3. Ontologies

Last time, we introduced the communication protocols, but we used plain text messages to send information among the agents. Such an approach can be used for smaller projects, however, one we need to send more complicated (structured) messages, it becomes less and less usable. Remember, how we sent lists delimited by "|" last time and imagine, what would happen if we needed to send something more complicated.

In Jade, we can send serialized Java objects in the messages. It is much more usable for more complex information. However, it still has at least two disadvantages. First, the multi-agent system is no longer multi-platform, as all agents need to be able to process Java objects, and second, we cannot use to sniffer to observe the messages sent by the agents. Both of these problems can be solved by ontologies.

Ontologies, apart from the description of the agent environment, provide a simple and useful possibility to communicate among the agent.

Definition of a new ontology

For simple projects, it is often easier to define an ontology manually, specifically for the problem we want to handle. There are two ways how to define an ontology in Jade. The older one needs and dictionary (user created) and a precise description of the properties of each concept in the ontology. The new approach defines and ontology as a JavaBeans hierarchy. Then, we only need to create a BeanOntology class and specify which classes (or, better, packages) are part of the ontology.

Each ontological class implements (either directly or indirectly) one of three interefaces – `AgentAction`, `Concept`, or `Predicate`. It is important to know, which one is used in which situation.

1. If agent A asks an agent B to perform an action, the content of the message must be the AID of agent B and an `AgentAction`, it should perform. Actions in Jade are defined in the `Action` class.
2. If agent A asks agent B, if a predicate is true, it sends a message with a `Predicate`.
3. Everything else is a `Concept`, typically, these are the parameters of actions, results of actions, and so on. Results are (together with the action they react to) wrapped in a `Result` class.

In the source codes for today's seminar, there is a definition of a simple ontology for the book trading example we did last time. It contains only three simple classes: `BookInfo`, `GetBookList` and `SellBook` and a class which defines the whole ontology – `BookOntology`. `BookInfo` is a `Concept`, it contains information about a book (name and price). `GetBookList` and `SellBook`, on the other hand, are `AgentAction`. As you can see, the classes are pretty simple. They only implement getters and setters for their properties and they also contain a few annotations, which express whether a property is optional or mandatory, and what is the type of the result of an action.

Exercise 1 Have a look at the ontology in mas.cv3.onto, its classes and their annotations.

Using the ontology

To work with an ontology, we have to use the `ContentManager`. It is obtained by calling the `getContentManager()` method. Before and agent can use an ontology, it needs to register it

with the `ContentManageru`. It also needs to specify the language which will be used for the communication (it basically describes the way of serialization and deserialization of the ontology classes). Both these actions are performed by the following two lines in the `setup()` method.

```
this.getContentManager().registerLanguage(codec);
this.getContentManager().registerOntology(onto);
```

where `codec` and `onto` are defined by the following two lines:

```
private Codec codec = new SLCodec();
private Ontology onto = BookOntology.getInstance();
```

Instead of `setContent` and `getContent` we now use `fillContent` and `extractContent`, provided by the `contentManager`. If an agent wants to send a `REQUEST` to another agent and use our ontology, it can do the following (agent asks the seller for a list of books it sells).

```
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.addReceiver(seller);
GetBookList gbl = new GetBookList();
msg.setOntology(onto.getName());
msg.setLanguage(codec.getName());
getContentManager().fillContent(msg, new Action(seller, gbl));
send(msg);
```

The seller agent receives the message (we can again create a suitable `MessageTemplate` to receive only messages intended for a given behavior) and extracts the class it contains. We extract the action, and according to the action provide the result (the list of available books).

```
ContentElement ce = getContentManager().extractContent(request);
Action a = (Action)ce;

if (a.getAction() instanceof GetBookList) {

      ArrayList bis = new ArrayList();
      for (Object key : catalogue.keySet()) {
            BookInfo bi = new BookInfo();
            bi.setName(key.toString());
            bi.setPrice((Integer)catalogue.get(key));
            bis.add(bi);
      }

      ACLMessage reply = request.createReply();
      reply.setPerformative(ACLMessage.INFORM);
      getContentManager().fillContent(reply,
                                    new      Result(a.getAction(),
bis));
      return reply;
}
```

Exercise 2 Compare today's example with the one from last seminar. Is the use of ontologies simpler or more complicated?

Exercise 3 Repeat the programming exercises from last time with the onologies instead of plain text messages.

The fact that an action must contain the name of the agent sho should perform it as one of its paramters complicates the sending of CFPs and REQUESTs for the lists of books. This time, we need to use the `prepareRequests`, or `prepareCfps` methods from the respective behaviros and create a list of messages. The messages are slightly different for each agent.

There is one more problem connected with the onologies. The FIPA specification do not specify the content of the Contract-Net messages from the onological point of view. It is common to send the action, which shall be performed in the CFP messages, but there is a problem with the PROPOSE and ACCEPT_PROPOSAL messages. In this seminar, we will send a `Result`, with `BookInfo` with a price the agent is willig to accept for the books. `ACCEPT_PROPOSAL` will be empty, and the final `INFORM` will contain a results with a `BookInfo`.

What have we learned today

1. Define a new ontology
2. Send messages with an ontology
3. Process ontological messages
4. (On blackboard and on PC next time) Design a communication in a more complicated system for a book trading competition.