**Курс: «Технологии обработки информации».**

**Лабораторная работа № 1.**

**Агентная платформа JADE: разработка тестового приложения**

доцент кафедры информатики и вычислительной техники Мурманского арктического государственного университета, филиал в г. Апатиты,

к.т.н., Тоичкин Николай Александрович

Содержание

[1. Назначение агентной платформы JADE и средства, предоставляемые разработчику агентных систем 2](#_Toc441279172)

[1.1 Архитектура агентной платформы JADE 3](#_Toc441279173)

[2. Настройка JADE. 5](#_Toc441279174)

[3. Полезные утилиты 8](#_Toc441279175)

[3.1 Remote management agent 8](#_Toc441279176)

[3.2 Dummy agent 8](#_Toc441279177)

[3.3 Sniffer management agent 9](#_Toc441279179)

[3.4 Introspecor management agent 9](#_Toc441279181)

[3.5 Log management agent 10](#_Toc441279182)

[3.6 DF GUI 10](#_Toc441279183)

[4. Создание тестовых агентов 11](#_Toc441279184)

[5. Импорт существующего проекта в Eclipse 15](#_Toc441279185)

[6. Источники 15](#_Toc441279186)

[Приложение 1 16](#_Toc441279187)

[Приложение 2 17](#_Toc441279189)

# Назначение агентной платформы JADE и средства, предоставляемые разработчику агентных систем

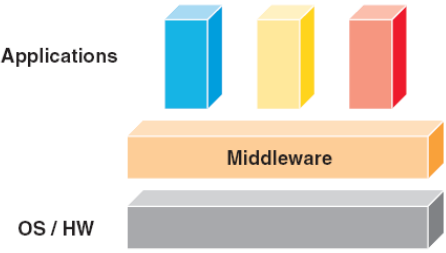
*Агент* – это программный объект, способный воспринимать ситуацию, принимать решения и коммуницировать с подобными себе объектами, динамически устанавливая с ними связи. Под *мультиагентной системой* (МАС) будем понимать множество программных агентов, организованных в одно или несколько сообществ, и предназначенных для решения определенной задачи.

Средой разработки и существования МАС являются *агентные* *платформы*. Было разработано множество программных реализаций агентных платформ, каждая из которых имеет свои особенности, достоинства и недостатки. Вот лишь небольшой список из более чем ста доступных платформ, публикуемых на сайте организации *AgentLink* (*European Coordination Action for* *Agent-based Computing*): *JADE, FIPA-OS, AOS, ZEUS, KADOMA, NOMADS, ARA, AGLETS, GRASSHOPPER, TRACY, AJANTA, LEAP, JACK, SEMOA.* Многие из них успешно существуют в виде коммерческих проектов (таких как *JACK*) или проектов, позиционируемых как проекты с открытым исходным кодом (*JADE*, *ZEUS* и др.).

В 90-х годах возникла необходимость создания единых стандартов на разработку агентных систем. В этот период были основаны две организации *MASIF* (*Mobile Agent System Interoperability Facility*) и *FIPA* (*Foundation of* *Physical Intelligent Agents*). В результате их работы появились стандарт *MASIF* и стандарт *FIPA*, дающие рекомендации по созданию систем мобильных агентов и систем интеллектуальных агентов.

Одной из наиболее популярных агентных платформ в настоящее время является платформа *JADE (Java Agent DEvelopment Framework)*. Проект JADE разрабатывается компанией *Telecom Italia Lab* с 2000 г.

Агентная платформа *JADE* является типичным *Middleware*, т.е. программным обеспечением (ПО) среднего уровня (рис. 1), преставляющим собой набор средств для создания и управления системой с множеством агентов.



**Рис. 1. Место Middleware в структуре ПО**

Платформа разработки мультиагентных систем *JADE* включает в себя динамическую среду, где могут «жить» *JADE* агенты; библиотеку классов, которую программисты могут использовать для разработки собственных агентов; набор графических инструментов, позволяющих управлять активностью запущенных агентов.

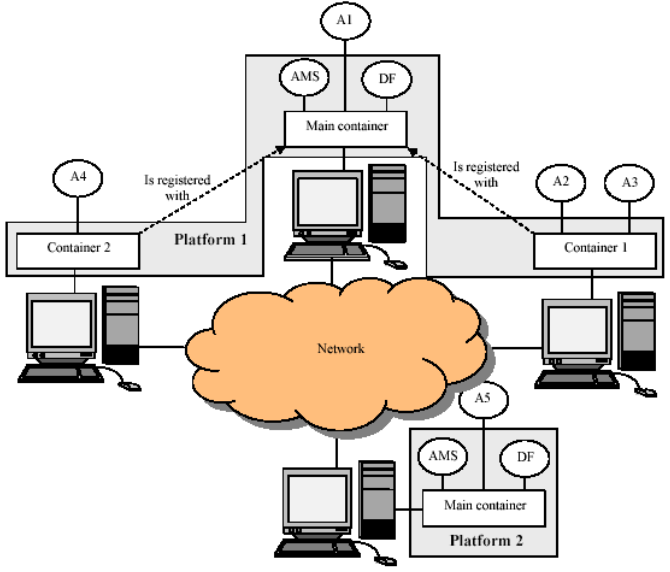
*JADE* предоставляет программисту – разработчику агентных систем следующий набор средств:

* *FIPA-compliant Agent Platform* – агентную платформу, основанную на стандарте *FIPA* и включающую обязательные типы системных агентов, которые автоматически активируются при запуске платформы.
* *Distributed Agent Platform* – распределенную агентную платформу, которая может использовать несколько компьютеров (узлов), причем на каждом узле запускается только одна *Java* *Virtual* *Machine*. Агенты исполняются как *Java*-потоки. Для доставки сообщений между агентами, в зависимости от их местонахождения, используется соответствующий транспортный механизм – *Multiple Domains support* – множество основанных на *FIPA*-спецификациях специализированных агентов, которые могут объединяться в федерацию, реализуя таким образом мультидоменную агентную среду.
* *Multithreaded execution environment with two-level scheduling*. Каждый *JADE*-агент имеет собственный поток управления, но он также способен работать в многопотоковом режиме. *Java Virtual Machinе* проводит планирование задач, исполняемых агентами или одним из них.
* *Object-оriented programming environment*. Большинство концепций, свойственных *FIPA*-спецификации, представляются *Java*-классами, формирующими интерфейс пользователя.
* *Library of interaction protocols*. Использование стандартных интерактивных протоколов *fipa-request* и *fipa-contract-net*. Для того чтобы создать агента, который мог бы действовать согласно таким протоколам, разработчикам прикладных программ нужно только имплементировать специфические доменные действия, в то время как вся независимая от прикладной программы протокольная логика будет осуществляться системой JADE.
* *Administration GUI*. Простые операции управления платформой могут исполняться через графический интерфейс, отображающий активных агентов и контейнеры агентов. Используя GUI, администраторы платформы могут создавать, уничтожать, прерывать и возобновлять действия агентов, создавать иерархии доменов и мультиагентные федерации.

Платформа *JADE* написана на языке программирования *Java* с использованием *Java* *RMI*, *Java* *CORBA* *IDL*, *Java* *Serialization* и *Java* *Reflection* *API*. Она упрощает разработку мультиагентных систем благодаря использованию *FIPA*-спецификаций и инструментов (tools), которые поддерживают фазы исправления ошибок (debugging) и развертывания (deployment) системы. Эта агентная платформа может распространяться среди 10 компьютеров с разными операционными системами, и ее можно конфигурировать через удаленный *GUI*-интерфейс. Процесс конфигурирования этой платформы достаточно гибкий. Единственным требованием такой системы является установка на компьютере *Java* *Run* *Time* требуемой версии.

## Архитектура агентной платформы JADE

Платформа *JADE* является распределенной и представляет собой набор *контейнеров* (рис. 2).

****

**Рис. 2. Платформа JADE**

*Контейнером* называется динамическая среда исполнения мультиагентных приложений, в которой находятся агенты. Каждый контейнер может содержать несколько агентов. Набор активных контейнеров называется *платформой*. Один из контейнеров всегда является главным (*Main container*), все остальные контейнеры связываются с ним и регистрируются в момент запуска. Поэтому первым контейнером при старте платформы должен быть главный, а все остальные контейнеры должны быть «обыкновенными» (т.е. ненеглавными) контейнерами и должны заранее «знать», как найти главный контейнер, на котором они будут регистрироваться, т.е. должны иметь данные о хосте и порте.

Другой главный контейнер, запущенный где-либо в сети, представляет собой другую платформу, на которой могут зарегистрироваться новые обычные контейнеры. Рис. 1 иллюстрирует эту концепцию на основе примера, показывающего две *JADE*-платформы, состоящие из трех и одного контейнера соответственно. *JADE*-агенты определяются с уникальными именами. При условии, что они знают имена других агентов, они могут общаться, независимо от их фактического местонахождения: в общем контейнере (агенты *А2* и *А3*), в разных контейнерах на одной платформе (агенты *А1* и *А2*), или вообще на разных платформах (*А4* и *А5*). Пользователю не обязательно знать, как работает динамическая среда *JADE*, но необходимо запускать ее перед началом выполнения своих агентов.

***Агенты AMS и DF***

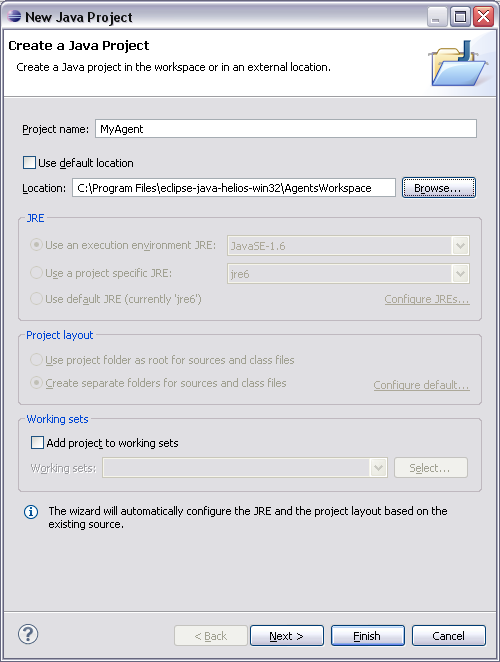
Кроме возможности приема регистраций от других контейнеров, главный контейнер отличается от обычного контейнера тем, что содержит два специальных агента, автоматически запускаемых одновременно с контейнером:

* *AMS* (*Agent Management System* – система управления агентами) обеспечивает службу управления агентами, которая позволяет создавать и удалять агентов, а также содержит в себе пространство имен агентов. Имя агента является уникальным и имеет следующий формат: *<nickname>@<platform-name>*. Зная имена друг друга, агенты могут обмениваться сообщениями как внутри контейнера и платформы, так и между различными платформами.
* *DF* (*Directory Facilitator* – менеджер директорий) представляет собой 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*, предоставляющих информацию о различных группах сервисов или о сервисах различных групп агентов.

# Настройка JADE.

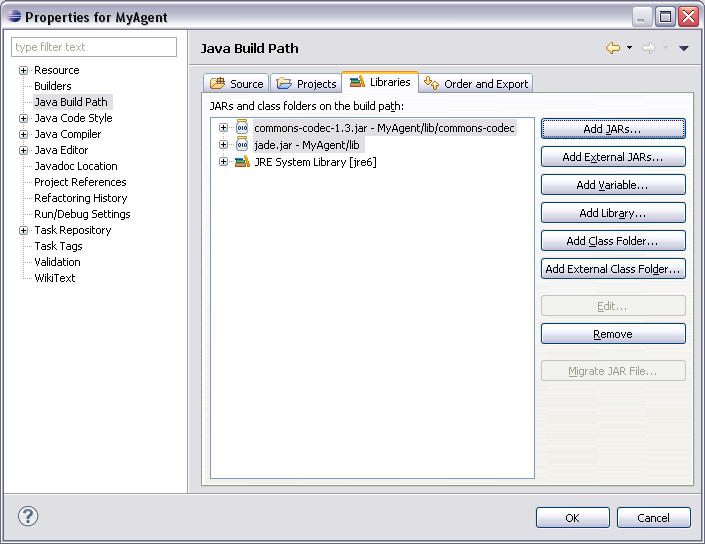
Для начала работы необходимо установить и настроить агентную платформу JADE, для этого надо выполнить следующие шаги:

1. Скачать дистрибутив *JADE* (архив **jadeAll.zip** с официального сайта проекта [1], который доступен после регистрации и авторизации; *WADE* качать не нужно!).
2. Скачать и установить *JDK* и *Eclipse* [2,3].
3. Выбрать место для извлечения архива: папку, с коротким путем, содержащим только английские названия.
4. Создать проект. В *Eclipse*чтобы создать новый *java*-проект:
   1. Выберите пункт меню **File -> New -> Java Project**;
   2. Введите имя проекта;
   3. Нажмите **Finish**.



**Рис. 3. Создание нового java-проекта в среде Eclipse**

1. Подключение библиотеки JADE к проекту:
   1. *В операционной системе:*
      * В папке проекта создайте подпапку \lib;
      * Скопируйте все файлы \*.jar из папки \JADE-bin-4.0\jade\lib\ и всех ее подпапок в папку \lib. Обычно это jade.jar и commons-codec-1.3.jar;
   2. *В Eclipse:*
      * Щелкните правой кнопкой на проекте и из контекстного меню выберите **Refresh**– папка появится в проекте;
      * Щелкните правой кнопкой на проекте и из контекстного меню выберите **Properties;**
      * В окне **Properties** выберите слева раздел **Java Build Path;**
      * В разделе **Java Build Path** выберите закладку **Libraries;**
      * Нажмите справа кнопку **Add JARs**, выберите скопированные \*.jar библиотеки, нажмите ОК.



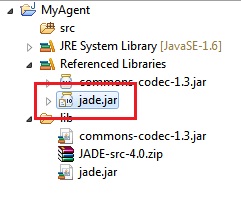
**Рис. 4. Окно свойств проекта – Java Build Path**

1. Подключение исходного кода JADE к библиотекам. Для того, чтобы отладчиком заходить внутрь системы JADE или просто смотреть код JADE, необходимо подключить исходный код JADE к библиотеке **jade.jar**, для этого:
   1. *В операционной системе:*
      * Скопируйте файл JADE-src-4.0.zip в папку \lib.
   2. *В Eclipse:*
      * Щелкните правой кнопкой на проекте и из контекстного меню выберите **Refresh**– файл JADE-src-4.0.zip появится в проекте;
      * В окне **Package Explorer**раскройте раздел **Referenced Libraries;**
      * Щелкните правой кнопкой на библиотеке **jade.jar** и из контекстного меню выберите **Properties**;
      * В окне **Properties** выберите слева раздел **Java Source Attachment**(рис. 5);
      * Нажмите кнопку **Workspace** и укажите файл с исходным кодом JADE.



**Рис. 5. Окно свойств архива**

Замечаем, что иконка библиотеки **jade.jar** изменилась (рис. 6).



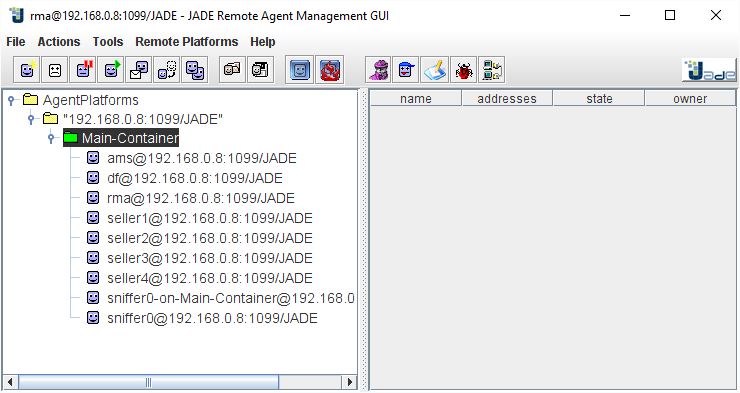
**Рис. 6. Окно Package Explorer**

# Полезные утилиты

## Remote management agent

*Remote management agent* (рис. 7) представляет собой графическую  
консоль для управления мультиагентным приложением. Позволяет создавать  
новые контейнеры, управлять агентами, создавать сообщения и запускать  
средства отладки. Открывается при запуске агента с параметром –gui.

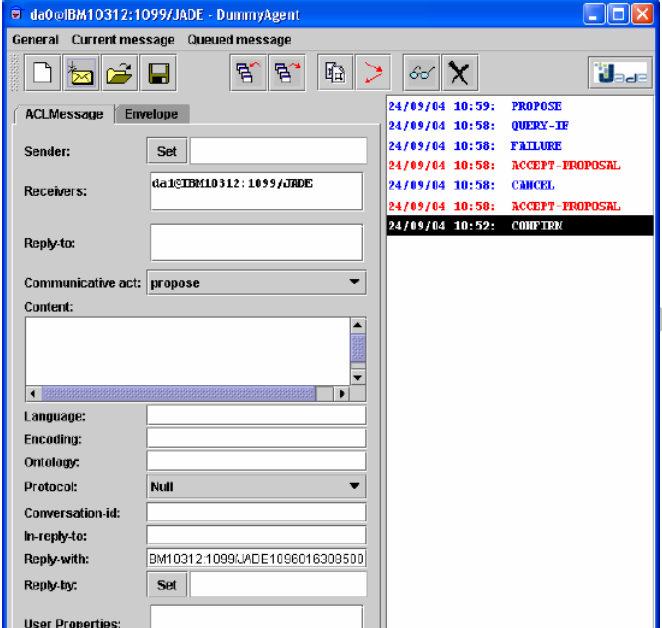
Доступ к остальным утилитам, рассматриваемым в данной главе осуществляется из RMA.



**Рис. 7. RMA**

## Dummy agent

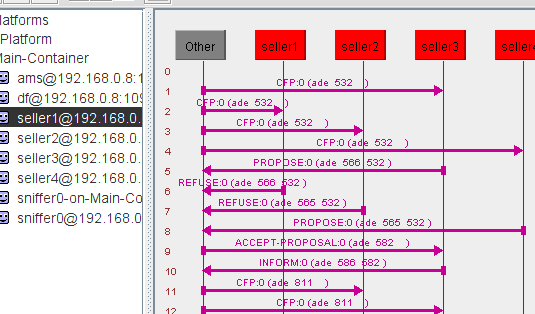
# *Dummy agent* (рис. 8) является графической утилитой, которая позволяет посылать и получать сообщения от имени определенного агента, а также сохранять и загружать очередь его сообщений (отправленных и полученных).



**Рис. 8. Dummy agent**

## Sniffer management agent

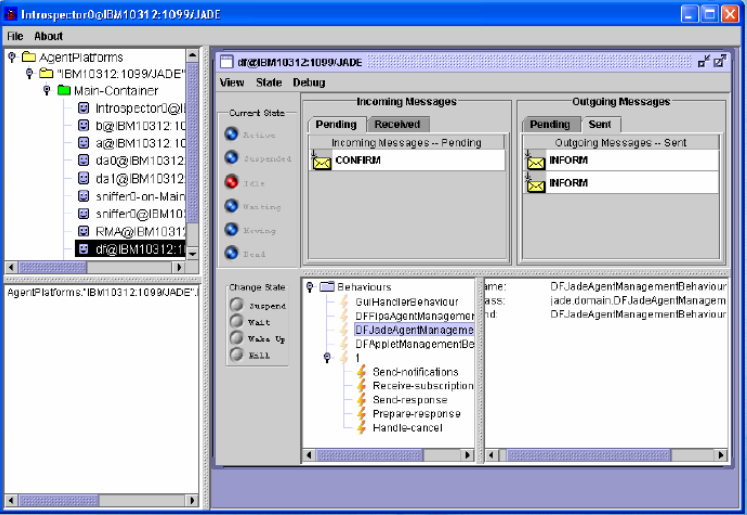
# *Sniffer agent* (рис. 9) – это графическая утилита для просмотра потока сообщений между избранными агентами. Представляет обмен сообщениями в виде диаграмм последовательностей. Позволяет сохранять/загружать поток сообщений между агентами.



**Рис. 9. Sniffer agent**

## Introspecor management agent

*Introspector agent* (рис. 10) – графическая утилита для просмотра внутреннего состояния агента. Позволяет контролировать жизненный цикл агента, просматривать очередь его сообщений, активные и выполненные поведения, а также запускать исполнение агента с задержками между операциями или по шагам. При этом «шагом» поведения агента считается исполнение метода *action()*, а не команда в коде языка Java.

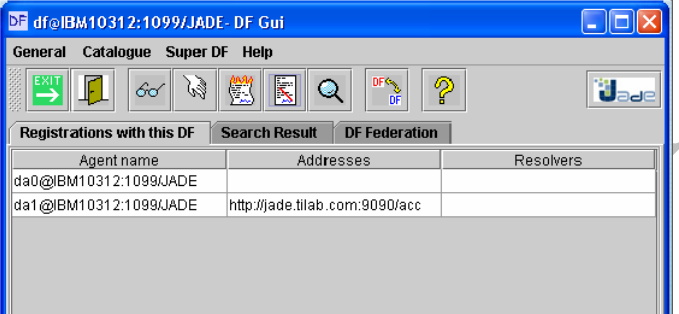
****Рис. 10. Introspector agent**

## Log management agent

*Log Manager agent*  – графическая утилита для отображения лога сообщений в процесе работы агентного приложения.

## DF GUI

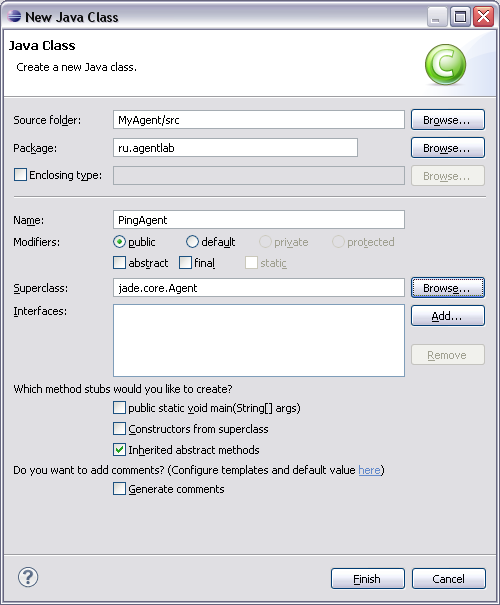
*DF GUI* (рис. 11) – графическая утилита для визуализации желтых страниц. Позволяет регистрировать и удалять сервисы агентов, а также осуществлять поиск сервисов.



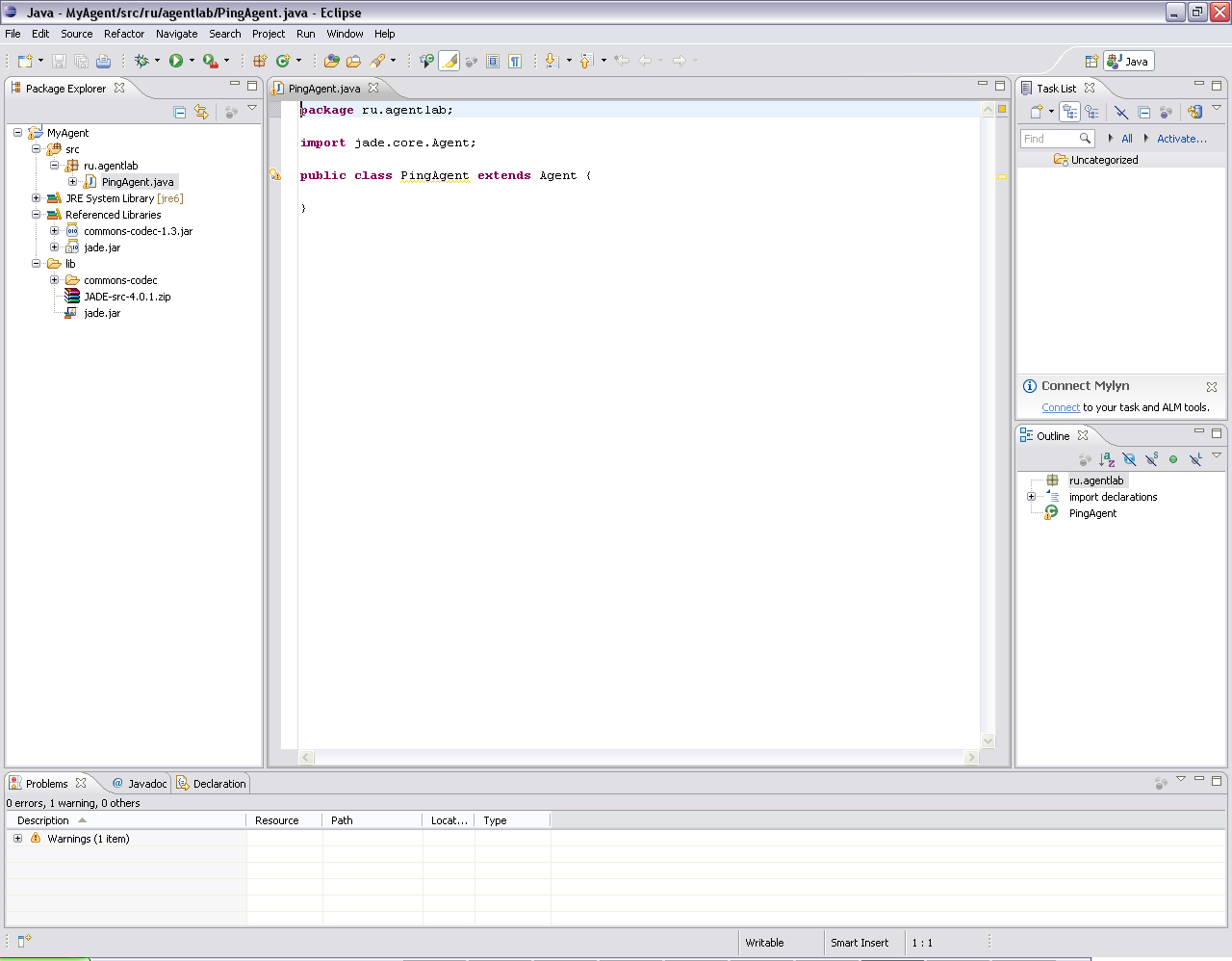
**Рис. 11. DF GUI**

# Создание тестовых агентов

* 1. Написание тестовых агентов: Рассмотрим тривиальную задачу: два агента постоянно обмениваются сообщениями между собой (пинг-понг). В Eclipse:
  + Нажмите правой кнопкой мыши на папке проекта в **Eclipse;**
  + Выберите пункт меню **New -> Class;**
  + Заполните поля как показано на рисунке (рис. 12) и нажмите **Finish**;

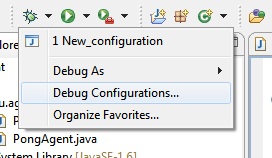


**Рис. 12. Создание нового проекта**

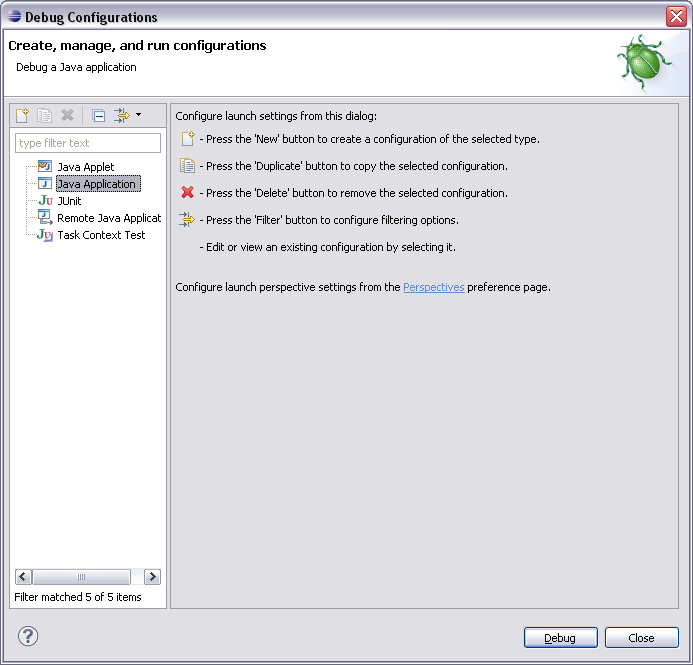


**Рис. 13. Окно редактора кода**

* + Отредактируйте файл первого агента PingAgent (см. приложение 1)**;**
  + Создайте и отредактируйте файл второго агента PongAgent (см. приложение 2).
  1. Запуск тестового агента:
  + В панели инструментов **Eclipse** откройте выпадающее меню **Debug** и выберите пункт **Debug Configurations...**(то же самое можно сделать из меню Run);

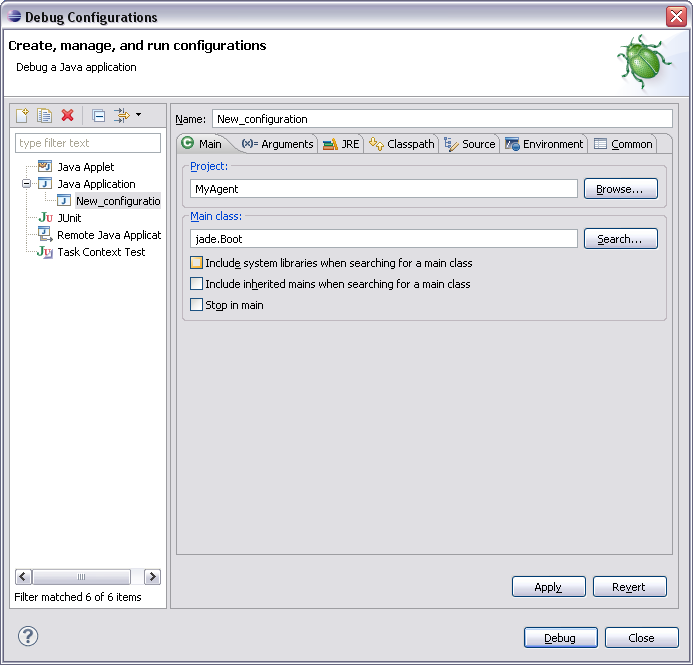


**Рис. 14.Выбор конфигурации запуска**



**Рис. 15. Debug Configuration (шаг 1)**

* + В окне **Debug Configurations**во вкладке **Main**введите имя вашего проекта и класс запуска **jade.Boot**;



**Рис. 16. Debug Configuration (шаг 2)**

* + В окне **Debug Configurations**во вкладке **Arguments** введите параметры без кавычек *"-gui test1:ru.agentlab.PingAgent;test2:ru.agentlab.PongAgent"*;

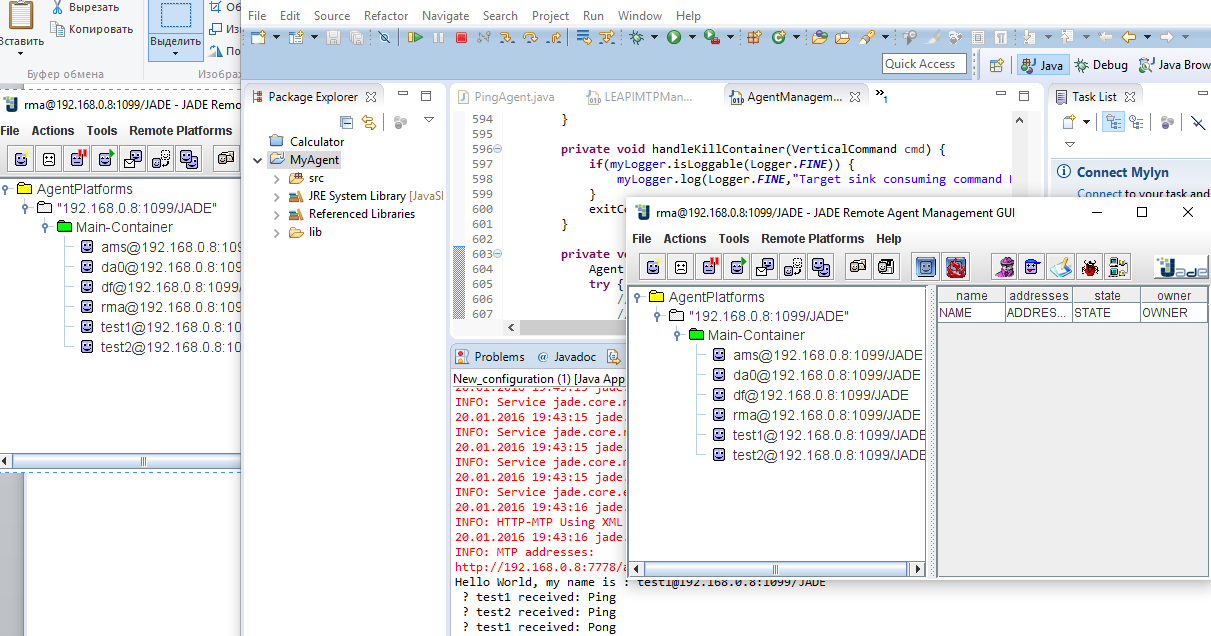
где:

* + - * test1 и test2 — произвольные строковые имена агентов;
      * ru.agentlab  — название java-пакета, в котором находится java-класс агента;
      * PingAgent и PingAgent — имена java-классов агентов.



**Рис. 17. Debug Configuration (шаг 3)**

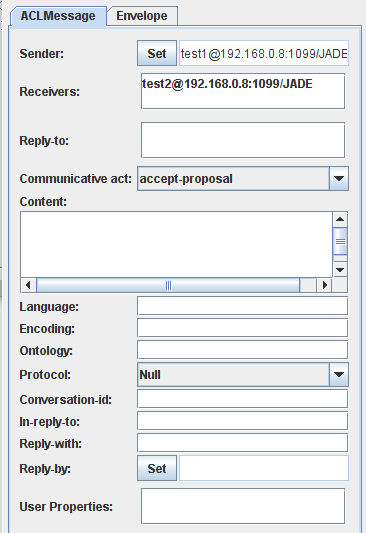
* + В окне **Debug Configurations**нажмите кнопку **Debug**;
  + Информация о запуске и ошибках выводится в консоль. В результате запуска вы увидите **окно Remote Agent Management GUI**.



**Рис. 18. Запуск агентов**

Как видно из консоли, агент test\_1 отправил сообщение ”Ping”, сам же его получил и вывел на экран. Также получил и вывел принятое сообщение агент test\_2, после чего отправил сообщение ”Pong” это был ответ на полученное сообщение. Далее агент test\_1 получив сообщение ”Pong”, выводит его.

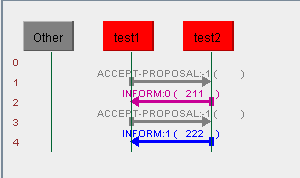
С помощью *DummyAgent* можно проверить работу нашего Агента, отправляя ему запросы. Для запуска *DummyAgent* необходимо нажать кнопку StartDummyAgent в ряду кнопок быстрого доступа пользовательского интерфейса платформы. В самом Агенте необходимо составить запрос в правильной форме, как это показано на рис.19



**Рис. 19. Dummy agent**

После правильного заполнения запроса необходимо нажать на кнопку: отправить  
сообщение. DummyAgent позволяет отправляет запрос с одного точного адреса агента на  
точный адрес другого агента даже если они находится в различных контейнерах. Но эти  
агенты должны быть прописаны в одной мультиагентной системе (т.е. известны DF агентам).

Для того, чтобы проще было отслеживать отправку сообщений, можно запустить еще  
одного агента *Sniffer*, в котором отражаются стрелочками отправленные сообщения между агентами (рис.20).



**Рис. 20. Запуск сниффера**

# Импорт существующего проекта в Eclipse

* + Выберите меню **File->Import...**.
  + В окне **Import** выберите вкладку **General**, а затем **Existing Project into Workspace**.
  + В окне укажите путь к папке проекта и выберите один или несколько обнаруженных проектов.
  + Нажмите Finish.

# Источники

1. Официальный сайт Java Agent Development Framework. URL: <http://jade.tilab.com/>
2. [Скачать Java SE Development Kit (JDK)](http://java.sun.com/javase/downloads/index.jsp) <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
3. Скачать среду Eclipse <http://www.eclipse.org/downloads/>
4. Агентная платформа JADE: <http://www.studfiles.ru/preview/2947243/#2947243>
5. Агенты и мультиагентные системы

<http://www.agentlab.ru/confluence/pages/viewpage.action?pageId=7012409>

# Приложение 1

**Листинг: Файл PingAgent.java**

**package** ru.agentlab;

**import** jade.core.AID;

**import** jade.core.Agent;

**import** jade.core.behaviours.CyclicBehaviour;

**import** jade.domain.AMSService;

**import** jade.domain.FIPAAgentManagement.AMSAgentDescription;

**import** jade.domain.FIPAAgentManagement.SearchConstraints;

**import** jade.lang.acl.ACLMessage;

**public** **class** PingAgent **extends** Agent {

**private** **static** **final** **long** ***serialVersionUID*** = 8257866411543354395L;

**public** **void** setup() {

System.***out***.println("Hello World, my name is : " + getAID().getName());

// Поведение агента исполняемое в цикле

addBehaviour(**new** CyclicBehaviour(**this**) {

**private** **static** **final** **long** ***serialVersionUID*** = 7774831398907094833L;

**public** **void** action() {

ACLMessage msg = receive();

**if** (msg != **null**) {

// Вывод на экран локального имени агента и полученного

// сообщения

System.***out***.println(" – " + myAgent.getLocalName() + " received: " + msg.getContent());

}

// Блокируем поведение, пока в очереди сообщений агента

// не появится хотя бы одно сообщение

block();

}

});

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** (AMSAgentDescription agent : agents) {

AID agentID = agent.getName();

ACLMessage msg = **new** ACLMessage(ACLMessage.***INFORM***);

msg.addReceiver(agentID);// id агента которому отправляем сообщение

msg.setLanguage("English");// Язык сообщения

msg.setContent("Ping"); // Содержимое сообщения

send(msg); // отправляем сообщение

}

}

}

# Приложение 2

**Листинг: Файл PongAgent.java**

**package** ru.agentlab;

**import** jade.core.Agent;

**import** jade.core.behaviours.CyclicBehaviour;

**import** jade.lang.acl.ACLMessage;

**public** **class** PongAgent **extends** Agent {

**private** **static** **final** **long** ***serialVersionUID*** = 3663966406239393054L;

**protected** **void** setup() {

addBehaviour(**new** CyclicBehaviour(**this**) {

**private** **static** **final** **long** ***serialVersionUID*** = -1912882200351395625L;

**public** **void** action() {

ACLMessage msg = receive();

**if** (msg != **null**) {

// Вывод на экран локального имени агента и полученного сообщения

System.***out***.println(" – " + myAgent.getLocalName() + " received: " + msg.getContent());

ACLMessage reply = msg.createReply();

reply.setPerformative(ACLMessage.***INFORM***); // устанавливаем перформатив сообщения

reply.setContent("Pong"); // содержимое сообщения

send(reply);// отправляем сообщения

}

block();

}

});

}

}