Описание модели агента

Торговли автомобилями

Разработала студентка группы 209 Деева Ирина Юрьевна

Принял к.т.н., доцент Приходько Т.А

Краснодар, 2016

Содержание

[1 Общие сведения 3](#_Toc465080006)

[1.1 Наименование модели агента 3](#_Toc465080007)

[1.2 Текстовое описание модели 3](#_Toc465080008)

[1.3 Анализ типа агента 3](#_Toc465080009)

[2 Описание величин 4](#_Toc465080010)

[2.1 Входные величины 4](#_Toc465080011)

[2.2 Выходные данные 4](#_Toc465080012)

[2.3 Внутренние величины 4](#_Toc465080013)

[3 Описание внутренних процессов 4](#_Toc465080014)

[4 Описание взаимодействия агента с другими агентами 4](#_Toc465080015)

[5 Схема взаимодействия агентов 6](#_Toc465080016)

[6 Коды агентов 6](#_Toc465080017)

[6.1 Класс продавца CarSellerAgent 6](#_Toc465080018)

[6.2 Класс покупателя CarBuyerAgent 9](#_Toc465080019)

[6.3 Класс интерфейса добавления автомобиля в каталог CarSellerGui 13](#_Toc465080020)

# 1 Общие сведения

# Наименование модели агента

Агент автомобильного дилера. Агент покупателя автомобилей.

# Текстовое описание модели

Агент продавца автомобилей предлагает автомобили с заданным километражем и ценой. Агент покупателя желает купить автомобиль, километраж которого не выше заданного, а цена не превышает желаемую.

Примеры требований покупателей и предложений продавцов:

**Требования покупателей**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Требования** | **Покупатели** | | | | |
|  | 1 | 2 | 3 | 4 | 5 |
| **Пробег автомобиля** | 800 | 900 | 1000 | 700 | 1500 |
| **Цена, которую согласен** **заплатить покупатель, д.е.** | 5000 | 4500 | 4200 | 4900 | 5200 |

**Предложения продавцов**

|  |  |  |
| --- | --- | --- |
| **Автомобили** | **Действительный пробег** | **Цена продавца, д.е.** |
| **1** | 1000 | 4900 |
| **2** | 650 | 5000 |
| **3** | 1000 | 4750 |
| **4** | 900 | 4600 |
| **5** | 4000 | **5100** |

# Анализ типа агента

Разработанная мультиагентная система представляет собой смешанный тип, а именно, она выполняет как функции агента сбора данных, то есть запрашивает некоторые данные или услуги от имени пользователя, так и агента поддержки принятия решений, так как интерпретирует и анализирует собранные данные и принимает решение о совершении сделки.

# Описание величин

# Входные величины

Для агента дилера:

- Километраж автомобиля;

- Предлагаемая цена автомобиля.

Для агента покупателя:

- Желаемый километраж автомобиля;

- Желаемая цена автомобиля.

# 2.2 Выходные данные

Выходные величины мультиагентной системы:

- Список активных агентов;

- Положительный ответ о наличии требуемого автомобиля;

- Отрицательный ответ о наличии требуемого автомобиля;

- Сообщение о совершённой сделке;

- Сообщение о невозможности совершения сделки.

# 2.3 Внутренние величины

Внутренние величины системы:

- Каталог автомобилей;

- Текущий проверяемый автомобиль в каталоге.

# Описание внутренних процессов

Агент покупателя посылает запрос на поиск активных агентов-дилеров, которые могут предложить автомобиль, цена и километраж которого не больше тех, что указал покупатель. Поиск подходящих автомобилей и агентов ведётся до тех пор, пока не будет найден хотя бы один подходящий автомобиль. Если подходят несколько автомобилей, выбирается автомобиль с наименьшей ценой. После нахождения автомобиля выдаётся сообщение о совершении сделки и автомобиль удаляется из каталога.

# Описание взаимодействия агента с другими агентами

В процессе работы мультиагентной системы взаимодействуют два основных агента:

- Агент продавца автомобилей;

- Агент покупателя автомобилей.

При добавлении агента продавца формируется каталог автомобилей, где указываются автомобили с их километражем и ценой (рисунок 1).

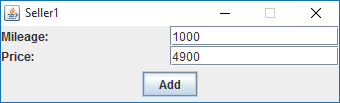


Рисунок 1 – Каталог

Затем, как только добавляется агент покупателя, начинается поиск активных агентов продавцов (рисунок 2).

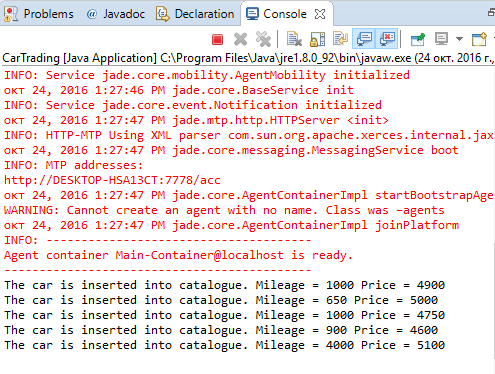


Рисунок 2 – каталоги продавцов

В их каталогах перебираются все возможные автомобили и выбираются те, цена и километраж которых не больше желаемых покупателем. Если подходящий автомобиль найден, посылается сообщение покупателю о совершении сделки и автомобиль удаляется из каталога (рисунок 3).

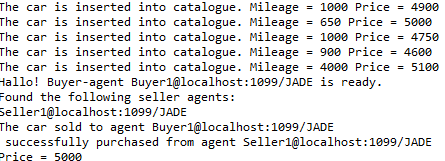
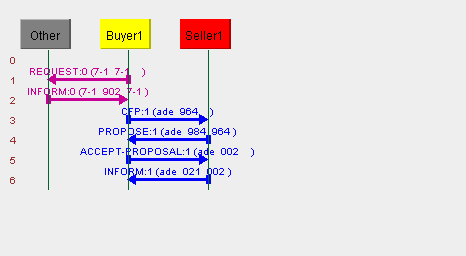


Рисунок 3 – Сообщение о совершённой сделке

# Схема взаимодействия агентов



# Коды агентов

# Класс продавца CarSellerAgent

package CarTrading;

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 CarSellerAgent extends Agent{

// The catalogue of cars for sale (maps the mileage of a car to its price)

private Hashtable catalogue;

// The GUI by means of which the user can add cars in the catalogue

private CarSellerGui myGui;

// Put agent initializations here

protected void setup() {

// Create the catalogue

catalogue = new Hashtable();

// Create and show the GUI

myGui = new CarSellerGui(this);

myGui.showGui();

// Register the car-selling service in the yellow pages

DFAgentDescription dfd = new DFAgentDescription();

dfd.setName(getAID());

ServiceDescription sd = new ServiceDescription();

sd.setType("car-selling");

sd.setName("JADE-car-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 car for sale

\*/

public void updateCatalogue(final String mileage, final int price) {

addBehaviour(new OneShotBehaviour() {

public void action() {

catalogue.put(mileage, new Integer(price));

System.out.println("The car is inserted into catalogue. Mileage = "+mileage+" Price = " + price);

}

} );

}

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 message = msg.getContent();

String[] arguments = message.split("\_");

Integer price = Integer.parseInt(arguments[1]);

ACLMessage reply = msg.createReply();

Enumeration meliages = catalogue.keys();

for(int i = 0; i < catalogue.size(); i++)

{

String meliageC = (String)meliages.nextElement();

if(Integer.parseInt(meliageC) <= Integer.parseInt(arguments[0]))

{

Integer priceC = (Integer) catalogue.get(meliageC);

if (priceC != null && priceC <= price) {

// The requested car is available for sale. Reply with the price

reply.setPerformative(ACLMessage.PROPOSE);

reply.setContent(String.valueOf(priceC.intValue()));

}

else {

// The requested car is NOT available for sale.

reply.setPerformative(ACLMessage.REFUSE);

reply.setContent("not-available");

}

myAgent.send(reply);

}

else

continue;

}

}

else {

block();

}

}

} // End of inner class OfferRequestsServer

private class PurchaseOrdersServer extends CyclicBehaviour {

public void action() {

MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.ACCEPT\_PROPOSAL);

ACLMessage msg = myAgent.receive(mt);

if (msg != null) {

String message = msg.getContent();

String[] arguments = message.split("\_");

Integer price = Integer.parseInt(arguments[1]);

ACLMessage reply = msg.createReply();

Enumeration meliages = catalogue.keys();

// ACCEPT\_PROPOSAL Message received. Process it

for(int i = 0; i < catalogue.size(); i++)

{

String meliageC = (String)meliages.nextElement();

if(Integer.parseInt(meliageC) <= Integer.parseInt(arguments[0]))

{

Integer priceC = (Integer) catalogue.remove(meliageC);

if (priceC != null && priceC <= price) {

// The requested car is available for sale. Reply with the price

reply.setPerformative(ACLMessage.INFORM);

System.out.println("The car sold to agent "+msg.getSender().getName());

}

else {

// The requested car is NOT available for sale.

reply.setPerformative(ACLMessage.FAILURE);

reply.setContent("not-available");

}

myAgent.send(reply);

}

else

continue;

}

}

else {

block();

}

}

} // End of inner class OfferRequestsServer

}

# Класс покупателя CarBuyerAgent

package CarTrading;

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 CarBuyerAgent extends Agent{

private String targetMeliage;

private String message;

// 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.");

Object[] args = getArguments();

if (args != null && args.length > 0) {

message = (String)args[0];

targetMeliage = message.split("\_")[0];

// Add a TickerBehaviour that schedules a request to seller agents every minute

addBehaviour(new TickerBehaviour(this, 30000) {

protected void onTick() {

// Update the list of seller agents

DFAgentDescription template = new DFAgentDescription();

ServiceDescription sd = new ServiceDescription();

sd.setType("car-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 car title specified");

doDelete();

}

}

// Put agent clean-up operations here

protected void takeDown() {

// Printout a dismissal message

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

}

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(message);

cfp.setConversationId("car-trade");

cfp.setReplyWith("cfp"+System.currentTimeMillis()); // Unique value

myAgent.send(cfp);

// Prepare the template to get proposals

mt = MessageTemplate.and(MessageTemplate.MatchConversationId("car-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(message);

order.setConversationId("car-trade");

order.setReplyWith("order"+System.currentTimeMillis());

myAgent.send(order);

// Prepare the template to get the purchase order reply

mt = MessageTemplate.and(MessageTemplate.MatchConversationId("car-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(" successfully purchased from agent "+reply.getSender().getName());

System.out.println("Price = "+bestPrice);

myAgent.doDelete();

}

else {

System.out.println("Attempt failed: requested car already sold.");

}

step = 4;

}

else {

block();

}

break;

}

}

public boolean done() {

if (step == 2 && bestSeller == null) {

System.out.println("Attempt failed: "+" not available for sale");

}

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

}

} // End of inner class RequestPerformer

}

# Класс интерфейса добавления автомобиля в каталог CarSellerGui

package CarTrading;

import jade.core.AID;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class CarSellerGui extends JFrame{

private CarSellerAgent myAgent;

private JTextField mileageField, priceField;

CarSellerGui(CarSellerAgent a) {

super(a.getLocalName());

myAgent = a;

JPanel p = new JPanel();

p.setLayout(new GridLayout(2, 2));

p.add(new JLabel("Mileage:"));

mileageField = new JTextField(15);

p.add(mileageField);

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 mileage = mileageField.getText().trim();

String price = priceField.getText().trim();

myAgent.updateCatalogue(mileage,Integer.parseInt(price));

mileageField.setText("");

priceField.setText("");

}

catch (Exception e) {

JOptionPane.showMessageDialog(CarSellerGui.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);

}

}