МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОПИСАНИЕ МУЛЬТИАГЕНТНОЙ МОДЕЛИ**

**МОДЕЛЬ “ТАКСИ”**

Выполнил студент группы 209/2                                    Д. В. Какаулин

Направление подготовки 01.04.02 Прикладная математика и информатика

Магистерская программа: Технологии программирования и разработки информационно-коммуникационных систем

Отчет приняла

к.т.н., доцент                                                                                     Т.А. Приходько

Краснодар

2024 г.

**СОДЕРЖАНИЕ**

[1 Общие сведения 3](#_Toc166769627)

[1.1 Наименования модели агентов 3](#_Toc166769628)

[1.2 Текстовое описание модели 3](#_Toc166769629)

[1.3 Анализ типов агентов 3](#_Toc166769630)

[2 Описание внутренних процессов 4](#_Toc166769631)

[2.1 Агент такси 4](#_Toc166769632)

[2.2 Агент клиент 4](#_Toc166769633)

[2.3 Агент координатор 4](#_Toc166769634)

[3 Описание взаимодействия агентов друг с другом 5](#_Toc166769635)

[4 Характеристики среды вокруг агентов 5](#_Toc166769636)

[5 Схема взаимодействия агентов 6](#_Toc166769637)

[6 Код агентов 6](#_Toc166769638)

[6.1 ClientAgent 6](#_Toc166769639)

[6.2 TaxiAgent 7](#_Toc166769640)

[6.3 CoordinatorAgent 8](#_Toc166769641)

# Общие сведения

## Наименования модели агентов

Полное наименование агентурной модели: динамическая система подбора машин такси для клиентов.

## Текстовое описание модели

Агенты такси создаются и добавляют себя в систему DF, клиенты делают запрос на вызов такси, координатор получает запрос, находит ближайшее не занятой такси, передает необходимые данные клиенту и такси. В случае отсутствия доступных такси, координатор ставит клиента в очередь.

## Анализ типов агентов

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование агента** | **Входные величины** | **Внутренние величины** | **Выходные величины** |
| Такси | * Координаты клиента; | * Текущие координаты такси; * Локальное имя агента; | * Отправка координатору информации о своем местоположении на запрос в случае незанятости агента; * Отправка координатору информации о завершении поездки в случае принятия подтверждения о выдаче заказа от координатора; |
| Клиент | * Локальное имя агента такси, назначенного на выполнение заказа координатором; | * Координаты клиента; * Локальное имя агента клиента | * Отправка запроса координатору на вызов такси с передачей своих координат; |
| Координатор | * Текущие координаты такси; * Координаты, указанные агентом-клиентом. | * Очередь клиентов; * Класс данных, содержащий поля AID клиента и его координат; | * Запрос всем агентам такси из DF на получение их текущих координат; * Подтверждение заказа ближайшему к указанным клиентом координатам доступному агенту такси; * Информирование клиента о подтверждении вызова такси; |

# Описание внутренних процессов

## Агент такси

Осуществляет цель выполнения услуги перевозки клиента в рамках общения с координатором:

* Внешняя цель – выполнить заказ, выданный координатором;
* Внутренняя цель – довести клиента до полученных координат и проинформировать координатора;

## Агент клиент

Осуществляет цель запроса услуги перевозки:

* Внешняя цель – запросить координатора подобрать ему такси;
* Внутренняя цель – заполнение вводных координат, ожидание сообщения от координатора о подтверждении поездки, удаление себя;

## Агент координатор

* Внешняя цель – получать запросы от клиентов, подбирать такси, информировать обоих агентов;
* Внутренняя цель – расчет расстояний между агентами клиента и такси, нахождение ближайшего к координатам клиента агента такси;

# Описание взаимодействия агентов друг с другом

Агент клиента взаимодействует с:

* Агентом координатором;

Агент координатор взаимодействует с:

* Агентом клиентом;
* Агентом такси;

Агент такси взаимодействует с:

* Агентом координатором;

Агенты такси вносят себя в DF в момент своей инициализации, генерируют свои координаты. Агент клиент в момент своей инициализации запускает OneShotBehaviour, в котором отправляет запрос агенту координатору на вызов такси. Агент координатор, получив запрос, добавляет его в очередь клиентов. Далее агент координатор берет пер вого клиента в очереди по принципу FIFO, опрашивает всех агентов такси через DF. Агенты такси отвечают при условии, что они свободны. Если в данный момент отсутствуют доступные агенты такси, координатор ничего не делает и уведомляет об этом в консоль, продолжая свою попытку обработать клиента в очереди спустя секунду времени. Если доступные агенты такси имеются, агент координатор итерирует по полученному списку, находя ближайшего к указанным клиентом координатам агента такси. Далее агент координатор подтверждает заказ выбранному агенту такси и информирует об этом клиента. Клиент удаляет себя по получении сообщения об успехе от координатора. Агент такси меняет свой статус на “занятый”, ожидает некоторое время и по окончании поездки возвращает статус “свободный”, информируя при этом координатора.

# Характеристики среды вокруг агентов

Базовая среды Jade без сторонних модификаций.

# Схема взаимодействия агентов

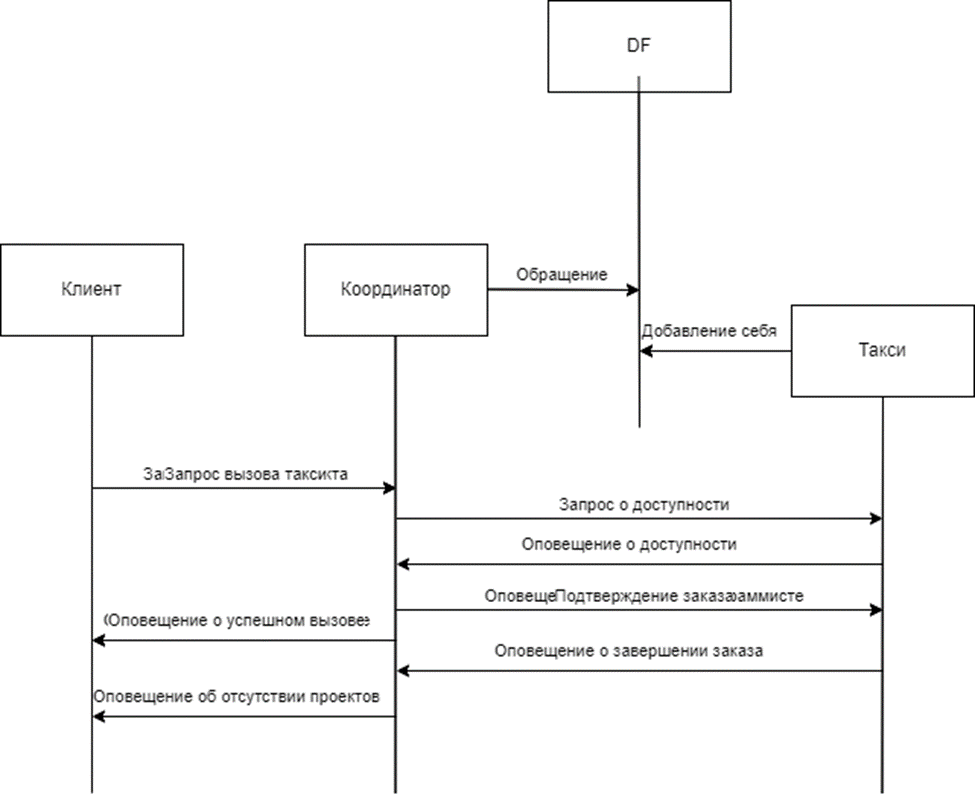


Рисунок 1 – Схема мультиагентной модели “Такси”

# Код агентов

## ClientAgent

1. public class ClientAgent extends Agent {
3. private String location;
4. private String localName;
5. @Override
6. protected void setup() {
7. Object[] args = getArguments();
8. if (args != null && args.length > 0) {
9. StringBuilder sb = new StringBuilder();
10. for (int i = 0; i < args.length; i++) {
11. if (i > 0) sb.append(",");
12. sb.append(args[i]);
13. }
14. location = sb.toString();
15. } else {
16. location = "0,0";
17. }
18. localName = getLocalName();
19. addBehaviour(new RequestTaxiBehaviour());
20. }
21. private class RequestTaxiBehaviour extends OneShotBehaviour {
22. @Override
23. public void action() {
24. ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
25. msg.addReceiver(new AID("coordinator", AID.ISLOCALNAME));
26. msg.setContent(location);
27. send(msg);
28. MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
29. ACLMessage reply = blockingReceive(mt);
30. if (reply != null) {
31. String taxiID = reply.getContent();
32. System.out.println("(Client) " + localName + ": Taxi " + taxiID + " will pick me up at " + location);
33. doDelete();
34. }
35. }
36. }
38. }

## TaxiAgent

1. public class TaxiAgent extends Agent {
3. private boolean busy = false;
4. private String location;
5. private String localName;
6. @Override
7. protected void setup() {
8. Random rand = new Random();
9. location = rand.nextInt(10) + "," + rand.nextInt(10);
10. DFAgentDescription dfd = new DFAgentDescription();
11. dfd.setName(getAID());
12. ServiceDescription sd = new ServiceDescription();
13. sd.setType("taxi");
14. sd.setName(getLocalName());
15. localName = getLocalName();
16. dfd.addServices(sd);
17. try {
18. DFService.register(this, dfd);
19. } catch (FIPAException fe) {
20. fe.printStackTrace();
21. }
22. System.out.println("(Taxi)" + localName + ": Is ready, start location: " + location);
23. addBehaviour(new RespondToCoordinatorBehaviour());
24. }
25. @Override
26. protected void takeDown() {
27. try {
28. DFService.deregister(this);
29. } catch (FIPAException fe) {
30. fe.printStackTrace();
31. }
32. }
33. private class RespondToCoordinatorBehaviour extends CyclicBehaviour {
34. @Override
35. public void action() {
36. ACLMessage msg = receive();
37. if (msg != null) {
38. if (msg.getPerformative() == ACLMessage.REQUEST && !busy) {
39. ACLMessage reply = msg.createReply();
40. reply.setPerformative(ACLMessage.INFORM);
41. reply.setContent(location);
42. send(reply);
43. } else if (msg.getPerformative() == ACLMessage.CONFIRM) {
44. busy = true;
45. String clientLocation = msg.getContent();
46. System.out.println("(Taxi)" + localName + ": Picking up client at " + clientLocation);
47. doWait(5000);
48. Random rand = new Random();
49. location = rand.nextInt(10) + "," + rand.nextInt(10);
50. busy = false;
51. System.out.println("(Taxi)" + localName + ": Finished the drive and now is free, current location: " + location);
52. ACLMessage done = new ACLMessage(ACLMessage.INFORM);
53. done.addReceiver(new AID("coordinator", AID.ISLOCALNAME));
54. done.setContent("done");
55. send(done);
56. }
57. } else {
58. block();
59. }
60. }
61. }
63. }

## CoordinatorAgent

1. public class CoordinatorAgent extends Agent {
3. private Queue<ClientRequest> clientQueue = new LinkedList<>();
4. @Override
5. protected void setup() {
6. addBehaviour(new HandleRequestsBehaviour());
7. addBehaviour(new CheckQueueBehaviour());
8. }
9. private class HandleRequestsBehaviour extends CyclicBehaviour {
10. @Override
11. public void action() {
12. ACLMessage msg = receive();
13. if (msg != null && msg.getPerformative() == ACLMessage.REQUEST) {
14. String clientLocation = msg.getContent();
15. AID client = msg.getSender();
16. clientQueue.add(new ClientRequest(client, clientLocation));
17. } else {
18. block();
19. }
20. }
21. }
22. private class CheckQueueBehaviour extends CyclicBehaviour {
23. @Override
24. public void action() {
25. if (!clientQueue.isEmpty()) {
26. ClientRequest clientRequest = clientQueue.peek();
27. DFAgentDescription template = new DFAgentDescription();
28. ServiceDescription sd = new ServiceDescription();
29. sd.setType("taxi");
30. template.addServices(sd);
31. try {
32. DFAgentDescription[] result = DFService.search(myAgent, template);
33. List<AID> taxiAgents = new ArrayList<>();
34. for (DFAgentDescription dfd : result) {
35. taxiAgents.add(dfd.getName());
36. }
37. if (!taxiAgents.isEmpty()) {
38. ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
39. for (AID taxi : taxiAgents) {
40. request.addReceiver(taxi);
41. }
42. send(request);
43. Map<AID, Double> taxiDistances = new HashMap<>();
44. for (int i = 0; i < taxiAgents.size(); i++) {
45. MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
46. ACLMessage reply = blockingReceive(mt, 1000); // Ждем ответ в течение 1 секунды
47. if (reply != null) {
48. String taxiLocation = reply.getContent();
49. AID taxi = reply.getSender();
50. double distance = calculateDistance(clientRequest.location, taxiLocation);
51. taxiDistances.put(taxi, distance);
52. }
53. }
54. AID closestTaxi = null;
55. double minDistance = Double.MAX\_VALUE;
56. for (Map.Entry<AID, Double> entry : taxiDistances.entrySet()) {
57. if (entry.getValue() < minDistance) {
58. minDistance = entry.getValue();
59. closestTaxi = entry.getKey();
60. }
61. }
62. if (closestTaxi != null) {
63. ACLMessage confirm = new ACLMessage(ACLMessage.CONFIRM);
64. confirm.addReceiver(closestTaxi);
65. confirm.setContent(clientRequest.location);
66. send(confirm);
67. ACLMessage informClient = new ACLMessage(ACLMessage.INFORM);
68. informClient.addReceiver(clientRequest.client);
69. informClient.setContent(closestTaxi.getLocalName());
70. send(informClient);
71. // System.out.println("Coordinator: Assigned " + closestTaxi.getLocalName() + " to client at " + clientRequest.location);
72. System.out.println("Coordinator: Assigned " + closestTaxi.getLocalName() + " to " + clientRequest.client.getLocalName() + " at " + clientRequest.location);
73. clientQueue.poll();
74. }
75. }
76. else
77. {
78. System.out.println("Coordinator: No available taxi to pick up " + clientRequest.client.getLocalName() + " at the moment");
79. doWait(1000);
80. }
81. } catch (FIPAException fe) {
82. fe.printStackTrace();
83. }
84. } else {
85. block(1000);
86. }
87. }
88. private double calculateDistance(String loc1, String loc2) {
89. try {
90. String[] coords1 = loc1.split(",");
91. String[] coords2 = loc2.split(",");
92. int x1 = Integer.parseInt(coords1[0]);
93. int y1 = Integer.parseInt(coords1[1]);
94. int x2 = Integer.parseInt(coords2[0]);
95. int y2 = Integer.parseInt(coords2[1]);
96. return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
97. } catch (NumberFormatException | ArrayIndexOutOfBoundsException e) {
98. System.err.println("Error parsing coordinates: " + loc1 + ", " + loc2);
99. e.printStackTrace();
100. return Double.MAX\_VALUE;
101. }
102. }
103. }
104. private static class ClientRequest {
105. AID client;
106. String location;
107. ClientRequest(AID client, String location) {
108. this.client = client;
109. this.location = location;
110. }
111. }
113. }