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 РАСПРЕДЕЛЕНИЯ ПРОЕКТОВ**

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](#_Toc166770187)

[1.1 Наименования модели агентов 3](#_Toc166770188)

[1.2 Текстовое описание модели 3](#_Toc166770189)

[1.3 Анализ типов агентов 3](#_Toc166770190)

[2 Описание внутренних процессов 4](#_Toc166770191)

[2.1 Агент проект 4](#_Toc166770192)

[2.2 Агент программист 4](#_Toc166770193)

[2.3 Агент координатор 5](#_Toc166770194)

[3 Описание взаимодействия агентов друг с другом 5](#_Toc166770195)

[4 Характеристики среды вокруг агентов 6](#_Toc166770196)

[5 Схема взаимодействия агентов 6](#_Toc166770197)

[6 Код агентов 6](#_Toc166770198)

[6.1 ProgrammerAgent 6](#_Toc166770199)

[6.2 TaxiAgent 8](#_Toc166770200)

[6.3 CoordinatorAgent 9](#_Toc166770201)

# Общие сведения

## Наименования модели агентов

Полное наименование мультиагентной модели: динамическая система подбора проектов для программистов, удовлетворяющих требованию по наличию опыта, с учётом наибольшей оплаты.

## Текстовое описание модели

Агенты проектов создаются и добавляют себя в систему DF, программисты каждую секунду делают запрос координатору на получение проекта. Координатор каждые 5 секунд обновляет свой список доступных проектов. При получении запроса от программиста координатор ищет подходящий по опыту проект с наибольшей оплатой. В случае отсутствия такового, координатор отвечает отказом. В случае наличия, координатор информирует агента программиста и агента проекта. Агент программист удаляет в себя, агент проект добавляет программиста в свой список программистов и, в случае достижения необходимого числа, удаляет себя.

## Анализ типов агентов

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование агента** | **Входные величины** | **Внутренние величины** | **Выходные величины** |
| Проект | * Информация о программисте; | * Список программистов; * Требуемое число программистов; * Требуемый минимальный опыт |  |
| Программист | * Информация о подобранном проекте; | * Опыт; | * Отправка запроса координатору на вызов подбор проекта с передачей значения своего опыта; |
| Координатор | * Информация о программисте (опыт); * Информация о проекте (требуемый опыт, оплата); | * Список текущих открытых проектов; | * 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 проект

Осуществляет цель заполнения проекта программистами:

* Внешняя цель – записать себя в DF с уточнением требуемого опыта и обещанного оклада;
* Внутренняя цель – добавлять программистов в проект до достижения требуемого значение, удаление себя при удовлетворении данного значения;

## Агент программист

Осуществляет цель запроса услуги перевозки:

* Внешняя цель – запросить координатора подобрать ему проект;
* Внутренняя цель – заполнение вводных данных опыта, ожидание сообщения от координатора о подтверждении назначения на проект, удаление себя;

## Агент координатор

* Внешняя цель – получать запросы от программистов, подбирать проект, информировать обоих агентов;
* Внутренняя цель – периодическое обновление своего внутреннего списка проектов информацией из DF, нахождение подходящего программисту проекта с максимальным окладом;

# Описание взаимодействия агентов друг с другом

Агент программиста взаимодействует с:

* Агентом координатор;

Агент координатор взаимодействует с:

* Агентом программист;
* Агентом проект;

Агент такси взаимодействует с:

* Агентом координатор;

Агенты проектов вносят себя в DF в момент своей инициализации, в описание они добавляют информацию о требуемом опыте работы для программистов и об окладе. Координатор периодически сверяет свой список проектов с проектами, находящимися в DF. Агент программист периодически делает запрос координатору на получение проекта. Агент координатор подбирает доступный проект с наибольшим окладом в соответствии с требованиями по опыту работы. В случае отсутствия такового, он информирует об этом программиста. В случае нахождения подходящего проекта, он информирует программиста о его наличии и проект о наличии подходящего программиста. Агент программист удаляет себя при получении положительного оповещения. Агент проект добавляет программиста в свой список имеющихся у него программистов и, в случае полного его заполнения, удаляет себя.

# Характеристики среды вокруг агентов

Свойства среды:

* Полностью обозримая;
* Детерминированная;
* Поседовательная;
* Статическая;
* Дискретная;
* Многоагентная;

# Схема взаимодействия агентов

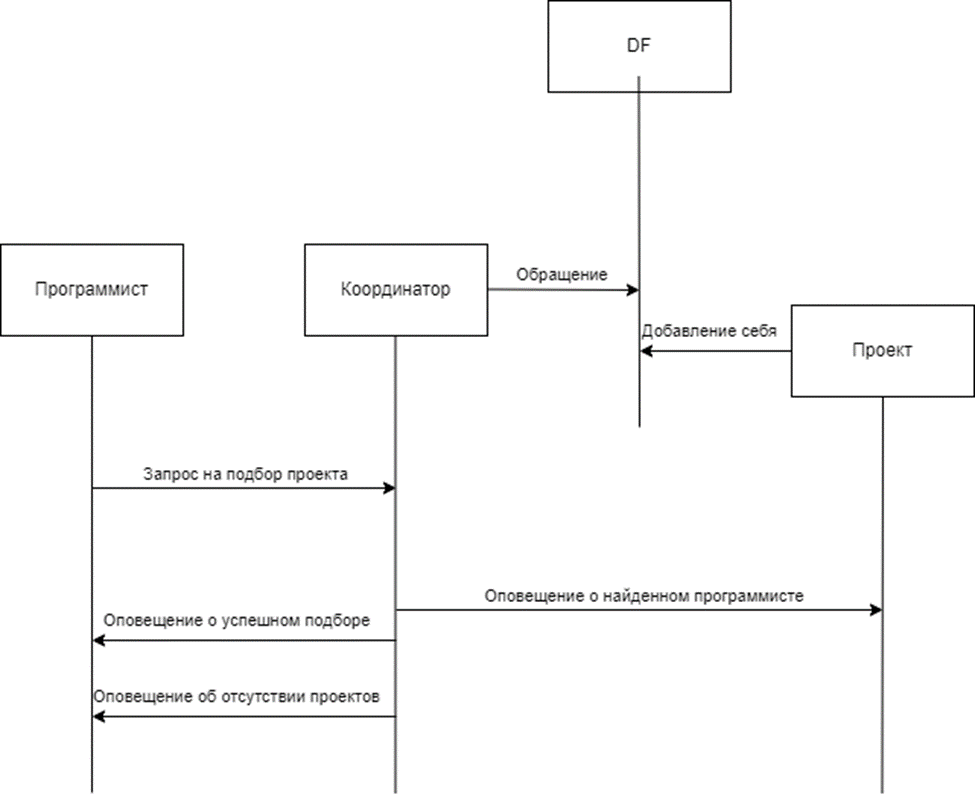


Рисунок 1 – Схема мультиагентной модели “Подбор проекта”

# Код агентов

## ProgrammerAgent

1. public class ProgrammerAgent extends Agent {
2. private int experience;
3. @Override
4. protected void setup() {
5. Object[] args = getArguments();
6. if (args != null && args.length > 0) {
7. experience = Integer.parseInt(args[0].toString());
8. } else {
9. experience = 0; // Default experience
10. }
11. DFAgentDescription dfd = new DFAgentDescription();
12. dfd.setName(getAID());
13. ServiceDescription sd = new ServiceDescription();
14. sd.setType("programmer");
15. sd.setName(getLocalName());
16. dfd.addServices(sd);
17. try {
18. DFService.register(this, dfd);
19. } catch (FIPAException fe) {
20. fe.printStackTrace();
21. }
22. addBehaviour(new ProjectSearchBehaviour());
23. }
24. @Override
25. protected void takeDown() {
26. try {
27. DFService.deregister(this);
28. } catch (FIPAException fe) {
29. fe.printStackTrace();
30. }
31. }
32. private class ProjectSearchBehaviour extends CyclicBehaviour {
33. @Override
34. public void action() {
35. ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
36. request.addReceiver(new AID("coordinator", AID.ISLOCALNAME));
37. request.setContent("Looking for projects with experience: " + experience);
38. send(request);
39. ACLMessage reply = blockingReceive();
40. if (reply != null) {
41. if (reply.getPerformative() == ACLMessage.INFORM) {
42. String projectDetails = reply.getContent();
43. if (projectDetails.startsWith("Project:")) {
44. System.out.println("(Prog)" + getLocalName() + " received project: " + projectDetails);
45. doDelete();
46. } else {
47. System.out.println("(Prog)" + getLocalName() + " received: " + projectDetails);
48. doWait(3000);
49. }
50. }
51. }
52. }
53. }
54. }

## ProjectAgent

1. public class ProjectAgent extends Agent {
2. private int cost;
3. private int requiredProgrammers;
4. private int minimumExperience;
5. private List<AID> programmers = new ArrayList<>();
6. @Override
7. protected void setup() {
8. Object[] args = getArguments();
9. if (args != null && args.length >= 3) {
10. cost = Integer.parseInt(args[0].toString());
11. requiredProgrammers = Integer.parseInt(args[1].toString());
12. minimumExperience = Integer.parseInt(args[2].toString());
13. }
14. DFAgentDescription dfd = new DFAgentDescription();
15. dfd.setName(getAID());
16. ServiceDescription sd = new ServiceDescription();
17. sd.setType("project");
18. sd.setName(cost + "," + requiredProgrammers + "," + minimumExperience);
19. dfd.addServices(sd);
20. try {
21. DFService.register(this, dfd);
22. } catch (FIPAException fe) {
23. fe.printStackTrace();
24. }
25. addBehaviour(new AddProgrammerBehaviour());
26. }
27. @Override
28. protected void takeDown() {
29. try {
30. DFService.deregister(this);
31. } catch (FIPAException fe) {
32. fe.printStackTrace();
33. }
34. }
35. private class AddProgrammerBehaviour extends CyclicBehaviour {
36. @Override
37. public void action() {
38. ACLMessage msg = receive();
39. if (msg != null && msg.getPerformative() == ACLMessage.INFORM) {
40. AID programmer = new AID(msg.getContent(), AID.ISGUID);
41. programmers.add(programmer);
42. System.out.println("Project: " + getLocalName() + " added programmer: " + programmer.getLocalName());
43. if (programmers.size() >= requiredProgrammers) {
44. System.out.println("Project: " + getLocalName() + " is fully staffed and will now terminate.");
45. doDelete();
46. }
47. } else {
48. block();
49. }
50. }
51. }
52. }

## CoordinatorAgent

1. public class CoordinatorAgent extends Agent {
2. private List<Project> projects = new ArrayList<>();
3. @Override
4. protected void setup() {
5. DFAgentDescription dfd = new DFAgentDescription();
6. dfd.setName(getAID());
7. ServiceDescription sd = new ServiceDescription();
8. sd.setType("coordinator");
9. sd.setName(getLocalName());
10. dfd.addServices(sd);
11. try {
12. DFService.register(this, dfd);
13. } catch (FIPAException fe) {
14. fe.printStackTrace();
15. }
16. addBehaviour(new FindProjectsBehaviour());
17. addBehaviour(new HandleRequestsBehaviour());
18. }
19. @Override
20. protected void takeDown() {
21. try {
22. DFService.deregister(this);
23. } catch (FIPAException fe) {
24. fe.printStackTrace();
25. }
26. }
27. private class FindProjectsBehaviour extends CyclicBehaviour {
28. @Override
29. public void action() {
30. projects.clear();
31. DFAgentDescription template = new DFAgentDescription();
32. ServiceDescription sd = new ServiceDescription();
33. sd.setType("project");
34. template.addServices(sd);
35. try {
36. DFAgentDescription[] result = DFService.search(myAgent, template);
37. for (DFAgentDescription dfd : result) {
38. AID projectAID = dfd.getName();
39. Iterator<ServiceDescription> services = dfd.getAllServices();
40. while (services.hasNext()) {
41. ServiceDescription service = services.next();
42. String[] projectDetails = service.getName().split(",");
43. int cost = Integer.parseInt(projectDetails[0].trim());
44. int requiredProgrammers = Integer.parseInt(projectDetails[1].trim());
45. int minimumExperience = Integer.parseInt(projectDetails[2].trim());
46. projects.add(new Project(projectAID, cost, requiredProgrammers, minimumExperience));
47. }
48. }
49. } catch (FIPAException fe) {
50. fe.printStackTrace();
51. }
52. block(5000);
53. }
54. }
55. private class HandleRequestsBehaviour extends CyclicBehaviour {
56. @Override
57. public void action() {
58. ACLMessage msg = receive();
59. if (msg != null && msg.getPerformative() == ACLMessage.REQUEST) {
60. String content = msg.getContent();
61. String[] parts = content.split(":");
62. if (parts.length == 2 && parts[0].trim().equals("Looking for projects with experience")) {
63. int experience = Integer.parseInt(parts[1].trim());
64. List<Project> suitableProjects = new ArrayList<>();
65. for (Project project : projects) {
66. if (project.minimumExperience <= experience) {
67. suitableProjects.add(project);
68. }
69. }
70. Project bestProject = null;
71. int maxCost = 0;
72. for (Project project : suitableProjects) {
73. if (project.cost > maxCost) {
74. bestProject = project;
75. maxCost = project.cost;
76. }
77. }
78. ACLMessage reply = msg.createReply();
79. if (bestProject != null) {
80. reply.setPerformative(ACLMessage.INFORM);
81. reply.setContent("Project: " + bestProject.aid.getLocalName() + ", Project Payment: " + bestProject.cost + ", Required Programmers: " + bestProject.requiredProgrammers + ", Minimum Experience: " + bestProject.minimumExperience);
82. send(reply);
83. ACLMessage projectMsg = new ACLMessage(ACLMessage.INFORM);
84. projectMsg.addReceiver(bestProject.aid);
85. // projectMsg.setContent("Adding programmer: " + msg.getSender().getLocalName());
86. projectMsg.setContent(msg.getSender().getName());
87. send(projectMsg);
88. } else {
89. reply.setPerformative(ACLMessage.INFORM);
90. reply.setContent("No suitable projects found");
91. }
92. send(reply);
93. }
94. } else {
95. block();
96. }
97. }
98. }
99. private static class Project {
100. AID aid;
101. int cost;
102. int requiredProgrammers;
103. int minimumExperience;
104. Project(AID aid, int cost, int requiredProgrammers, int minimumExperience) {
105. this.aid = aid;
106. this.cost = cost;
107. this.requiredProgrammers = requiredProgrammers;
108. this.minimumExperience = minimumExperience;
109. }
110. }
111. }