами»*

В соответствии с описанием основных типов поведения агентов, проанализируем поведение агентов *Book-buyer* и *Book-seller* в приложении «Торговля книгами».

30. Поведение агента Book-buyer

Агент *Book-buyer* должен периодически запрашивать у агента продавца книгу, которую ему было поручено купить. Этого можно достичь, если модифицировать режим *TickerBehaviour* так, что каждый «тик» добавляет другой режим, который фактически выполняет запрос к агенту-продавцу. Метод *setup()* класса *BookBuyerAgent* необходимо изменить следующим образом.

```
protected void setup() {

// Printout a welcome message

System.out.println(“Hello! Buyer-agent “+getAID().getName()+” is ready.”);

// Get the title of the book to buy as a start-up argument Object[] args =
getArguments();

if (args != null && args.length > 0) { targetBookTitle = (String)
args[0]; System.out.println(“Trying to buy “+targetBookTitle);

// Add a TickerBehaviour that schedules a request to seller agents every minute addBehaviour(new
TickerBehaviour(this, 60000) {

protected void onTick() { myAgent.addBehaviour(new
RequestPerformer());

}

} );

}

else {
```

```
// Make the agent terminate

System.out.println("No target book title specified"); doDelete();

}

}
```

Обратите внимание на использование защищѐнной переменной *myAgent*: каждый режим содержит указатель на агента, который его исполняет.

Поведение агента Book-seller

Каждый агент *Book-seller* ожидает прихода запросов от агентов- покупателей и обслуживает их. Запросы могут быть различных типов: запрос о представлении предложения на книгу и запрос на выполнение заказа. Для реализации запросов различных типов агенту *Book-seller* необходимо задать два циклических поведения: одно для обслуживания запросов на предложение, а другое для обслуживания заказов. Принципы обнаружения и обслуживания входящих запросов от агентов-покупателей, описано в пункте Класс «взаимодействие между агентами», где будет обсуждаться взаимодействие между агентами. Кроме того, необходимо задать агенту *Book-seller* одноразовое поведение, реализующее обновление каталога книг, доступных для продажи в тех случаях, когда пользователь добавляет новую книгу из GUI. Класс *BookSellerAgent* может быть реализован следующим образом (классы *OfferRequestsServer* и *PurchaseOrdersServer* будут описаны в пункте Класс «взаимодействие между агентами»).

```
import jade.core.Agent;

import jade.core.behaviours.*; import
java.util.*;

public class BookSellerAgent extends Agent {

// The catalogue of books for sale (maps the title of a book to its price) private Hashtable catalogue;

// The GUI by means of which the user can add books in the catalogue private BookSellerGui
myGui;

// Put agent initializations here protected
void setup() {

// Create the catalogue catalogue =
new Hashtable();

// Create and show the GUI

myGui = new BookSellerGui(this);
myGui.show();

// Add the behaviour serving requests for offer from buyer agents addBehaviour(new
OfferRequestsServer());

// Add the behaviour serving purchase orders from buyer agents
addBehaviour(new PurchaseOrdersServer());
```

```
}
// Put agent clean-up operations here protected
void takeDown() {
// Close the GUI myGui.dispose();
// Printout a dismissal message
System.out.println("Seller-agent "+getAID().getName()+" terminating.");
}
/**
This is invoked by the GUI when the user adds a new book for sale
*/
public void updateCatalogue(final String title, final int price) { addBehaviour(new
OneShotBehaviour() {
public void action() { catalogue.put(title, new
Integer(price));
}
} );
}
}
```

31. ***Класс «взаимодействие между агентами»***

Одной из наиболее важных функциональных возможностей, которую предоставляют JADE агенты, является возможность коммуникации между ними. Обмен сообщениями в JADE является асинхронным и осуществляется с помощью Message Transport System. Формат сообщения соответствует стандарту FIPA. При этом внутри платформы для ускорения работы сообщения пересылаются в виде Java-объектов, а при обмене между платформами (или с другими MAC) – в виде XML-строки. При отправке сообщения оно попадает в
«почтовый ящик» (очередь сообщений) агента-адресата, о чем агент уведомляется. Сообщение может быть получено из «почтового ящика» агента и обработано любым из его режимов. Для выбора нужного типа сообщения из очереди программист может использовать фильтры.

Для ускорения разработки мультиагентных приложений в JADE существует библиотека шаблонов переговоров (*FIPA protocols*), таких как аукцион, contract net, подписка и т.д.

32. ***Язык ACL***

Сообщения, которыми обмениваются между собой агенты JADE, имеют формат языка ACL, определенного FIPA как международный стандарт взаимодействия агентов. Этот формат включает следующие поля:

- отправитель сообщения;

- список получателей;
- коммуникативные действия («пожелания»), указывающие цель отправки сообщения. Типы сообщений:
  - REQUEST (запрос), если отправитель желает, чтобы получатель произвèл некоторое действие,
  - INFORM (информирование), если отправитель желает, чтобы получатель был извещèн о некотором факте,
  - QUERY_IF, если отправитель желает знать, достигнуто или нет заданное условие,
    CFP (извещение о предложении), PROPOSE (предложение), ACCEPT_PROPOSAL (принятие предложения), REJECT_PROPOSAL (отклонение предложения), если отправитель и получатель ведут переговоры,
    другие типы коммуникативных действий;
- содержание, т.е. фактическая информация, содержащаяся в сообщении (например, действия, которые будут выполняться по запросу REQUEST; факт, о котором отправитель желает информировать в сообщении INFORM и т.п.);
- язык, на котором представлено содержание (отправитель и получатель должны иметь возможность кодировать/декодировать выражения согласно синтаксису языка);
- онтология, т.е. словарь символов, используемых в содержании, и их значения (отправитель и получатель должны понимать смысл символов одинаково);
- поля, используемые для управления несколькими параллельными переговорами, определяющие интервал времени для получения ответа, такие как *conversation-id, reply-with, in-reply-to, reply-by.*

Сообщение в JADE реализуется как объект класса *jade.lang.acl.ACLMessage,* предоставляющего методы *get()* и *set()* для установки значений всех полей при получении и посылке сообщений.

33. *Посылка сообщений*

Чтобы отправить сообщение другому агенту, необходимо заполнить поля объекта *ACLMessage* и вызвать метод *send()* класса *Agent*. Приведенный ниже код демонстрирует, как создать и отправить сообщение, информирующее агента по имени *Peter* о том, что сегодня идет дождь.

```
ACLMessage msg = new
ACLMessage(ACLMessage.INFORM); msg.addReceiver(new
AID(“Peter”, AID.ISLOCALNAME));
msg.setLanguage(“English”); msg.setOntology(“Weather-
forecast-ontology”); msg.setContent(“Today it’s raining”);

send(msg);
```

34. *Посылка сообщений в приложении «Торговля книгами»*

В приложении «Торговля книгами» используется сообщение CFP для извещения о том, что агент продавца *Buyer-agent* отправил агенту покупателя *Seller-agent* предложение на приобретение книг. Сообщение PROPOSE содержит предложение продавца, а сообщение ACCEPT_PROPOSAL содержит заказ покупателя. Сообщение REFUSE агента продавца сообщает о том, что книги, запрошенной агентом покупателя, нет в каталоге. В обоих типах сообщений, отправляемых агенту покупателя, содержится название книги. Содержанием сообщения PROPOSE должна быть цена на книгу. Ниже приведен пример, демонстрирующий, каким образом создается и оправляется сообщение CFP.

```
// Message carrying a request for offer ACLMessage cfp =
new ACLMessage(ACLMessage.CFP); for (int i = 0; i <
sellerAgents.lenght; ++i) {
cfp.addReceiver(sellerAgents[i]);

}

cfp.setContent(targetBookTitle);

myAgent.send(cfp);
```

35. *Получение сообщений*

Среда исполнения JADE автоматически помещает сообщения, как только они доставлены, в очередь сообщений получателя. Агент может получать сообщения из своей очереди сообщений с помощью метода *receive()*. Этот метод возвращает первое сообщение, находящееся в очереди (и удаляет его оттуда), либо ничего не возвращает, если очередь сообщений пуста, после чего сразу же завершается.

```
ACLMessage msg = receive(); if
(msg != null) {

// Process the message

}
```

36. *Блокирование режима работы агента для ожидания сообщения*

Очень часто в поведении агента необходимо реализовать получение сообщений от других агентов, например, в режимах *OfferRequestsServer* и *PurchaseOrdersServer*, которые были описаны в пункте Поведение агента Book- seller, требуется обслуживать сообщения от агента покупателя, пересылающего заявки на предложение и заказ. Такое поведение должно работать постоянно (циклическое поведение), причем при каждом выполнении метода *action()* требуется проверка, что сообщение было получено и обработано. Ниже представлен режим *OfferRequestsServer* (режим *PurchaseOrdersServer* реализуется аналогично).

```
/**

Inner class OfferRequestsServer.

This is the behaviour used by Book-seller agents to serve incoming requests

for offer from buyer agents.
```

```
If the requested book is in the local catalogue the seller agent replies with a PROPOSE
message specifying the price. Otherwise a REFUSE message is sent back.

*/

private class OfferRequestsServer extends CyclicBehaviour { public void
action() {

ACLMessage msg = myAgent.receive(); if
(msg != null) {

// Message received. Process it String title =
msg.getContent(); ACLMessage reply =
msg.createReply();

Integer price = (Integer) catalogue.get(title); if (price !=
null) {

// The requested book is available for sale. Reply with the price
reply.setPerformative(ACLMessage.PROPOSE);
reply.setContent(String.valueOf(price.intValue()));

}

else {

// The requested book is NOT available for sale.
reply.setPerformative(ACLMessage.REFUSE);
reply.setContent("not-available");

}

myAgent.send(reply);

}

}

} // End of inner class OfferRequestsServer
```

В данном коде поведение *OfferRequestsServer* реализовано как внутренний класс класса *BookSellerAgent*. Это упрощает реализацию, т.к. обеспечивает прямой доступ к каталогу книг для продажи, но не является обязательным.

Метод *createReply ()* класса *ACLMessage* автоматически создает новую настройку *ACLMessageproperly* получателя и все поля, используемые для управления сеансом (*conversation-id, reply-with, in-reply-to*).

При добавлении поведения, описанного выше, поток агента входит в непрерывный цикл, что слишком нагружает процессор. Чтобы избежать этого, *метод action()* поведения *OfferRequestsServer* следует выполнять лишь тогда, когда появится новое сообщение. Для этого можно использовать метод *block()* класса поведений. Этот метод помечает поведение как «заблокированное», так что агент не планирует следующих выполнений. Когда в очередь сообщений агента добавится новое сообщение, все заблокированные режимы снова становятся доступными для исполнения и получают возможность обрабатывать полученные сообщения. Метод *action()* должен быть изменен следующим

образом.

```
public void action() {

ACLMessage msg = myAgent.receive(); if
(msg != null) {

// Message received. Process it

...

}

else {
block();

}

}
```

Приведенный выше код является типичным (и рекомендуемым) шаблоном для реализации приема сообщений внутри поведения агента.
37.

38. ***Выбор сообщений с указанными характеристиками из очереди сообщений***

Т.к. оба режима *OfferRequestsServer* и *PurchaseOrdersServer* являются циклическими, их метод *action()* которых начинается с вызова метода *myAgent.receive()*, что порождает следующую проблему: как определить, что поведение *OfferRequestsServer* выбирает из очереди сообщений только сообщения, в которых содержится запрос о наличии книги, а поведение *PurchaseOrdersServer* – только сообщения, содержащие заказы на поставку? Для разрешения этой проблемы нужно изменить код, указав соответствующие

«шаблоны» при вызыве метода *receive()*. Когда шаблон указан, метод *receive()* возвращает первое сообщение (если оно есть), соответствующее данному шаблону, игнорируя все неподходящие сообщения. Подобные шаблоны реализованы как экземпляры класса *jade.lang.acl.MessageTemplate*, который предоставляет набор методов, позволяющих просто и гибко создавать шаблоны.

Как было указано в пункте *Посылка сообщений в приложении «Торговля книгами»*, для запроса о наличии книги используется сообщение CFP, и сообщение ACCEPT_PROPOSAL – для передачи предложения о заказе. Поэтому необходимо изменить метод *action()* класса *OfferRequestServer* так, чтобы вызов *myAgent.receive()* игнорировал все сообщения, кроме CFP.

```
public void action() {

MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP); ACLMessage
msg = myAgent.receive(mt);

if (msg != null) {

// CFP Message received. Process it

...

}
```

```
else {
block();

}

}
```

39. *Сложные переговоры*

Режим *RequestPerformer*, описанный в пункте Поведение агента Book- buyer, представляет собой пример поведения, проводящего «сложные» переговоры. В данном случае, «переговоры» – это последовательность сообщений, которыми обмениваются два или более агентов с чѐтко определѐнными причинными и временными отношениями. Поведение *RequestProposal* должно послать CFP-сообщение нескольким агентам, известным агентам-продавцам, получить обратно все ответы и, в случае, если получен хотя бы один ответ типа PROPOSE, позже послать сообщение ACCEPT_PROPOSAL агенту-продавцу, который сделал предложение, и получить обратно ответ. Всякий раз, когда происходит обмен сообщениями, необходимо определить управляющие поля в сообщениях, участвующих в обмене. Это позволит легко и однозначно создавать шаблоны, соответствующие возможным ответам.

```
/**

Inner class RequestPerformer.

This is the behaviour used by Book-buyer agents to request seller agents the target
book.

*/

private class RequestPerformer extends Behaviour {

private AID bestSeller; // The agent who provides the best offer private int
bestPrice; // The best offered price

private int repliesCnt = 0; // The counter of replies from seller agents private
MessageTemplate mt; // The template to receive replies

private int step = 0; public
void action() { switch (step) {

case 0:

// Send the cfp to all sellers

ACLMessage cfp = new ACLMessage(ACLMessage.CFP); for
(int i = 0; i < sellerAgents.length; ++i) {
cfp.addReceiver(sellerAgents[i]);

}

cfp.setContent(targetBookTitle);
cfp.setConversationId(“book-trade”);

cfp.setReplyWith(“cfp”+System.currentTimeMillis()); // Unique value myAgent.send(cfp);

// Prepare the template to get proposals
```

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

step = 1;
break; case
1:

// Receive all proposals/refusals from seller agents ACLMessage
reply = myAgent.receive(mt);

if (reply != null) {

// Reply received

if (reply.getPerformative() == ACLMessage.PROPOSE) {

// This is an offer

int price = Integer.parseInt(reply.getContent()); if (bestSeller
== null || price < bestPrice) {

// This is the best offer at present bestPrice = price;

bestSeller = reply.getSender();

}

}

repliesCnt++;

if (repliesCnt >= sellerAgents.length) {

// We received all replies step = 2;

}

}

else {
block();

}

break; case
2:

// Send the purchase order to the seller that provided the best offer ACLMessage order = new
ACLMessage(ACLMessage.ACCEPT_PROPOSAL); order.addReceiver(bestSeller);

order.setContent(targetBookTitle);
order.setConversationId("book-trade");

order.setReplyWith("order"+System.currentTime
Millis()); myAgent.send(order);

// Prepare the template to get the purchase order reply
```

```
mt = MessageTemplate.and(MessageTemplate.MatchConversationId("book-trade"),
MessageTemplate.MatchInReplyTo(order.getReplyWith()));

step = 3;
break; case
3:

// Receive the purchase order reply reply =
myAgent.receive(mt);

if (reply != null) {

// Purchase order reply received

if (reply.getPerformative() == ACLMessage.INFORM) {

// Purchase successful. We can terminate System.out.println(targetBookTitle+"
successfully purchased."); System.out.println("Price = "+bestPrice);

myAgent.doDelete();

}

step = 4;

}

else {
block();

}

break;

}

}

public boolean done() {

return ((step == 2 && bestSeller == null) || step == 4);

}

} // End of inner class RequestPerformer
```

JADE предоставляет основу для реализаций обмена сообщениями по определенным протоколам в пакете *jade.protopackage*. В частности, обмен сообщениями, который был реализован в примере, поддерживается протоколом *Contract net* и может быть реализован с помощью класса *jade.proto.ContractNetInitiator*.

40. *Получение сообщений в блокирующем режиме*

Кроме метода *receive()*, класс *Agent* предоставляет также метод *blockingReceive()*, который является блокирующим вызовом, т.к. он не возвращает управление до тех пор, пока в очереди сообщений агента не появится сообщение. Перегруженная версия этого метода получает в качестве аргумента *MessageTemplate* (метод не возвращает управление до тех пор, пока в очереди сообщений агента не появится сообщение, удовлетворяющее шаблону).

Подчеркнем, что метод *blockingReceive()* фактически блокирует поток агента. Поэтому, если из некоторого режима вызывается метод *blockingReceive(),* выполнение других режимов приостанавливается до тех пор, пока метод *blockingReceive()* не вернѐт управление. Хорошим стилем программирования переговоров между агентами является использование *blockingReceive()* в методах *setup()* и *takeDown()*; использование внутри поведения метода *receive()* в сочетании с *Behaviour.block()* (как показано в пункте *Блокирование режима работы агента для ожидания сообщения*).

41. ***Сервис «желтых страниц»***

В приведенном примере было сделано допущение, что существует некое фиксированное множество агентов-продавцов (*seller1* и *seller2*) и каждый покупатель заранее знает о них (поле *AID[] sellerAgents* класса *BookBuyerAgent* в коде, приведенном в *пункте Передача аргументов агенту*). В данной главе описано, как использовать сервис «жѐлтых страниц», предоставляемый платформой JADE, чтобы позволить агентам-покупателям динамически узнавать об агентах-продавцах, доступных в данный момент времени.

42. ***Агент DF***

Агент *Directory Facilitator* (DF) – менеджер директорий (согласно спецификации FIPA) представляет службу желтых страниц (*yellow pages*), где агенты могут публиковать информацию о предоставляемых ими сервисах. С помощью DF агент может находить агентов, предоставляющих необходимые ему сервисы, и вступать с ними в переговоры. Внутри одной платформы может существовать несколько DF, например, предоставляющих информацию о различных группах сервисов или о сервисах различных групп агентов.

Каждая платформа JADE по умолчанию поддерживает агента DF (его локальное имя «df»). Могут быть запущены и объединены несколько DF- агентов (включая те, что находятся в платформе по умолчанию), для обеспечения единого распределенного каталога «желтых страниц».

43. ***Взаимодействие с DF***

Поскольку DF является агентом, имеется возможность взаимодействовать с ним стандартным образом, обмениваясь ACL-сообщениями, используя, согласно спецификациям FIPA, соответствующий язык содержания (язык *SLO*) и соответствующую онтологию (онтологию *FIPA-agent-management*). Чтобы упростить эти взаимодействия, JADE предоставляет класс *jade.domain.DFService*, с помощью которого можно публиковать и выполнять сервисы через вызовы методов.

Публикация сервисов

Агент, желающий опубликовать (сделать доступными) один или более сервисов, должен предоставить DF, включающий AID этого агента, список возможных языков и онтологий, которые должны быть известны другим агентам для взаимодействия с ним, и список сервисов для публикации. Для каждого публикуемого сервиса предоставляется описание, включающее тип сервиса, его имя, языки и онтологии, необходимые для его использования, и ещѐ некоторые свойства, специфичные для данного сервиса. Классы *DFAgentDescription, ServiceDescription* и *Property,* входящие в состав пакета *jade.domain.FIPAAgentManagement,* соответственно представляют собой три упомянутые абстракции.

Чтобы опубликовать сервис, агент должен создать описание, представленное экземпляром класса *DFAgentDescription,* и вызвать статический метод *register()* класса *DFService*. Обычно регистрация сервиса выполняется в методе *setup(),* как показано ниже в

примере с агентом-продавцом книг.

```
protected void setup() {

...

// Register the book-selling service in the yellow pages DFAgentDescription dfd
= new DFAgentDescription(); dfd.setName(getAID());

ServiceDescription sd = new ServiceDescription();
sd.setType("book-selling");

sd.setName("JADE-book-trading"); dfd.addServices(sd);

try {

DFService.register(this, dfd);

}

catch (FIPAException fe) {
fe.printStackTrace();

}

...

}
```

Заметим, что в этом простом примере не были описаны ни языки, ни онтологии, ни свойства, специфичные для данного сервиса.

Когда агент завершает работу, следует отменить регистрацию опубликованных сервисов.

```
protected void takeDown() {

// Deregister from the yellow pages try {

DFService.deregister(this);

}

catch (FIPAException fe) {
fe.printStackTrace();

}

// Close the GUI myGui.dispose();

// Printout a dismissal message

System.out.println("Seller-agent "+getAID().getName()+" terminating.");

}
```

Поиск сервисов

Агент, которому требуется найти некоторые сервисы, должен предоставить агенту DF описание необходимого шаблона. Результатом поиска является список всех описаний, которые удовлетворяют указанному шаблону. Описание соответствует шаблону, если все поля, указанные в шаблоне, присутствуют с теми же значениями и в описании.

Использование статического метод *search()* класса *DFService* можно проиллюстрировать примером, в котором агент-покупатель книг динамически находит всех агентов, предоставляемых сервисом *«book-selling»* (продажа книг).

```
public class BookBuyerAgent extends Agent {

// The title of the book to buy private String
targetBookTitle;

// The list of known seller agents private
AID[] sellerAgents;

// Put agent initializations here protected
void setup() {

// Printout a welcome message

System.out.println("Hello! Buyer-agent "+getAID().getName()+" is ready.");

// Get the title of the book to buy as a start-up argument Object[] args =
getArguments();

if (args != null && args.length > 0) { targetBookTitle = (String)
args[0]; System.out.println("Trying to buy "+targetBookTitle);

// Add a TickerBehaviour that schedules a request to seller agents every minute addBehaviour(new
TickerBehaviour(this, 60000) {

protected void onTick() {

// Update the list of seller agents

DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("book-selling");

template.addServices(sd); try {

DFAgentDescription[] result = DFService.search(myAgent, template); sellerAgents = new
AID[result.length];

for (int i = 0; i < result.length; ++i) { sellerAgents[i] =
result[i].getName();

}

}

catch (FIPAException fe) {
fe.printStackTrace();

}

// Perform the request myAgent.addBehaviour(new
RequestPerformer());

}
```

```
} );

...
```

Заметим, что обновление списка известных агентов-продавцов производится каждый раз перед очередной попыткой купить указанную книгу, т.к. агенты-продавцы в системе могут динамически появляться и исчезать. Класс *DFService* также предоставляет поддержку подписки на уведомления от DF о появлении агента, предоставляющего указанный сервис (методы *searchUntilFound()* и *createSubscriptionMessage()*).

44. ***Описание сценария «Торговля книгами»***

Пусть в сценарии будут действовать 4 продавца книг и 4 покупателя. Продавцы могут предлагать по различным ценам как одну, так и несколько книг трех наименований: «Hamlet», «Romeo and Juliet», «King Lear». Каждый покупатель стремится купить одну из 3-х книг, причем двое из них хотят купить книгу одного наименования. Каждому продавцу и каждому покупателю сопоставляется собственный агент. Индивидуальные значения свойств агентов определены в Таблица 1.

Таблица 1 – Значения свойств агентов

Агент	**Свойства**
BookSeller1	Название книги = Hamlet, Цена = 100 Название книги = King Lear, Цена = 320
BookSeller2	Название книги = Hamlet, Цена = 150
BookSeller3	Название книги = Romeo and Juliet, Цена = 200 Название книги = Hamlet, Цена = 120
BookSeller4	Название книги = Romeo and Juliet, Цена = 250 Название книги = King Lear, Цена = 300
BookBuyerHamlet	Название книги = Hamlet
BookBuyerRomeoAndJuliet	Название книги = Romeo and Juliet
BookBuyerKingLear	Название книги = King Lear
BookBuyerHamlet2	Название книги = Hamlet

Агенты продавцов и покупателей имеют определенную модель поведения и критерии принятия решения по выбору приемлемого варианта решения (Таблица 2).

Таблица 2 – Описание модели поведения агентов

Наименование агента	**Цель**	**Необходимые параметры**	**Критерии выбора**	**Особенности**
Агент продавца	Продать весь имеющийся в наличии товар	Список продаваемых книг с указанием	Продажа книги первому обратившемуся	Не выполняется просмотр
	(книги)	наименования и цены за один экземпляр	покупателю по первоначально заявленной цене	списка покупателей, продажа книги по одной первому обратившемуся покупателю

Агент покупателя	Приобрести определенный вид товара (книгу с заданным наименованием)	Наименование книги, которую необходимо приобрести	Выбор среди всех продавцов того, который предлагает книгу с подходящим наименованием по минимальной цене	Принятие решения о покупке после первого просмотра списка всех продавцов, предлагающих книги. Удаление агента из системы после удачной покупки

45. ***Последовательность выполнения сценария «Торговля книгами»***

Создадим агентов продавцов книг с помощью RMA. Для этого выберем пункт главного меню *Actions -> Start New Agent*. Назовем агента *BookSeller1* (Рис. 1). Нажмем кнопку <*OK*>.

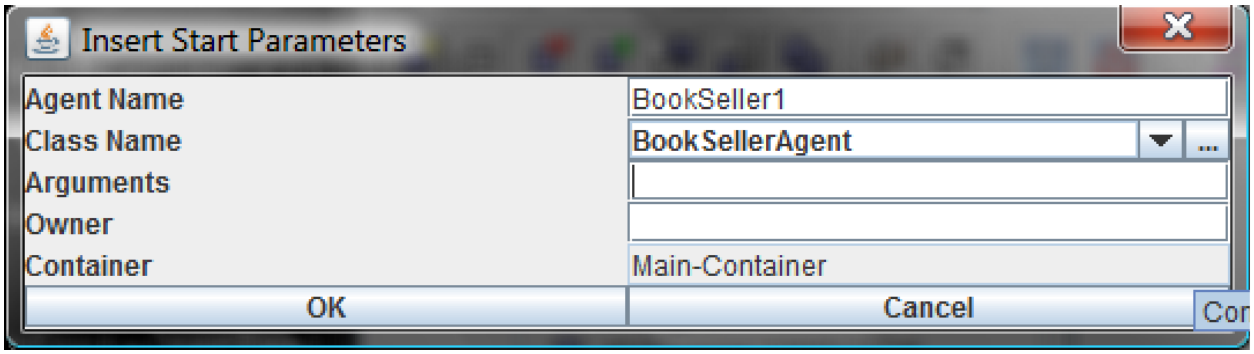


Рис. 1 Форма создания агента продавца

На экране появится форма для ввода наименования книги и ее цены (Рис. 2). После каждого нажатия на кнопку <OK> будет создано новое наименование книги с заданными в форме параметрами для агента продавца, имя которого указано в заголовке формы.

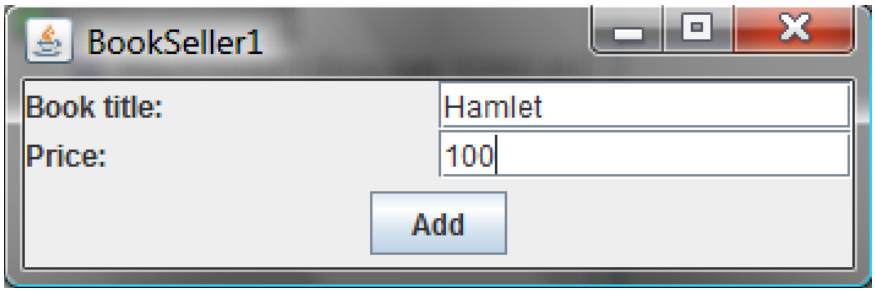


Рис. 2. Форма ввода параметров агента продавца

Примечание: при закрытии формы на Рис. 2 агент прекращает работу! Поэтому формы ввода параметров для всех агентов продавцов должны все время быть открытыми!
Выполним аналогичные действия для создания всех 4-х агентов продавцов с набором книг для каждого из них (на Рис. 3 показана форма создания агента второго продавца).

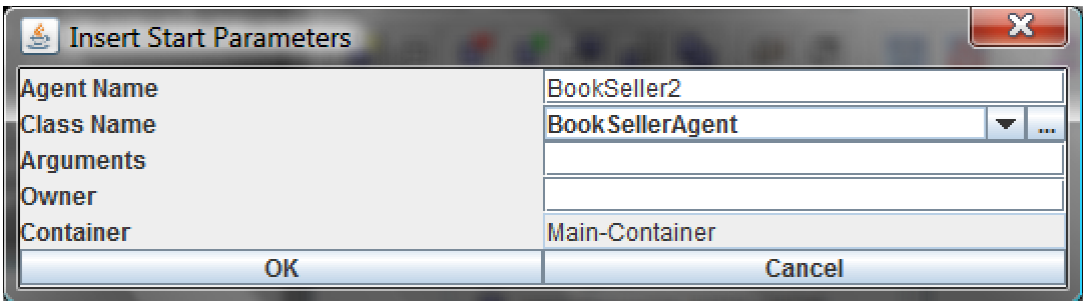


Рис. 3 Форма создания второго продавца

После создания и ввода параметров всех агентов продавцов контейнер по умолчанию в RMA выглядит, как показано на Рисунок 4.

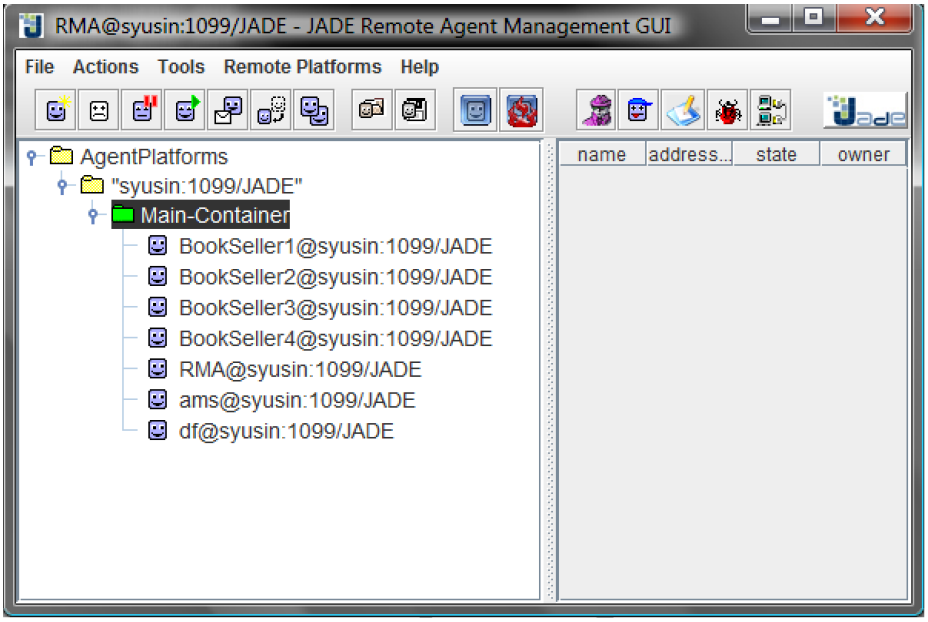


Рис. 4 Контейнер по умолчанию с зарегистрированными агентами продавцов в RMA

Окно командной строки показано на Рис. 5.

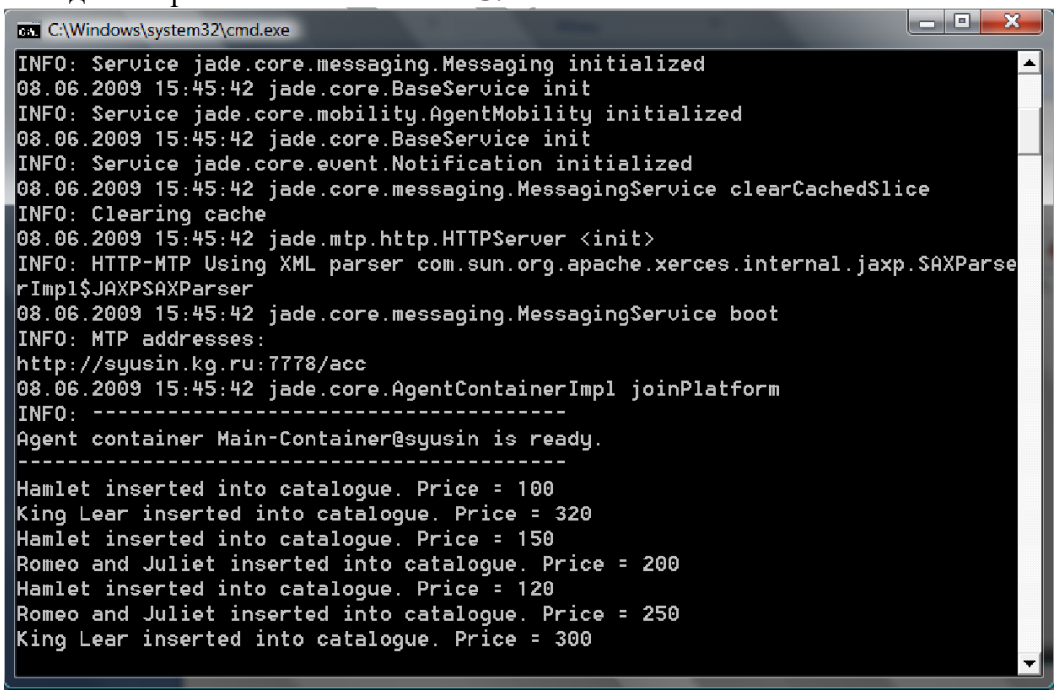


Рис. 5 Окно командной строки после ввода всех агентов продавцов

Создадим агента покупателя книг. Для этого выберем пункт главного меню *Actions -> Start New Agent*. Назовем агента *BookBuyerHamlet* (он хочет купить книгу «Hamlet»). Наименование книги, которую хочет купить агент, указывается в поле *Arguments*. На Рис. 6 показана форма создания агента покупателя книги «Hamlet».

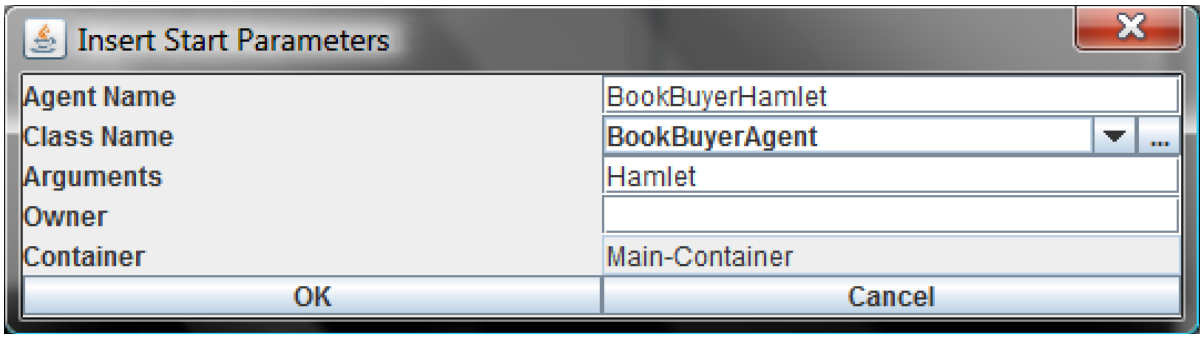


Рис. 6 Форма создания агента покупателя книги «Hamlet»

После создания агент регистрируется в системе и сразу же начинает выполнять матчинг для поиска соответствия наименования требуемой книги и наименований предлагаемых книг. Агент покупателя посылает запросы агентам продавцов периодически через определенный интервал времени.

Создадим еще двух агентов покупателей книг, как показано на Рис. 7, Рис. 8.

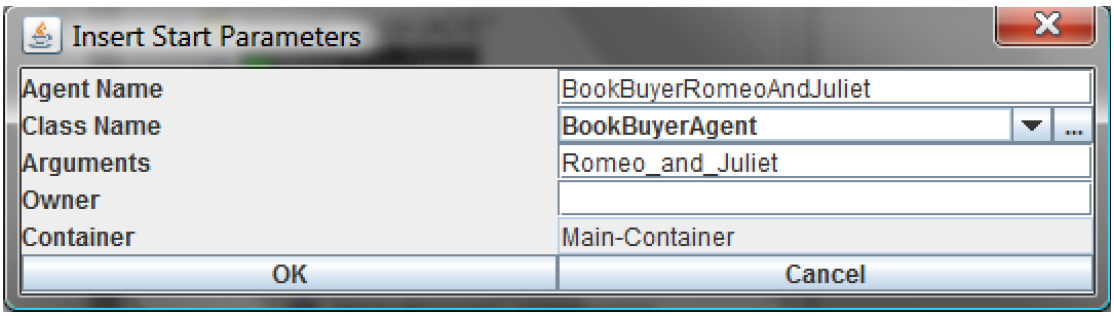


Рис. 7 Форма создания агента покупателя книги «Romeo and Juliet»

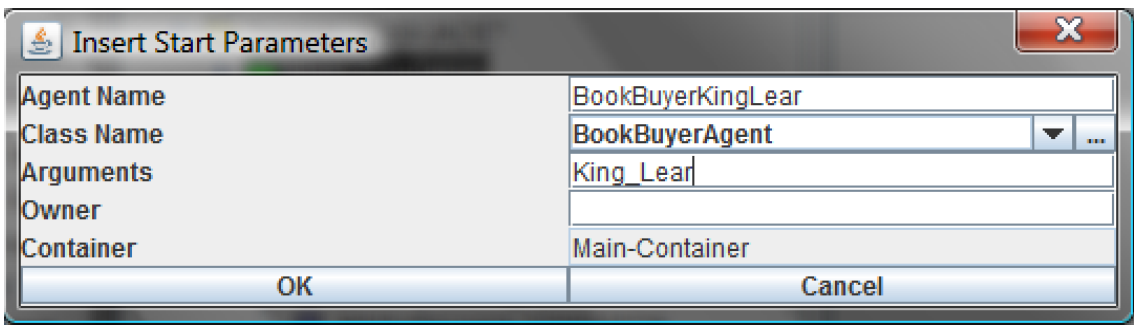


Рис. 8 Форма создания агента покупателя книги «King Lear»

Примечание: наименования книг в форме создания не должны содержать пробелов, т.к. система обрезает наименования до первого пробела и, соответственно, искомая книга найдена не будет (хотя агент продавца и позволяет вводить наименования книг с пробелами). Поэтому пробелы должны заменяться в наименовании, например, нижним подчеркиванием. Пример неудачного ввода наименования книги с пробелом представлен на Рис. 9. Наличие пробела привело к сокращению наименования «Romeo and Juliet» до «Romeo» и матчинг был осуществлен неуспешно.

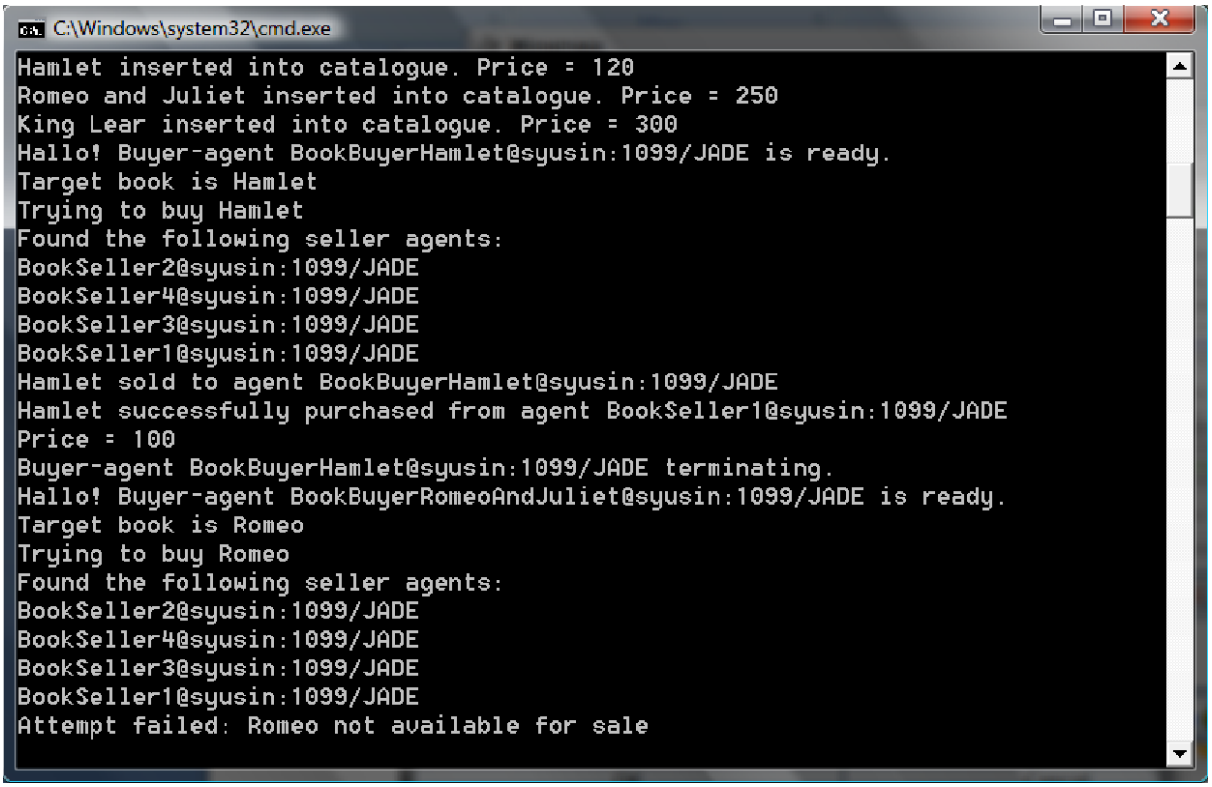


Рис. 9 Ошибочный ввод наименования книги

После регистрации всех агентов покупателей они сразу вступают в процесс матчинга. Каждый агент покупателя находит наиболее подходящего продавца (предлагающего наименьшую цену за искомую книгу) и создает с ним связь. Результат матчинга представлен на Рис. 10.

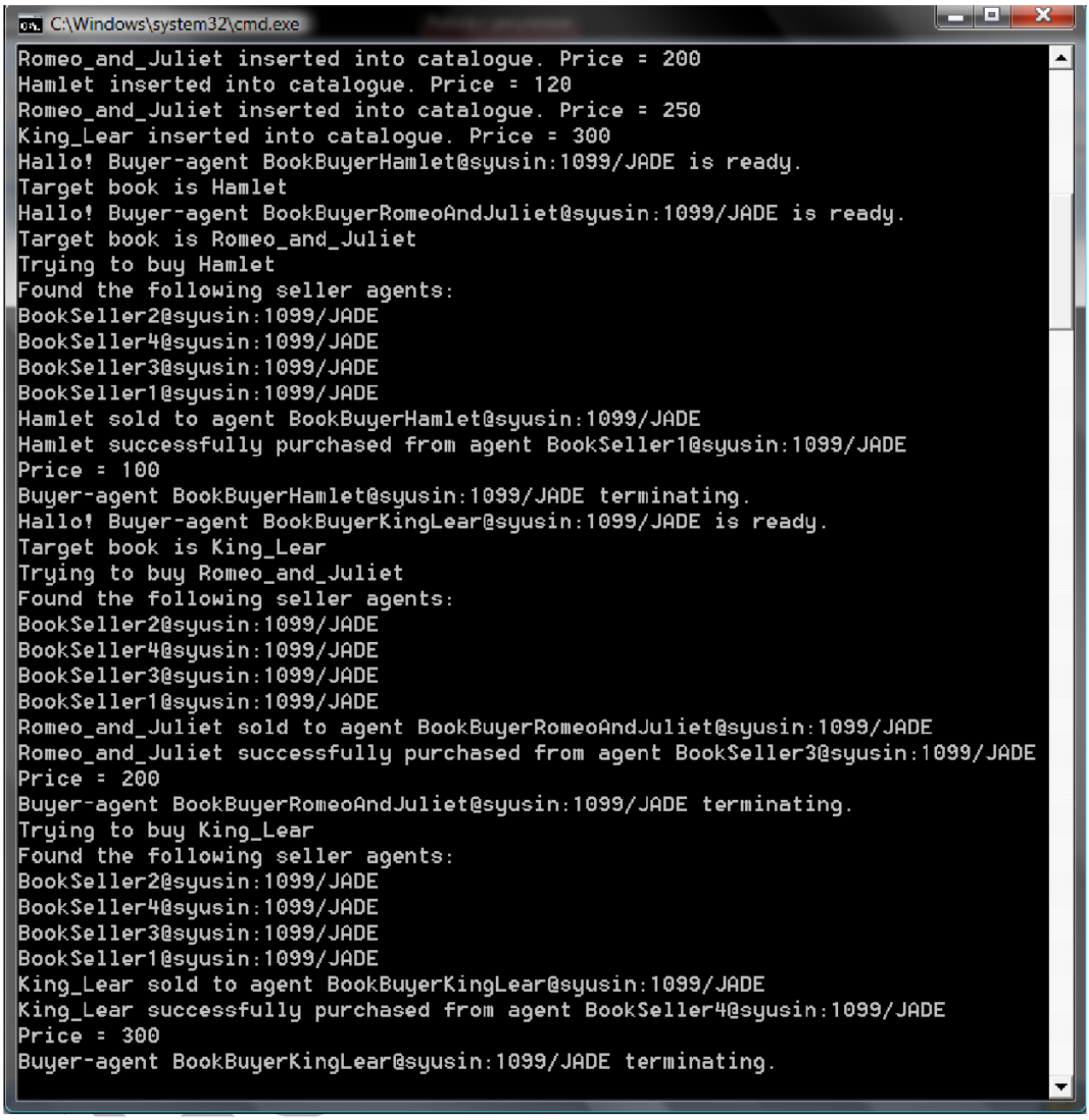


Рис. 10 – Результат матчинга агентов

Зарегистрируем еще одного покупателя книг BookBuyerHamlet2, который также стремится купить книгу с наименованием «Hamlet» (Рис. 11).

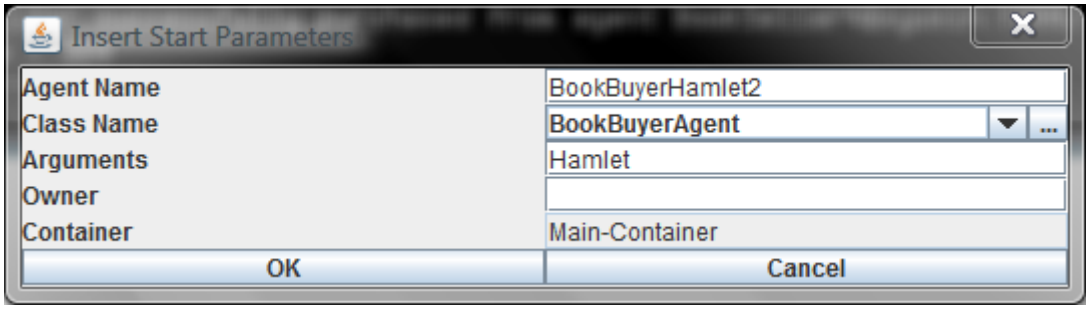


Рис. 11 Форма создания агента BookBuyerHamlet2

В результате матчинга новый агент успешно покупает книгу, установив связь с продавцом *BookSeller3*, выбрав минимальную цену из оставшихся вариантов. Таким образом, новые агенты будут и далее устанавливать связи в порядке возрастания цены на книги (поскольку в данном примере это единственный количественный критерий отбора) пока не будут задействованы все варианты. Результат матчинга для агента *BookBuyerHamlet2* приведены на Рис. 12.

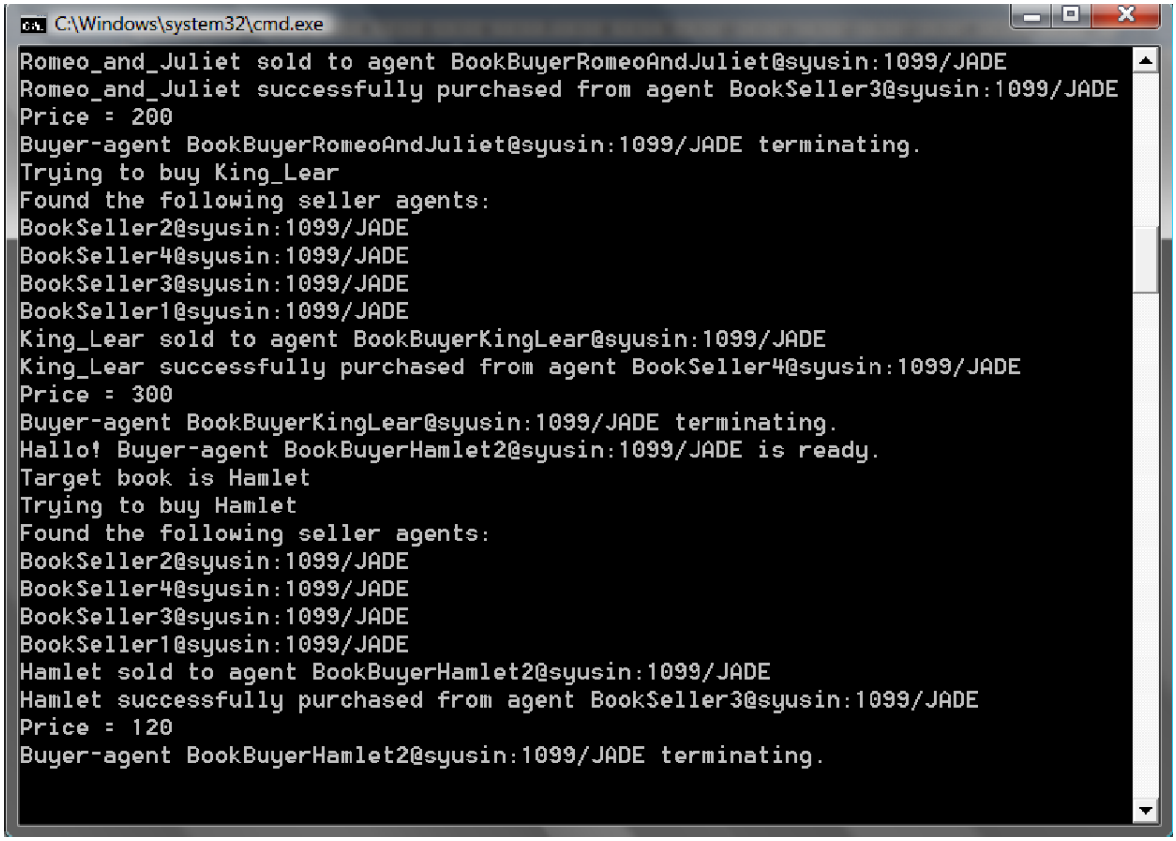


Рис. 12 Результат матчинга для агента BookBuyerHamlet2

9.2 Методические рекомендации по подготовке письменных работ

Порядок составления и оформления отчета о практической работе

В значительной мере эффективность решения задачи по выполнению практической работы зависит от качества соответствующего отчета. Для этого необходимо соблюдать следующие основные требования по составлению и оформлению отчета, обусловленные соответствующими нормативными документами. Текст отчета должен быть лаконичным и вместе с тем информативным. Текст должен быть изложен с соблюдением правил грамматики. Отчет составляется с обязательным составлением следующих разделов:

1. Заголовок отчета.
2. Цели работы.
3. Методика работы.
4. Порядок выполнения работы (этапы работы).
5. Выводы по работе.

1. В **заголовке отчета** приводятся наименования идентифицирующих признаков: **Отчет о практической работе № 1** по теме, например, «**Установка IDE NetBeans, подключение библиотеки JADE, создание простого агентного приложения**», ниже указываются данные студента (фамилия и инициалы, вид обучения, специальность, курс, группа).

2. В разделе **Цель работы** формулируется цели работы студента в соответствии с содержанием раздела «Постановка задачи» данной работы и индивидуального задания студенту на работу.

3. В разделе **Методика работы** указывается методика работы в соответствии с имеющейся формулировкой в разделе «Методика работы» данной работы и при необходимости уточняется в зависимости от содержания конкретного варианта задания студенту на практическую работу.

4. **Порядок выполнения работы**. Приводятся номера и наименования этапов работы, предусмотренные для работы данного Практикума. По каждому из этапов приводится описание выполненных студентом работ, направленных на достижение цели работы. Пропуск какого-либо из этапов работы Практикума не допускается. В рамках этапов помещается соответствующий иллюстративный материал - таблицы, рисунки (графики), полученные по ходу решения задачи работы. Обозначение иллюстративного материала выполняется в соответствии с правилами, принятыми для публикаций. Обозначение каждой таблицы и рисунка должно иметь номер и наименование. Внутри каждого отчета таблицы и рисунки обозначаются соответственно сквозными номерами. Обозначение таблицы указывается над таблицей, а обозначение рисунка под рисунком. Приводимые в тексте данной работы примеры включать в отчет не разрешается. Применяется только материал, полученный в ходе работы студентом по соответствующему заданию, полученному от преподавателя.

5. Последним разделом отчета являются **выводы** по работе. Это самая сложная и трудная часть работы. Очень важно, чтобы выводы отражали методику, технологию, применяемые программно-аппаратные средства решения задачи. Полезно каждому из этапов работы формулировать не менее одного вывода. Вывод может содержать от одного до трех предложений. Формулировки выводов должны быть конкретными, информативными, лаконичными, по возможности подкрепляться количественными данными.

Оформление отчета выполняется с учетом общепринятых правил. Графическая часть отчетов должна соответствовать правилам графического оформления. Текст отчета набирается в редакторе Word через 1,5 интервала, 14 кегль. Следует использовать шрифт Times New Roman. Заголовки разделов и подразделов выделяются жирным шрифтом. После окончания оформления отчета он проверяется студентом на предмет качество содержания и формы. При условии обнаружения ошибок последние исправляются. После устранения дефектов отчета его экранная форма, или принтерная распечатка предъявляется преподавателю. При условии обнаружения преподавателем ошибок в отчете студент их исправляет и предъявляет отчет преподавателю повторно. Если ошибок нет, то отчет принимается и сохраняется на жестком диске.

Отчет по работе сохраняется студентом в виде отдельного файла. В имени файла указывается фамилия студента и номер выполненной работы. Файл сохраняется в папке с фамилией студента в папке соответствующей студенческой группы. Папка группы создается на первом занятии. В имени папки группы должен присутствовать индекс группы. Папка группы включается в папку «Мои документы».

Приложение 1. Аннотация
рабочей программы дисциплины

АННОТАЦИЯ РАБОЧЕЙ ПРОГРАММЫ ДИСЦИПЛИНЫ

Цель дисциплины: приобретение знаний, навыков и умений в области проектирования и разработки мультиагентных систем.

Задачи: сформировать у студентов понятия о роли и месте мультиагентного подхода к решению задач в области информатизации и автоматизации систем управления, о его достоинствах и ограничениях. Сформировать знания об основных видах агентных архитектур и стратегиях управления мультиагентными коллективами и познакомить с ними на практике. Предоставить информацию о назначении и основных характеристиках существующих мультиагентных систем и их функциональных возможностях. Сформировать у студентов навыки самостоятельной разработки мультиагентных систем.

В результате освоения дисциплины обучающийся должен:

знать

общие принципы построения, основные свойства и архитектуры автономных агентов;

методологию, методы и модели формирования МАС;

о базовых ситуациях, режимах и моделях взаимодействия, коммуникации, кооперации;

программные языки и инструментальные средства реализации искусственных агентов.

уметь

осуществлять синтез искусственных агентов различных классов и выбор эффективных архитектур МАС для конкретных, специфических приложений;

программировать агентов с использованием языков ориентированного программирования, библиотек агентов и агентских сред; разработки структур коммуникации агентов на основе стандарта ACL (Agents Communication);

применять восходящее и нисходящее проектирование МАС.

владеть

вопросами о причинах появления и основных направлениях развития теории агентов и МАС как стратегической области информатики и искусственного интеллекта;

важнейшими способами разработки агентов (поведенческая, деятельностная, логическая, лингвистическая, программистская и пр.) и формализмах описания мультиагентных систем различных классах;

методами моделирования поведения и действий агентов.