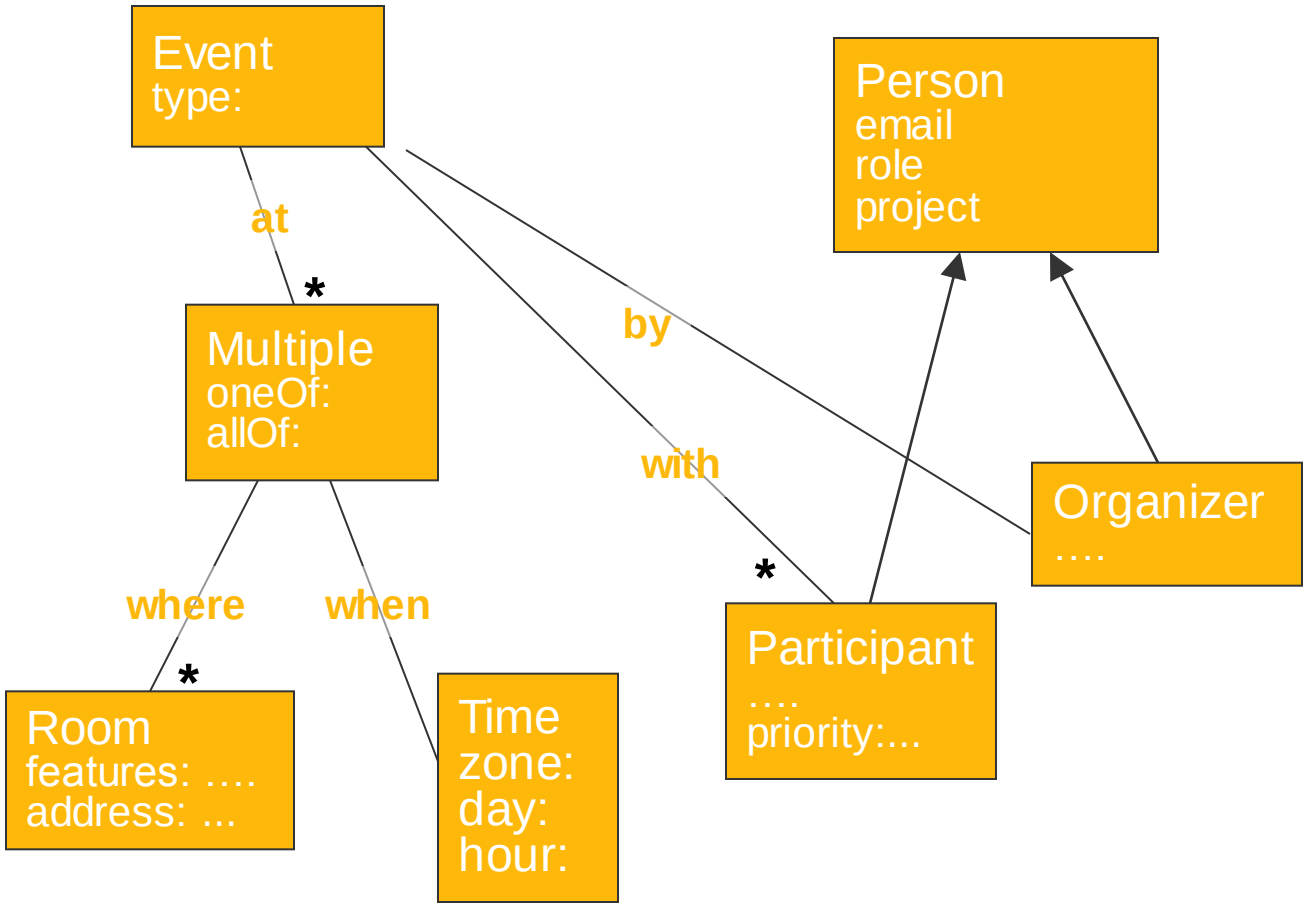
JADE Ontologies

2012-2013

Пример за онтология



Онтологията и езиците за съдържание

- Действителната информация, която е предавана от агента за изпращане до агента за получаване на ACL съобщението,е включена в слот за съдържание на съобщението.
- Според спецификациите на FIPA стойностите на тези слотове може да бъдат или низови типове данни, или необработена последователност от байтове.
- Може да се наложи агентите да си предават сложи съобщения, съдържащи множество слотове със съдържания.

Онтологията и езиците за съдържание

- Когато се предава сложна информация е необходимо да се приеме добре дефиниран синтаксис.
- Може да се анализира съдържанието на съобщението от страна на приемащия агент и да се извлече всяко специфично парче от информация до вид, който е изпратен от изпращащия агент.
- Според FIPA този синтаксис е познат като език за съдържание. FIPA не налага точно определен език за използване, но дефинира и препоръчва да използва езикът SL, когато комуникацията е с AMS и DF.

Пример за информация, кодирана с SL

(Book :title "Programming Multi Agent Systems with JADE" :authors(sequence "F. Bellifemine" "G. Caire" "D. Greenwood") :editor Wiley)

Когато получаващият агент получи такъв низ, той трябва да може да анализира този SL синтаксис, за да разбере информацията, която синтаксиса носи със себе си.

Онтологията и езиците за съдържание

- Набор от концепции и символи използвани са описание са познати като *онтология*.
- За разлика от езиците за съдържание, които обикновено са независими от дадена област, то онтологиите обикновено са специфични за дадена област.
- Докато е удобно един низ от сложна информация да се включи в едно ACL съобщение, то в ответната страна е доста неудобно тази информация да се анализира от приемащия агент.

Онтология

- Дефинира се собствен речник и семантика за съдържанието на съобщението, което се обменя между агентите.
- JADE осигурява три начина за имплементиране на комуникацията между агенти.

Методи за комуникация

- Използване на низ от символи за предствяне на съдържанието.
- Използване на Java технологията за предаване на сериализирани Java обекти директно като съдържание на съобщението.
- Дефиниране на обекти, които се предават, които са разширения на предифинирани класове.

Преглед на методите за комуникация

- 1st Method:
  - Content type: Strings
  - Getting method: getContent()
  - Setting method: setContent()
- 2nd Method:
  - Content type: Java Objects
  - Getting method: getContentObject()
  - Setting method: setContentObject()
- 3rd Method:
  - Content type: Ontology Objects
  - Getting method: extractContent()
  - Setting method: fillContent()

Примери

- The bank example consists of 2 versions
  - The first version, Bank-1-jObjects, shows how to implement communication between agents using Java objects.
  - The second version, Bank-2-Onto, shows how to implement the communication between agents using an ontology.

Пример Bank - Java обекти

- Two agents implement the client and server roles for the bank with savings accounts
- The **`BankServerAgent`** class, acts as a server and the **`BankClientAgent`** class acts as client.
- The two classes use (*implement*) a common interface, **`BankVocabulary`**, that defines the constants which represent the terms that constitute the specific language of the agents.

Пример Bank - Java обекти

- Протоколи за комуникация
  - **`REQUEST`**: The client agent sends this message to create an account or to make an operation.
  - **`INFORM`**: The server agent responds with this message after processing a **`REQUEST`**.
  - **`NOT_UNDERSTOOD`**: The client and the server agents respond with this message if it cannot decode the content of the message.
  - **`QUERY_REF`**: The client agent sends this message to the server agent to query information about a specific account.

Пример Bank - Java обекти

- Classes of Bank Example
  - **`Account`**: concept of a bank savings account
  - **`Operation`**: concept of a bank operation
  - **`MakeOperation`**: action of making an operation such as deposit or withdrawal
  - **`OperationList`**: concept of the list of last operations
  - **`CreateAccount`**: action of creating an account
  - **`Information`**: concept of querying information about an account such as the balance and the list of last operations
  - **`Problem`**: result of an action that fails

Пример за предаване на Java обекти

- Ако обектите могат да се предадат като част от съобщението, то трябва да се декларира като имплементира **`java.io.Serializable`** интерфейса.

Пример за предаване на Java обекти

```
class MakeOperation implements
java.io.Serializable
 {

     private String accountId;
     private int type;
     private float amount;

     public String getAccountId() {
         return accountId;
     }

     public int getType() {
         return type;
     }

     public float getAmount() {
         return amount;
     }

     public void setAccountId(String
                                     accountId) {
         this.accountId = accountId;
     }

     public void setType(int type) {
         this.type = type;
     }

     public void setAmount(float amount) {
         this.amount = amount;
     }
}
```

Пример за предаване на Java обекти

```
MakeOperation mo = new MakeOperation();
mo.setAccountId(acc.getId());
mo.setType(command);
mo.setAmount(amount);

ACLMessage msg = new ACLMessage(
  ACLMessage.REQUEST );
msg.addReceiver(server);
try {
   msg.setContentObject( mo );
}
catch (Exception ex) { ex.printStackTrace(); }
send(msg);
```

Пример за предаване на Java обекти

```
class ReceiveMessages extends CyclicBehaviour
{
   public ReceiveMessages(Agent a) {
     super(a);
   }
   public void action()
   {
     ACLMessage msg = receive();
     if (msg == null) { block(); return; }
     try {
       Object content = msg.getContentObject();
       switch (msg.getPerformative()) {
       case (ACLMessage.REQUEST):
       if (content instanceof CreateAccount)
         addBehaviour(new HandleCreateAccount(myAgent, msg));
       else if (content instanceof MakeOperation)
         addBehaviour(new HandleOperation(myAgent, msg));
       ...
   }
}
```

Пример за предаване на Java обекти

```
class HandleOperation extends OneShotBehaviour
{
   ACLMessage request;
   public HandleOperation(Agent a, ACLMessage
     request) {
     super(a);
     this.request = request
   }
   public void action() {
     try {
       Operation op =
         (Operation) request.getContentObject();
       ACLMessage reply = request.createReply();
       Object result = processOperation(op);
       ...
     } catch (Exception ex) { ex.printStackTrace(); }
   }
}
```

Недостатъци 1/2

- Изпращащия агент трябва да преобразува предаваната информацията в съответно дефинирано ACL съобщение и получаващият агент трябва да направи противоположно преобразуване.
- Получаващият агент трябва да направи поредица от семантични проверки, за да удостовери, че получената информация спазва правилата на онтологията, споделена между комуникиращите агенти.
- JAVA сериализация е приложима само за JAVA среди. Ако един JADE агент трябва да предаде информация до друг агент, от отдалечена FIPA съвместима платформа различна от JADE, няма абсолютно никаква гаранция, че получателят ще разбере информацията от слота за съобщения, кодирана чрез JAVA сериализация.

Недостатъци 2/2

- JAVA сериализация генерира формат, който е нечетлив от хората. В много случай възможността да се прочете информацията от слота за съдържание в дадено съобщение е изключително полезна в процес на изследване на даден проблем.
- Агента, който получава съобщение няма никакви средства за определяне на вида на обекта, който той ще получи, когато декодира информацията от слот за съдържание.

Bank Example 2 - Ontology

- An ontology describes the elements that can be used as content of agent messages.
- An ontology is composed of two parts:
  - The vocabulary that describe the terminology of concepts used by agents in their space of communication.
  - The nomenclature of the relationships between these concepts, and that describe their semantics and structure.
- An ontology is implemented by extending the class **Ontology** predefined in JADE and adding a set of element schemas describing the structure of concepts, actions, and predicates that are allowed to compose the content of your messages.
- You may also extend directly the basic ontology classes **BasicOntology** or **ACLOntology** according to your need.

Bank Example 2 - Ontology

- In this example, we defined the BankOntology class that our two agents use to communicate in place of the java objects.

- We re-use our java objects from before, but instead of using them directly in the content of messages, we just wrap them into specific terms and concepts defined within the BankOntology class.

Bank Example 2 - Ontology

- The **`AgentActionSchema`** class inherits from the **`ConceptSchema`** class which in turn is a subclass of the **`TermSchema`** class. While the **`PredicateSchema`** class inherits from the **`ContentElementSchema`** class.

- But at the base, these interfaces have a common superclass which is the **`ObjectSchema`** class.

Bank Example 2 - Ontology

- java.lang.Object
  - jade.content.schema.ObjectSchema
    - jade.content.schema.ObjectSchemaImpl
      - jade.content.schema.TermSchema
        - jade.content.schema.ConceptSchema
        - jade.content.schema.AgentActionSchema
- java.lang.Object
  - jade.content.schema.ObjectSchema
    - jade.content.schema.ObjectSchemaImpl
      - jade.content.schema.ContentElementSchema
        - jade.content.schema.PredicateSchema

Bank Example 2 - Ontology

- An important point to know is when to use one or another of these ontology objects
- Agent A requests agent B to perform a specific task.
  - FIPA requires the content of the message that A sends to B must be an action.
  - In JADE, the task will be defined by a java object implementing the *AgentAction* interface and the action will be an instance of the class **Action** to which you pass in arguments the AID of agent B and the object describing the task to be performed.
- Agent A asks agent B if a given proposition is true.
  - According to FIPA, the content of the message must be the object representing the proposition to check.
  - In JADE a proposition can be defined by a java object implementing the interface *Predicate*.
- Some objects are not agent actions neither propositions, so we define these as concepts
  - They implement the interface *Concept*.

Bank Example 2 - Ontology

- JADE also provides support for defining atomic elements that constitute generally the slots of the abstract concepts
  - String, Integer, Float, *etc.*
  - The support for these atomic types of objects is provided through the class **PrimitiveSchema** and handled by the *BasicOntology* class.

Bank Example 2 - Ontology

- **`Account`** class implements the **`Concept`** interface
- **`Operation`** class implements the **`Concept`** interface
- **`MakeOperation`** class implements the **`AgentAction`** interface
- **`CreateAccount`** class implements the **`AgentAction`** interface
- **`Information`** class implements the **`AgentAction`** interface
- **`Problem`** class implements the **`Concept`** interface

Bank Example 2 - Ontology

- Define the vocabulary of your agents communication space.

```
public interface BankVocabulary

{

   ...

   public static final String MAKE_OPERATION =
     "MakeOperation";

   public static final String MAKE_OPERATION_TYPE = "type";

   public static final String MAKE_OPERATION_AMOUNT =
     "amount";

   public static final String MAKE_OPERATION_ACCOUNTID =
     "accountId";

   ...

}
```

Bank Example 2 - Ontology

- Define the java class that specifies the structure and semantic of the object *MakeOperation*.
  - Same definition as Java Objects example except that it implements *AgentAction* and not *java.io.Serializable*

Bank Example 2 - Ontology

```
class MakeOperation implements AgentAction
{
    private String accountId;
    private int type;
    private float amount;

    public String getAccountId() {
        return accountId;
    }

    public int getType() {
        return type;
    }

    public float getAmount() {
        return amount;
    }

    public void setAccountId(String accountId) {
        this.accountId = accountId;
    }

    public void setType(int type) {
        this.type = type;
    }

    public void setAmount(float amount) {
        this.amount = amount;
    }
}
```

Bank Example 2 - Ontology

- Define the schema of the object

```
public class BankOntology extends Ontology implements
   BankVocabulary {
   public static final String ONTOLOGY_NAME = "Bank-Ontology";
   private static Ontology instance = new BankOntology();
   public static Ontology getInstance() { return instance; }

   private BankOntology() {
      super(ONTOLOGY_NAME, BasicOntology.getInstance());
      try {
      // MakeOperation
      add(as = new AgentActionSchema(MAKE_OPERATION), MakeOperation.class);
      as.add(MAKE_OPERATION_TYPE, (PrimitiveSchema)
         getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);
      as.add(MAKE_OPERATION_AMOUNT, (PrimitiveSchema)
         getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);
      as.add(MAKE_OPERATION_ACCOUNTID, (PrimitiveSchema)
         getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
   }
      catch (OntologyException oe) {
         oe.printStackTrace();
      }
      }
      }// BankOntology
```

Bank Example 2 - Ontology

```
public class BankClientAgent extends Agent implements BankVocabulary {

        ...
        private Codec codec = new SLCodec();
        private Ontology ontology = BankOntology.getInstance();

        protected void setup() {

        // Register language and ontology
        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(ontology);

        ...
        }
       void requestOperation() {

       MakeOperation mo = new MakeOperation();
       mo.setType(command);
       mo.setAmount(amount);
       mo.setAccountId(acc.getId());
       sendMessage(ACLMessage.REQUEST, mo);
       }
}//class BankClientAgent
```

Bank Example 2 - Ontology

```
public class BankServerAgent extends Agent implements
  BankVocabulary
{
   ...
   private Codec codec = new SLCodec();
   private Ontology ontology = BankOntology.getInstance();
   ...

   protected void setup() {

      // Register language and ontology
      getContentManager().registerLanguage(codec);
      getContentManager().registerOntology(ontology);
      ...
   }

   ...

  }// End BankServerAgent
```

Bank Example 2 - Ontology

```
class ReceiveMessages extends CyclicBehaviour
   {
      public ReceiveMessages(Agent a){
         super(a);
      }

      public void action()   {
         ACLMessage msg = receive();
         if (msg == null) { block(); return; }

        try {
         ContentElement content = getContentManager().extractContent(msg);
         Concept action = ((Action)content).getAction();

         switch (msg.getPerformative())   {
         case (ACLMessage.REQUEST):
           ...
            if (action instanceof CreateAccount)
                addBehaviour(new HandleCreateAccount(myAgent, msg));
            else if (action instanceof MakeOperation)
                addBehaviour(new HandleOperation(myAgent, msg));
            ...
            break;
}
        catch(Exception ex) { ex.printStackTrace(); }
      }
   }// End ReceiveMessages
```

Bank Example 2 - Ontology

```
class HandleOperation extends OneShotBehaviour
  {
     private ACLMessage request;

     HandleOperation(Agent a, ACLMessage request)
     {
        super(a);
        this.request = request;
     }

     public void action()
     {
       try {
         ContentElement content = getContentManager().extractContent(request);
         MakeOperation mo = (MakeOperation)((Action)content).getAction();
         //Process the operation
         Object obj = processOperation(mo);
         //Send the reply
         ...
       }
       catch(Exception ex) { ex.printStackTrace(); }
     }

  }// End HandleOperation
```