Tutorial on Jess agents in DCaseLP

1 Creating a Jess program for the Jess agent

In order to create a Jess agent, you need first to create the Jess program for your agent.

1. Open the `JessAgentSkeleton.clp` that is contained in the `jessInJADE` directory with any text editor, and save it somewhere in you file system, with a name of your choice. Let us suppose that you saved the file as `C:\JessAgents\myFirstAgent.clp`.

2. You need to decide which role your agent plays, and which other agents (and their roles) it will interact with. Let us suppose that the agent `myFirstAgent` plays the role `First`, and is expected to communicate with an agent, named `mySecondAgent` (that we will define later), that plays the role `Second`.

3. Add this rule to the RULES section of your `myFirstAgent.clp` agent:

```
(defrule first1
 => (bind ?cid (newcid))
 (bind ?*addr* (read_addr "Second"))
 (if (eq nil ?*addr*) then
 (bind ?*addr* (fetch_addr "Second")))
 (bind ?content do_you_want_to_chat_with_me)
 (send (ACLMessage  (communicative-act PROPOSE)
 (receiver ?*addr*) (protocol "First")
 (conversation-id ?cid)
 (content ?content))) (assert (state stateFirst1 ?cid))
 (printout t crlf crlf "The first agent sent an
 PROPOSE " ?content " to an agent playing the
 Second role whose name is " ?*addr* crlf)
)
```

This rule is used when the agent is activated, and simply sends a message with performative PROPOSE and content ”do_you_want_to_chat_with_me” to any agent in the MAS that plays the role ”Second”. The information about the message sent, is printed on the standard output.

4. Open the `JessAgentSkeleton.clp` once again, and save it somewhere in you file system, with a name of your choice. Let us suppose that you saved the file as `C:\JessAgents\mySecondAgent.clp`.

5. Add this rule to the RULES section of your `mySecondAgent.clp` agent:

```
(defrule second1
 =>
 (wait_msg "PROPOSE" "First")
  (assert (state send (get_cid ?*msg*)))
 (retract-string "(message \"PROPOSE\" \"First\")")
 (bind ?current_content (get_content ?*msg*))
 (printout t crlf crlf "The second agent
 received a PROPOSE " ?current_content " from an
 agent playing the First role" crlf)
)

(defrule second2
 (state send ?cid)
 =>
 (send (ACLMessage (communicative-act
 ACCEPT_PROPOSAL)
 (sender Second) (receiver First)
 (conversation-id ?cid)
 (content yes_sure)))
  (printout t crlf crlf "The second agent sent
  an ACCEPT_PROPOSAL yes_sure to an agent playing
  the First role" crlf)
)
```

The first rule, named second1, allows the second agent to wait for a message from an agent playing the First role. Once the message has been received, the second agent changes its state into a state where it will answer to the proposal (`(state send ?cid)`), where ?cid is the conversation identifier of the current conversation. The second rule, named second2, is used when the agent is in the state `(state send ?cid)`. In this case the second agent answers to the first agent, with an ACLMessage with performative ACCEPT_PROPOSAL, and content "yes_sure".

2 Creating a Java stub for integrating the Jess agent into JADE

Since there are two roles played by agents in the MAS, you need to define two Java stubs.

1. Open the `JavaStubSkeleton.java` file that is contained in the `jessInJADE` directory with any text editor, and save it somewhere in you file system, with a name of your choice. Let us suppose that you save the file as `C:\JessAgents\First.java`, since it will define the stub for the agent that has the `myFirstAgent.clp` jess program, and that plays the `First` role.

2. Modify the code in the following way

```
...
/********** COMPLETE THE CODE HERE (name of
the class: substitute the string --write here
the name of the agent class-- with the name
of the class) ****************/
public class First extends jessInJADE.JessAg

...
      /********** COMPLETE THE CODE HERE
      (name of the role(s)
       played by the instances of this class:
       substitute the string --write here
       the role -- with the string that
       represents the name
       of the role) ****************/
       dfd.addProtocols("First");


      ...
      /********** COMPLETE THE CODE HERE
      (jessFile variable:
      substitute the string --write here
      the path to the jess file containing
      the jess code for this agent--
      with the string that represents the name
      of the file) ****************/
      String jessFile = "./myFirstAgent.clp";
```

3. Open once again the `JavaStubSkeleton.java` file that is contained in the `jessInJADE` directory with any text editor, and save it somewhere in you file system, with a name of your choice, for example `C:\JessAgents\Second.java`, since it will define the stub for the agent that has the `mySecondAgent.clp` jess program, and that plays the `Second` role.

4. Modify the code in the following way

```
...
/********** COMPLETE THE CODE HERE (name
of the class: substitute the string --write
here the name of the agent class--
with the name of the class) ***************/
public class Second extends jessInJADE.JessAg

...
      /********** COMPLETE THE CODE HERE
      (name of the role(s)
      played by the instances of this class:
      substitute the string --write here
      the role -- with
      the string that represents the name
      of the role) ***************/
      dfd.addProtocols("Second");


      ...
      /********** COMPLETE THE CODE HERE
      (jessFile variable:
      substitute the string --write here the path
      to the jess file containing the jess
      code for this agent--
      with the string that represents the name
      of the file) ***************/
      String jessFile = "./mySecondAgent.clp";
```

5. Move to the directory `C:\JessAgents\` and compile the Java files by typing the command `javac*.java`.

6. From the directory `C:\JessAgents\`, run JADE and integrate two agent instance into it, `f` of type First and `s` of type Second, by typing the command

```
java jade.Boot f:First s:Second.
```

7. You should obtain the following output, that demonstrates that the two agents successfully communicated:

```
The first agent sent an PROPOSE
do_you_want_to_chat_with_me to an agent
playing the Second role whose name is
s@klimt:1099/JADE

The second agent received a PROPOSE
do_you_want_to_chat_with_me
from an agent playing the First role

The second agent sent an ACCEPT_PROPOSAL
yes_sure to an agent playing the First role
```

8. The code of both Jess and Java components corresponding to the agents of this example can be found in the directory `DCaseLP\jessInJADE\Tutorial`

3 Making Jess and tuProlog agents interact

The interaction between a Jess and a tuProlog agent is completely transparent. Try to build a very simple tuProlog agent that behaves like the "myFirst" agent that we just implemented using Jess.

The tuProlog code of this agent, that we name `myFirstAgentInProlog.pl`, and that should be saved in the same directory where the other agents are (`C:\JessAgents\`), is the following one:

```
main :- send_msg.

main :- true.

send_msg :-
 sent(no),
 ask_address("Second",Ag),bound(Ag),
 rand_int(1000000,N),text_term(Txt,N),
 text_concat("fff",Txt,CID),
 pack(do_you_want_to_chat_with_me,STR),
 send("PROPOSE",STR,Ag,'"First"',CID),
 nl,nl,retract(sent(no)),
```

```
 write("The agent myFirstAgentInProlog has sent
 a PROPOSE do_you_want_to_chat_with_me to an agent
 playing the Second role whose name is "),write(Ag),
 nl,nl.

send_msg :- true.

sent(no) :- true.
```

This agent sends a "`PROPOSE do_you_want_to_chat_with_me`" message to an agent playing the Second role, in the same way as the Jess "myFirstAgent" did.

From the directory `C:\JessAgents\`, run JADE and integrate two agent instance into it, `fP`, the tuProlog agent that behaves according to the theory saved in the `myFirstAgentInProlog.pl`, and `s`, the Jess agent of type Second that we have previously built (type the command `java jade.Boot fP:tuPInJADE.JadeShell42P(myFirstAgentInProlog.pl) s:Second`).

The messages that should be printed by the two agents to the standard output are the following ones:

```
'The agent myFirstAgentInProlog has sent a PROPOSE
do_you_want_to_chat_with_me'' to an agent playing
the Second role whose name is ''s@klimt:1099/JADE'

The second agent received a PROPOSE
do_you_want_to_chat_with_me from an agent
playing the First role

The second agent sent an ACCEPT_PROPOSAL yes_sure
to an agent playing the First role
```

The Jess agent defined by the "mySecondAgent.clp" program, that we already know to be able to communicate with a Jess agent, has successfully communicated also with an agent written in tuProlog, without needing any change to its code!!!!

The code of the myFirstAgentInProlog agent can be found in the directory `DCaseLP\jessInJADE\Tutorial`