2. Communication Protocols

Communication protocols were created to allow the agents to communicate regardless of their implementation, including the environment and the programming language. The protocols describe what type of messages and when the agent can send, and what should happen in case anything fails.

Performatives

Each message must have a so called performative. It indicates what is the purpose of the messages, whether it is a request, information, etc. During the last seminar, we sent only messages with the INFORM performative. Although it works, it is not a correct use of the communication protocols. Moreover, it also slightly complicates the design of more complicated agents – agents can filter messages by performatives.

Message templates

Message templates (implemented in the `MessageTemplate` class, which has several `matchXXX` methods) can be used to filter messages. The most important match methods are `matchPerformative`, `matchCoversationID`, and `matchSender`. All of them have a string parameter and check, whether the string matches given part of message. You will almost never use ConversationID yourselves, however, it can be used if you want to make more communications at once, possibly with many different agents. These conversations can be distinguished by the ConversationID, and the behaviors, which implement the communication protocols, use them to this end.

You can also create a custom template to filter messages. You just need to implement the `MessageTemplate.MatchExpression` interface, which has a single `match` method. It should return true or false depending on whether it matches the given message or not.

Communication protocols

The most important protocols (for us during the seminar) are Request and Contract-Net. The Request protocol is used if an agent (Initiatior) wants to ask another agent (Responder) to do something. The Contract-Net protocol is a little more complicated – and agent (Initiator) first asks several different agents (Reponders) to propose conditions (e.g. price) under which they will perform the action. The responders then answer with a proposal and the Initiator chooses one or more agents to perform the action.

The protocols are governed by the FIPA organization and their precise definitions are available at http://www.fipa.org/repository/ips.php3.

Today's seminar

Today, we will use a different scenario from last time. It is in fact a (very) augmented Jade tutorial – selling and buying books. We have two types of agents – BookSellerAgent sells book and BookBuyerAgent buys books. Both types of agents can be present multiple times in the system. Different sellers can offer different books for different prices. The seller can tell which books it sells, it can sell a book and provide the price of any book it sells. The buying agent can ask the seller for a list of books for sale, and buy a selected book, or ask all the seller for the price and buy the book from the cheapest one.

The seller agent also has a rather simplistic user interface, which can be used to add a book to its catalogue. You can have a look at how the user interface can be created, but we will not discuss it any further in the seminar.

All seller agents have the books “LOTR” and “Hobbit” in their catalogue with a random price (so that you do not have to always add some book). The buying agent first asks all the seller for the list of books they sell, and tries to buy “LOTR” for the best possible price.

Request

Request is the simplest of the protocols. It is used in any situation, where an agent (Initiator) wants to ask another agent (Responder) to do something. If you look at the specification of the protocol, you can see that it contains only a few steps. The Initiator first sends a message with the REQUEST performative, where it asks the Responder to perform an action. The Responder can send an AGREE message, which means he will perform the action, or it can do perform the action right away and send back either an INFORM with the result of the action or DONE, if there is no result and the task is done.

In Jade, the Request protocol is implemented in a pair of behaviors: `AgentREInitiator` and `AgentREResponder`[1], which take care of the roles of Initiator and Responder. Each implementation of an agent use a behavior derived from these two, if it wants to use the Request protocol.

AchieveREInitiator

Using the `AchieveREInitiator` class is simple. The constructor takes the message, which shall be send and a reference to the agent which sends the message. The behavior takes care of the sending of the message itself. The only method which must be implemented is `handleInform` which is called when an INFORM message is received from the Responder. It is also possible to implement other methods (`handleAgree`, `handleRefuse`, or `handleFailure`) which handle messages with the appropriate performative.

`AchieveREInitiator` can also send a request to more than one agent at a time, the agents are just added as recipients to the message passed in the constructor. The answers from the Responders can be processed either one by one, or in the `handleAllResultNotifications` method, which gets all the answers at once.

AchieveREResponder

To create the `AchieveREResponder` class, we need to setup a `MessageTemplate` based on the type of messages we want to react to, most often, these are REQUEST messages of the FIPA_REQUEST protocol.

```
MessageTemplate template = MessageTemplate.and(
  MessageTemplate.MatchProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST),
  MessageTemplate.MatchPerformative(ACLMessage.REQUEST) );
```

Then, we need to implement a `handleRequest` method, which handles the request. This method returns a message which shall be sent as the response. If it returns `null`, or an AGREE message (it is optional in the protocol, however is should be sent especially in cases where the processing of the request can take longer time), we must implement the `prepareResultsNotification` method, which returns the results of the task (as an INFORM message).

[1] RE in the name of the class is an abbreviation for Rational Effect

Example

Today's source codes contain an example implementation of both the behaviors. `BookBuyerAgent` has a `ListBooks` behavior, which implements `AchieveREInitiator`, and asks all seller about the list of books they sell. (Responder is implemented as a `ListAvailableBooks` behavior).

`BookBuyerAgent` sends a REQUEST which contains `get-books-list` and `BookSellerAgent` responds with a list of books delimited by "|".

`BookSellerAgent` can also react to a REQUEST containing `sell-book|book`, it sells the book `book` to the agent who sent the request.

Exercise 8 Take a look at `BookBuyerAgent` and `BookSellerAgent` and inspect the behaviors of the for the Request protocol.

Exercise 9 Change the implementation of `BookBuyerAgent` in such a way that it asks all the sellers every minute for the list of books and attempts to buy one of them from a selected seller. You may use the fact that you can add a behavior to an agent from another behavior.

Contract-Net

The Contract-Net protocol is a little more complicated. The Initiator in this case sends a CFP (Call for Proposal) message to several Responders. This message contains the action the Responder should perform. Each Responder replies with a message (PROPOSE), in which it describes the conditions (e.g. price) for performing the action. It can also reply it will not perform the action at all (REFUSE). Initiator chooses from the replies some Responders to perform the action and sends them an ACCEPT_PROPOSAL message. It sends REJECT_PROPOSAL to the other Responders.

The first step of the protocol is the same as in the Request protocol, only the Initiator send a CFP instead of REQUEST. This is the way, how the Responder knows it should not perform the action, but send its conditions. The action is performer after the Responder receives the ACCEPT_PROPOSAL message.

Contract-Net Initiator

In Jade, the Initiator is implemented in the `ContractNetInitiator` behavior. First, we need to pass the CFP message, which shall be sent to the Responders, to the constructor. Then, the agent waits for the responses. Each time a response is received, the `handlePropose` or `handleRefuse` methods are called. After all responses are received, the `handleAllResponses` method is called. The initiator then selects the Reponders which will perform the action and sends them an ACCEPT_PROPOSAL message. It sends a REJECT_PROPOSAL to the rest of the Responders. The Initiator can send these messages as they arrive from the `handlePropose`, or wait for all responses and send the decisions from the `handleAllResponses` method. After the decisions are sent, the agent waits for the INFORM messages about the completion of the action and can handle them in the `handleInform` method.

Contract-Net Responder

The Responder of the Contract-Net protocol in implemented in `ContractNetResponder`. After it receives a CFP, it handles it with the `handleCfp` method. Here, it specifies its condition (PROPOSE) or rejects the task right away (REFUSE).

After it receives the ACCEPT_PROPOSAL message, it performs the action specified in the CFP. ACCEPT_PROPOSAL can be handled in the `handleAcceptProposal` method. The agent can also handle rejection of its offer in `handleRejectProposal`.

Example

An example implementation of the Contract-Net protocol can be found in the `BookSellerAgent` and `BookBuyerAgent` agents. The buyer uses the protocol to buy a book from a seller with the lowest price. The Initiator is implemented in `BookBuyerAgent` in the `BuyBook` class and the Responder Is implemented in `OfferBookPrices` in `BookSellerAgent`.

Exercise 10 Have a look at the above mentioned classes and find out how they work.

Exercise 11 Change the `BookBuyerAgent` in such a way that it buys your selected book from one of previous exercises from the cheapest seller.

Exercise 12 Try to connect to the system running on u-pl4 and trade among yourselves.

Exercise 13 Would it be possible to integrate the buyer and seller into a single trading agent, which would perform both buying and selling of books and were able to obtain books specified by the user? Would it be possible to write an agent which would make money by trading books?

Tips and tricks

1. If something strange happens during the communication, you do not have to create messages with error performatives (REFUSE, FAILURE, NOT_UNDERSTOOD). You can just throw the respective exception in the handler and Jade will take care of creating the message. The content of the message is set to the text of the exception.
2. In Jade, there is a `FIPAService` class, which has a `FIPAService.doFipaRequestClient(Agent a, ACLMessage request)` method. This method can take care of the Initiator of the Request protocol and returns the message sent by the Responder. Its main advantage (and disadvantage at the same time) compared to special behavior it that it blocks the calling behavior (and the whole agent) until the reply is received. It is useful in situation where the agent cannot continue without he reply, such as in the `setup()` and `takeDown()` methods.

What have we learnt today

Today, we discussed how the two most important protocols in MAS work and how they are implemented in Jade. The implementation of other protocol is very similar. We have also introduced more complicated behaviors which implement these protocols. We have also shown (but not discussed deeply), how to add a simple GUI to an agent.