ЛАБОРАТОРНАЯ РАБОТА № 5
Программирование пользовательских онтологий в JADE
Язык содержания FIPA-SL0

Цели работы:

1. Освоить элементарные навыки программирования пользовательских онтологий в JADE.

2. Изучить принцип построения ACL-сообщений с использованием языка содержания FIPA-SL0 и пользовательских онтологий.

5.1. Теоретические сведения

5.1.1. Онтологии в JADE

Общение агентов предполагает наличие у них некоторых общих знаний о предметной области, которые называются онтологией. Онтология включает в себя термины, которые будут использоваться в общении, и знания, относящиеся к терминам. Эти знания содержат определения терминов, их атрибуты, а также отношения между терминами и ограничениями. Подходы к реализации онтологий в контексте интеллектуальных агентов можно разделить на 3 категории:

- онтологии для частных случаев;
- «стандартные»;
- глобальные онтологии.

Частные онтологии заданы неявно в программном коде агентов и не могут быть повторно использованы в других программных продуктах. «Стандартные» онтологии позволяют преодолеть отмеченные выше ограничения путем использования онтологий, которые могут быть общими для различных разработчиков программного обеспечения. Глобальные онтологии (также называемые общими или онтологиями верхнего уровня) ставят задачу определения наиболее общих понятий, не относящихся к конкретным предметным областям.

Применительно к мультиагентным системам онтология есть система определений сущностей, о которых беседуют агенты. Пусть, например, один агент сообщает другому, что температура воздуха на улице +20 °C. Для того, чтобы второй агент мог понять это сообщение, оба агента должны иметь общую онтологию, содержащую понятия «температура», «воздух», «улица», «градус Цельсия».

Агент А, посылая агенту Б некоторое ACL-сообщение, передает ему определенный объем информации I. Внутри ACL-сообщения информация I представлена выражением на определенном языке содержания (например, языке SL0) и в определенном формате (например, строковом). Агенты А и Б могут иметь собственное внутреннее представление I.

Внутреннее представление I должно быть удобным для обработки информации, поэтому ACL-сообщения для этих целей не используются. Например, факт *«студент Андрей, ему 23 года»* будет представлен в ACL-сообщении в строковом формате следующим образом:

```
(student (name Andrei) (age 23))
```

Хранить данную информацию внутри агента в строковом виде неудобно, для обработки ее целесообразно представить в виде Java-объекта. Например, приведенная выше информация может быть представлена экземпляром следующего класса:

```
Class Student
{
  String name;
  int age;
  public String getName() {return name; }
  public void setName(String n) {name = n; }
  public int getAge() {return age; }
  public void setAge(int a) {age = a; }
...
}
```

Инициализируется экземпляр класса следующим образом:

```
Student stud = new Student();
stud.setName(“Andrei”);
stud.setAge(23);
```

Java-объект может быть преобразован в строковое значение поля :content ACL-сообщения. Для этого в JADE предусмотрены два метода, реализованные в классе ContentManager: fillContent(ACLMessage msg, ContentElement content) и extractContent(ACLMessage msg).

В процессе преобразования могут использоваться разные кодеки языка содержания, подключаемые методом registerLanguage(), который влияет на поле :language ACL-сообщений. Для объекта «Действие» необходимо выпол-

нить следующую последовательность, где reg – экземпляр класса, реализующего интерфейс AgentAction:

```
Action a = new Action();
a.setActor(receiver);
a.setAction(reg);
try { manager.fillContent(msg, a); }
catch (Exception pe) { pe.printStackTrace(); }
```

В JADE реализован встроенный кодек языка SL0, подключаемый следующим образом:

```
registerLanguage(new SLCodec(0));
```

Типовая настройка для работы с кодеком и онтологией выполняется следующим образом:

```
  private ContentManager manager  = (ContentManager)
                                   getContentManager();
  // Этот агент "разговаривает " на языке SL
  private Codec codec = new SLCodec();
  // This agent "knows" the Music-Shop ontology
  private Ontology ontology = StudentOntology.
                                        getInstance();

  protected void setup()
  {
    manager.registerLanguage(codec);
    manager.registerOntology(ontology);
    ...
  }
```

Для создания онтологии в JADE необходимо выполнить следующую последовательность действий:

1. Разработать класс для каждой роли в онтологии (Концепция, Действие, Предикат).

2. Создать подкласс Ontology.

3. Добавить роли онтологии к данному объекту.

Каждая роль в онтологии описывается названием и совокупностью слотов. Для уже рассмотренного примера, где понятие *студент* описывается классом Student, добавление роли к представляющему онтологию myOnto объекту выполняется следующим образом:

```
Ontology myOnto = new DefaultOntology();
...
ConceptSchema cs = (ConceptSchema) getSchema(STUDENT);
cs.add("name", (PrimitiveSchema) getSchema
    (BasicOntology.STRING), , ConceptSchema.MANDATORY);
cs.add("age", (PrimitiveSchema) getSchema
                              (BasicOntology.STRING));
```

Здесь Ontology – встроенный в JADE интерфейс, описывающий онтологию; DefaultOntology – класс, реализующий этот интерфейс.

Каждый слот содержит:

– наименование, идентифицирующее слот;

– категорию, указывающую на тип значения слота;

– фактор обязательности.

Примитивное значение слота может быть строковым или цифровым (String, Integer). В этом случае вызывается метод с приведением возвращаемого значения:

```
(PrimitiveSchema) getSchema (BasicOntology.STRING).
```

Если значением слота является экземпляр некоторой онтологической сущности, то вызывается метод:

```
(ConceptSchema) getSchema(<Имя роли>).
```

При задании фактора обязательности используются значения:

– ConceptSchema.MANDATORY – слот обязательно имеет значение;

– ConceptSchema.OPTIONAL – значение слота может быть не указано.

Чтобы зарегистрировать онтологию для данного агента, используется метод registerOntology(Ontology o) класса ContentManager (см. текст программы в прил. 1).

Поле того, как агент зарегистрировал онтологию, он может извлекать содержание полученного ACL-сообщения вызовом следующего метода:

```
ContentElementList CEList = extractContent(msg)
```

5.1.2. Язык содержания FIPA-SL0

Спецификация FIPA определяет синтаксис и семантику языка FIPA Semantic Language (FIPA-SL), предназначенного для совместного использования с FIPA-ACL. Язык FIPA-SL является универсальным формализмом представления содержания сообщения.

Поле **:content** ACL-сообщения может содержать SL-выражения трех типов:

- *Высказывание* (proposition или predicates), которому в данном контексте может быть присвоено значение истинности («true» или «false»). Высказывание представляет собой правильно построенную формулу (ППФ) и используется в коммуникативном акте типа **inform** и производных от него коммуникативных актах.
- *Действие* (action), которое может быть выполнено. Действие может быть одиночным или составным, построенным с использованием операторов последовательности и альтернативы. Действие используется как выражение содержания в коммуникативном акте типа **request** и производных от него коммуникативных актах.
- *Идентифицирующее ссылочное выражение* (identifying reference expression – IRE), которое идентифицирует объект предметной области. Оно представляет собой ссылочный оператор и используется в коммуникативном акте типа **inform-ref** и производных от него коммуникативных актах.

Ниже приведен пример ACL-сообщения:

```
(REQUEST
   :sender (agent-identifier :name SendAgent@PK1:1099/
                                                    JADE)
   :receiver (set(agent-identifier :name RecvAgent@
                                        PK1:1099/JADE))
   :content "((action (agent-identifier :name hh@PC:
                                             1099/JADE)
     (REGISTER :student (STUDENT :name Ivanov
                                  :groupnumber \"8307\")
               :course (COURSE :name \"MAS Course\"
                        :instructor
        (INSTRUCTOR :name Panteleev :dept CS)))))"
                    :language  fipa-sl
                    :ontology  student-ontology
  )
```

Поле :content имеет следующую структуру:

```
:content (action1 (role1 :parameter1 value1
                         :parameter2 value2)
                  (role2 :parameter3 value3
                         :parameter4
```

```
(role3 :parameter5 value5)))
```

Значение value1 обычно представляется в строковом виде. Если оно содержит внутри себя пробелы, то заключается в кавычки. Пример:

```
:name "Ivan Petrov"
```

5.2. Порядок выполнения работы

В данной лабораторной работе предлагается расширить пользовательскую онтологию StudentOntology, содержащую следующие базовые роли:

- концепции: *Студент, Преподаватель, Курс* (*Student*, *Instructor*, *Course*);
- действия: *Регистрировать* (*Register*);
- утверждения: *Курс доступен* (*CourseAvailable*);
- предикаты: *Зарегистрирован* (*Registered*).

Каждая из этих ролей реализована одноименным классом.

На рис. 5.1 приведена модель иерархии классов StudentOntology в UML.

Примечание: в языке Java строчные и прописные буквы различаются, в том числе и в именах файлов классов. Поэтому всегда необходимо соблюдать регистр (верхний и нижний).

1. Изучить пример пользовательской онтологии StudentOntology, для чего проанализировать диаграмму классов StudentOntology и сравнить ее с программной реализацией соответствующих классов в приведенном листинге (прил. 1).

2. Расширить онтологию StudentOntology в соответствии с выданным вариантом задания.

Варианты заданий:

2.1. Расширить класс Instructor представлением информации о должности преподавателя. Например, добавить поле «position».

2.2. Добавить действие «deregister», отменяющее регистрацию студента на данный курс, и предикат «deregistered», соответствующий факту отмены регистрации на данный курс.

2.3. Добавить в концепцию «курс» внутреннюю концепцию «расписание_курса» («CourseSchedule»). Пример формата (CourseSchedule :day Monday :time 13-15).

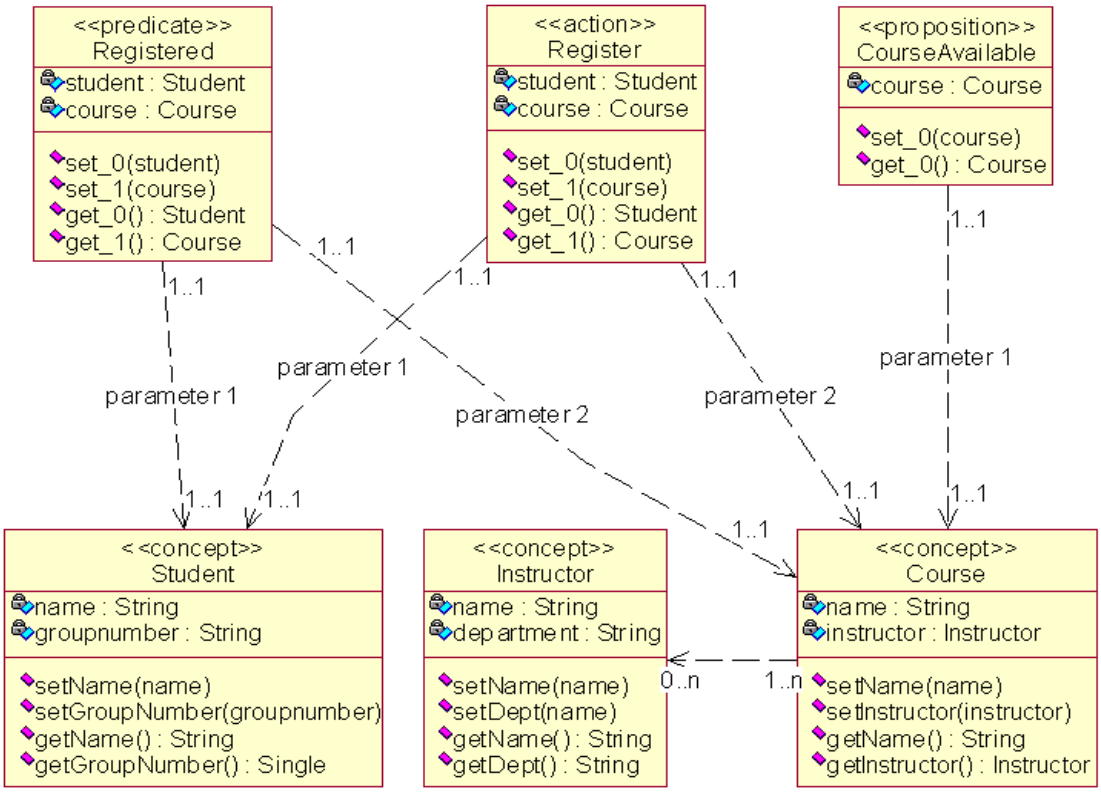

Рис. 5.1. Иерархия классов StudentOntology

2.4. Расширить класс Student, добавив в него поле «age».

2.5. Вывести общие слоты для классов Student и Instructor в новый класс Person (name) и затем сделать классы Student и Instructor дочерними для Person.

3. Запрограммировать расширение онтологии, модифицируя приведенный программный код.

Для расширения онтологии путем добавления в нее новой роли необходимо сделать следующее:

3.1. Определить соответствующий данной роли java-класс, для чего можно воспользоваться примерами уже реализованных классов для других ролей онтологии.

3.2. Добавить определение новой роли в файл онтологии StudentOntology.java, руководствуясь образцами определения уже имеющихся ролей.

Пусть, например, необходимо добавить роль «deregister», означающую отмену регистрации студента на определенный курс. Формат представления роли следующий:

```
(deregister :student student_name :course course_name).
```

Определение соответствующего java-класса deregister.java следующее:

```
class deregister
{
  private Student student;
  private Course course;
  public void setStudent(Student s) { student=s;}
  public Student getStudent() { return student; }
  public void setCourse(Course c) { course=c; }
  public Course getCourse() { return course; }
}
```

Примечание: если поля значений у роли именованные, то в классе, описывающем данную роль, методы установки и считывания значения поля записываются в виде getName или setName. Если поля значений неименованные, то используются методы вида set_x и get_x, где x – порядковый номер поля. Несоблюдение этого правила приведет к ошибке декодирования ACL-сообщения, например, при вызове метода extractContent(msg).

Для регистрации роли «deregister» в онтологии StudentOntology.java в раздел «actions» надо добавить константу «DEREGISTER»:

```
// Actions
 public static final String DEREGISTER = "DEREGISTER";
```

В конструктор StudentOntology() надо добавить следующий код:

```
private StudentOntology(Ontology base)
{
  ...
  add(new AgentActionSchema(DEREGISTER),
                                    Deregister.class);
  AgentActionSchema as = (AgentActionSchema)
                                  getSchema(DEREGISTER);
  as.add("student", (ConceptSchema)
                                   getSchema(STUDENT));
  as.add("course", (ConceptSchema) getSchema(COURSE));
  ...
}
```

4. Реализовать отправку и получение ACL-сообщения с использованием данных из расширенной онтологии. Для этого написать код двух агентов, один из которых будет отправлять сообщение, а другой – получать.

5.3. Содержание отчета

1. Определение онтологии.
2. Диаграмма классов пользовательской онтологии StudentOntology.
3. Вариант задания на расширение пользовательской онтологии.
4. Диаграмма классов пользовательской онтологии StudentOntology, расширенная в соответствии с заданием.
5. Исходные тексты Java-файлов с классами получающего и принимающего сообщений агентов, а также с классами расширенной онтологии.
6. Текст bat-файла для запуска агентов.
7. Протокол общения агентов (из Sniffer).
8. Выводы.

5.4. Вопросы для самоконтроля

1. Что такое онтология?
2. Какими методами реализуется пользовательская онтология в JADE?