Пример создания модели в AGNES

Подкорытов Дмитрий Игоревич

6 июля 2012 г.

Рассмотрим стандартный механизм создания модели работающей в среде AGNES. Возьмем следующую задачу: есть некий граф дорог, на котором расположены автомобили. Каждый автомобиль оборудован устройством для приема-передачи сообщений. На площади так же расположены более мощные стационарные передатчики. Один из автомобилей фиксирует событие, информацию о котором он должен передать выделенной стационарной вышке. Необходимо создать модель распространения сигнала внутри описанной сети.

Для создания агентной модели в качестве атомарной единицы (агента) выберем приемник-передатчик, характеризующийся: координатой, дальностью действия, функцией вероятности доставки сообщения. Для создания агентов AGNES необходимо знать JAVA хотя бы на начальном уровне. Рассмотрим пошаговое создание простой реализации агентов, реализующих описанную модель:

1. Необходимо создать класс-наследник класса AgnesFunctionalAgent, назовем новый класс Transmitter;

```
public class Transmitter extends AgnesFunctionalAgent
```

2. У агента-передатчика будет 3 атрибута: координаты (X,Y) и максимальный радиус действия передатчика. Для того, чтобы передать аргументы из конфигурационного файла в конструктор агента используется метод setInitParam. В качестве аргументов метод принимает два строчных параметра: название и значение аргумента. Радиус действия агент будет получать из файла конфигурации, а координаты случайным образом будут устанавливаться в конструкторе setup;

```
protected Integer coordX;
protected Integer coordY;
protected Integer maxRadius;

// Конструктор агента AGNES
protected void setup() {
    super.setup();
    coordX = RandomGenerator.getRandomInt(1000);
    coordY = RandomGenerator.getRandomInt(1000);
}

// Метод обработки параметров из
```

```
// конфигурационного файла
protected void setInitParam(String name, String val){
    super.setInitParam(name, val);
    if (name.equals("R")) {
        maxRadius = Integer.parseInt(val);
    }
}
```

3. Основным способом взаимодействия между агентами является обмен сообщениями. В нашей модели агенты будут принимать и обрабатывать сообщение о событии (содержащее только число соответствующее количеству агентов через которые прошло сообщение). В простой модели будем следить за одним сообщением, для этого у каждого агента будет флаг обозначающий, прошла обработка сообщения о событии или нет. Когда агент получает сообщение, он сперва проверяет состояние флага, если сообщение уже получалось, то ничего не делается, если не обрабатывалось, то:

   - Агент увеличивает счетчик внутри сообщения.
   - Переправляет новое сообщение своим соседям.
   - Взводит флаг обработки сообщения.

   Для обработки сообщений необходимо реализовать абстрактный метод MessageProcessing, в который приходят 3 параметра:

   - Имя отправителя.
   - Текст Сообщение.
   - Идентификатор агента отправителя.

```
protected boolean isMessageProcessed = false;
protected void MessageProcessing(String sender,
                        String mess, AID aidSender) {
  if (!isMessageProcessed) {
    Integer currentIteration = Integer.parseInt(mess);
    currentIteration++;
    ResendMessage(currentIteration.toString());
    isMessageProcessed = true;
  }
}
```

4. Для поиска агентами друг друга используется сервис “желтых страниц” JADE. Агенту-передатчику нужно знать своих соседей, чтобы взаимодействовать с ними и передавать им сообщения. Для поиска агенты должны зарегистрироваться в “желтых страницах” и найти друг друга при помощи методов registerService и findAllService соответственно. Соседями выбираются агенты, находящиеся на расстоянии меньшем радиуса действия передатчика. Для того, чтобы считать расстояние, расширим сообщения агента. Для получения информации о координатах другого агента будет использоваться сообщение “***getCoord***”, в ответ будет отправляться сообщение вида “***myCoord#X#Y***”. Т.е. в методе MessageProcessing появится следующий код:

```
if ("getCoord".equals(mess)) {
  String message = "myCoord#" + coordX + "#" + coordY;
  SendMessage(aidSender, message);
} else if (mess.startsWith("myCoord")) {
  String[] coords = mess.split("#");
  Integer X = Integer.parseInt(coords[1]);
  Integer Y = Integer.parseInt(coords[2]);
  Integer R2 = (coordX - X) * (coordX - X)
          + (coordY - Y) * (coordY - Y);
  if (maxRadius > Math.sqrt(R2)) {
      if (!neighbours.contains(aidSender)) {
          neighbours.add(aidSender);
      }
  }
}
```

5. Для синхронизации работы модели агенты должны использовать два метода:

   - `iAmReady` должен вызываться когда агент загружен и сделал все подготовительные работы.
   - `PreRunning` вызывается когда все агенты в модели готовы к работе.

   Таким образом в конструктор нужно добавить метод по регистрации агента в “желтых страницах”. А в методе PreRunning нужно найти всех передатчиков и отправить им запрос на получение координат.

```
protected void setup() {
    super.setup();
    coordX = RandomGenerator.getRandomInt(1000);
    coordY = RandomGenerator.getRandomInt(1000);
    registerService("Transmitter");
    iAmReady();
}
protected void PreRunning() {
    super.PreRunning();
    AID[] transmits = findAllService("Transmitter");
    for (int i = 0; i < transmits.length; i++) {
        SendMessage(transmits[i], "getCoord");
    }
}
```

6. Последнее, что осталось реализовать для получения первой модели, это сбор данных и инициализирующее сообщение о событии, для передачи его в сети. Чтобы сохранить данные на диск используется метод writeDataOnDisk, принимающий два параметра: название файла и строку с информацией. При получении сообщения о событии агент будет сохрянять его вместе со своими координатами в файл. Для инициализации процесса передачи сообщения, после старта модели сделаем

так, чтобы 0 агент с 5-тисекундным ожиданием отправлял сообщение о событии своему первому соседу. Для отложенного выполнения действия можно использовать поведение DelayBehaviour. У каждого агента Agnes есть его идентификационный номер, который хранится в аттрибуте myIdentificationNumber. Добавим две строки в метод MessageProcessing и расширим метод PreRunning.

```
protected void MessageProcessing(
  ...
  String result = coordX + ";"
        + coordY + ";" + currentIteration;
  writeDataOnDisk("FirstModel.csv", result);
  ...
}

protected void PreRunning() {
  ...
  if (myIdentificationNumber == 0) {
    addBehaviour(new DelayBehaviour(this, 5000) {
      protected void mainAction() {
        if (neighbours.size() > 0) {
          SendMessage(neighbours.get(0), "0");
        }
      }
    });
  }
}
```

7. Осталось только запустить модель. Создадим модель из 50-ти передатчиков и запустим моделирование сроком на одну минуту. Конфигурационный файл модели выглядит следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<AgnesConfig workTime="60">
    <Agent className="Transmitter">
        <Param count="50" R="500"/>
    </Agent>
</AgnesConfig>
```

Итоговый код агента:

```
package AGNES.model;

import AGNES.core.*;
import AGNES.core.behaviours.DelayBehaviour;
import AGNES.helpers.*;
import jade.core.*;
import java.util.*;

/**
 *
```

```
 * @author Podkorytov
 */
public class Transmitter extends AgnesFunctionalAgent {

  protected Integer coordX;
  protected Integer coordY;
  protected Integer maxRadius;
  protected List<AID> neighbours = new ArrayList<AID>();

  protected void setup() {
    super.setup();
    coordX = RandomGenerator.getRandomInt(1000);
    coordY = RandomGenerator.getRandomInt(1000);
    registerService("Transmitter");
    iAmReady();
  }

  protected void setInitParam(String name, String val) {
    super.setInitParam(name, val);
    if (name.equals("R")) {
      maxRadius = Integer.parseInt(val);
    }
  }
  protected boolean isMessageProcessed = false;

  protected void MessageProcessing(
      String sender,
      String mess,
      AID aidSender) {
    if ("getCoord".equals(mess)) {
      String message = "myCoord#"
          + coordX + "#" + coordY;
      SendMessage(aidSender, message);
    } else if (mess.startsWith("myCoord")) {
      String[] coords = mess.split("#");
      Integer X = Integer.parseInt(coords[1]);
      Integer Y = Integer.parseInt(coords[2]);
      Integer R2 = (coordX - X) * (coordX - X)
          + (coordY - Y) * (coordY - Y);
      if (maxRadius > Math.sqrt(R2)) {
        if (!neighbours.contains(aidSender)) {
          neighbours.add(aidSender);
        }
      }
    } else {
      if (!isMessageProcessed) {
        Integer currentIteration = Integer.parseInt(mess);
        String result = coordX + ";"
            + coordY + ";" + currentIteration;
        writeDataOnDisk("FirstModel.csv", result);
```

```
            currentIteration++;
            ResendMessage(currentIteration.toString());
            isMessageProcessed = true;
          }
        }
      }

      protected void ResendMessage(String newMess) {
        for (AID neighbour : neighbours) {
          SendMessage(neighbour, newMess);
        }
      }

      protected void PreRunning() {
        super.PreRunning();
        AID[] transmitters = findAllService("Transmitter");
        for (int i = 0; i < transmitters.length; i++) {
          SendMessage(transmitters[i], "getCoord");
        }
        if (myIdentificationNumber == 0) {
          addBehaviour(new DelayBehaviour(this, 5000) {

            protected void mainAction() {
              if (neighbours.size() > 0) {
                SendMessage(neighbours.get(0), "0");
              }
            }
          });
        }
      }
    }
```