Отчёт о лабораторной работе №3

Типы поведения агентов JADE

Выполнила студентка ФКТиПМ

Группа 209

Деева Ирина Юрьевна

Принял к.т.н., доцент Приходько Т.А

г. Краснодар

2016

**Цель работы:**

Изучить виды поведения агентов на примере мультиагентной системы, построенной по принципу pub/sub.

**Результат выполнения работы:**

1. В данной работе рассматривается пример мультиагентного приложения «Торговля книгами», в которой есть агент-продавец, торгующий книгами по определённым ценам, и агент-покупатель, желающий приобрести определённую книгу.

**Листинг класса-продавца**

|  |
| --- |
| package bookTrading;  import jade.core.Agent;  import jade.core.behaviours.\*;  import jade.lang.acl.ACLMessage;  import jade.lang.acl.MessageTemplate;  import jade.domain.DFService;  import jade.domain.FIPAException;  import jade.domain.FIPAAgentManagement.DFAgentDescription;  import jade.domain.FIPAAgentManagement.ServiceDescription;  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.showGui();  // 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();  }  // Add the behaviour serving queries 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() {  // 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.");  }  /\*\*  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));  System.out.println(title+" inserted into catalogue. Price = "+price);  }  } );  }  /\*\*  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() {  MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);  ACLMessage msg = myAgent.receive(mt);  if (msg != null) {  // CFP Message received. Process it  String title = msg.getContent();  System.out.print("Content" + title);  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);  }  else {  block();  }  }  } // End of inner class OfferRequestsServer  /\*\*  Inner class PurchaseOrdersServer.  This is the behaviour used by Book-seller agents to serve incoming  offer acceptances (i.e. purchase orders) from buyer agents.  The seller agent removes the purchased book from its catalogue  and replies with an INFORM message to notify the buyer that the  purchase has been sucesfully completed.  \*/  private class PurchaseOrdersServer extends CyclicBehaviour {  public void action() {  MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.ACCEPT\_PROPOSAL);  ACLMessage msg = myAgent.receive(mt);  if (msg != null) {  // ACCEPT\_PROPOSAL Message received. Process it  String title = msg.getContent();  ACLMessage reply = msg.createReply();  Integer price = (Integer) catalogue.remove(title);  if (price != null) {  reply.setPerformative(ACLMessage.INFORM);  System.out.println(title+" sold to agent "+msg.getSender().getName());  }  else {  // The requested book has been sold to another buyer in the meanwhile .  reply.setPerformative(ACLMessage.FAILURE);  reply.setContent("not-available");  }  myAgent.send(reply);  }  else {  block();  }  }  } // End of inner class OfferRequestsServer  } |

**Листинг класса-покупателя**

|  |
| --- |
| package bookTrading;  import jade.core.Agent;  import jade.core.AID;  import jade.core.behaviours.\*;  import jade.lang.acl.ACLMessage;  import jade.lang.acl.MessageTemplate;  import jade.domain.DFService;  import jade.domain.FIPAException;  import jade.domain.FIPAAgentManagement.DFAgentDescription;  import jade.domain.FIPAAgentManagement.ServiceDescription;  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("Hallo! 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("Target book is "+targetBookTitle);  // Add a TickerBehaviour that schedules a request to seller agents every minute  addBehaviour(new TickerBehaviour(this, 60000) {  protected void onTick() {  System.out.println("Trying to buy "+targetBookTitle);  // 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);  System.out.println("Found the following seller agents:");  sellerAgents = new AID[result.length];  for (int i = 0; i < result.length; ++i) {  sellerAgents[i] = result[i].getName();  System.out.println(sellerAgents[i].getName());  }  }  catch (FIPAException fe) {  fe.printStackTrace();  }  // Perform the request  myAgent.addBehaviour(new RequestPerformer());  }  } );  }  else {  // Make the agent terminate  System.out.println("No target 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.");  }  /\*\*  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.currentTimeMillis());  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 from agent "+reply.getSender().getName());  System.out.println("Price = "+bestPrice);  myAgent.doDelete();  }  else {  System.out.println("Attempt failed: requested book already sold.");  }  step = 4;  }  else {  block();  }  break;  }  }  public boolean done() {  if (step == 2 && bestSeller == null) {  System.out.println("Attempt failed: "+targetBookTitle+" not available for sale");  }  return ((step == 2 && bestSeller == null) || step == 4);  }  } // End of inner class RequestPerformer  } |

**Листинг класса-интерфейса**

|  |
| --- |
| package bookTrading;  import jade.core.AID;  import java.awt.\*;  import java.awt.event.\*;  import javax.swing.\*;  public class BookSellerGui extends JFrame {  private BookSellerAgent myAgent;    private JTextField titleField, priceField;    BookSellerGui(BookSellerAgent a) {  super(a.getLocalName());    myAgent = a;    JPanel p = new JPanel();  p.setLayout(new GridLayout(2, 2));  p.add(new JLabel("Book title:"));  titleField = new JTextField(15);  p.add(titleField);  p.add(new JLabel("Price:"));  priceField = new JTextField(15);  p.add(priceField);  getContentPane().add(p, BorderLayout.CENTER);    JButton addButton = new JButton("Add");  addButton.addActionListener( new ActionListener() {  public void actionPerformed(ActionEvent ev) {  try {  String title = titleField.getText().trim();  String price = priceField.getText().trim();  myAgent.updateCatalogue(title, Integer.parseInt(price));  titleField.setText("");  priceField.setText("");  }  catch (Exception e) {  JOptionPane.showMessageDialog(BookSellerGui.this, "Invalid values. "+e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);  }  }  } );  p = new JPanel();  p.add(addButton);  getContentPane().add(p, BorderLayout.SOUTH);    // Make the agent terminate when the user closes  // the GUI using the button on the upper right corner  addWindowListener(new WindowAdapter() {  public void windowClosing(WindowEvent e) {  myAgent.doDelete();  }  } );    setResizable(false);  }    public void showGui() {  pack();  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();  int centerX = (int)screenSize.getWidth() / 2;  int centerY = (int)screenSize.getHeight() / 2;  setLocation(centerX - getWidth() / 2, centerY - getHeight() / 2);  super.setVisible(true);  }  } |

1. При запуске программы и добавлении агента-продавца инициализируется каталог книг, куда можно заносить книги и их цены.

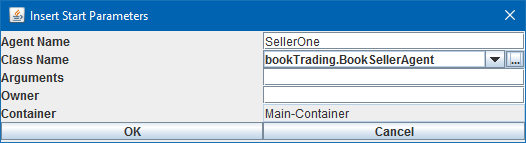


Рисунок 1 – Создание агента-продавца

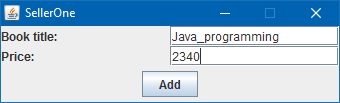


Рисунок 2 – заполнение каталога книг

1. Затем при создании агента-покупателя мы в аргументах указываем книгу, которую хотим купить. Программа сама найдёт подходящего агента-продавца, предлагающего желаемую книгу по минимальной цене.

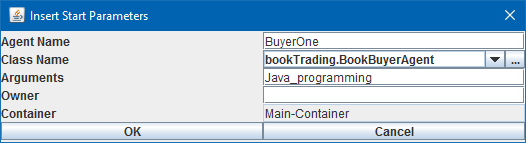


Рисунок 3 – Создание агента-покупателя

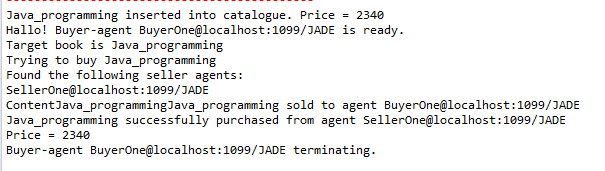


Рисунок 4 – результат работы программы

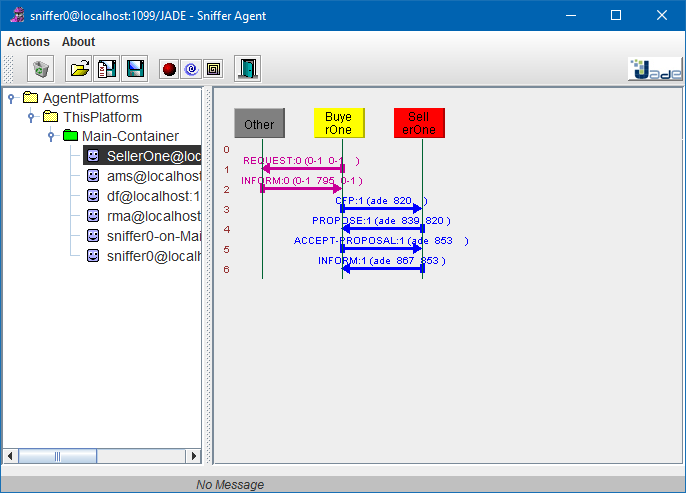


Рисунок 5 – Общение агентов в сниффере