1. The jessInJADE Package

We call "Jess agents" those agents that are JADE agents, since they extend the JADE "Agent" class, but whose initial beliefs and behaviour are defined in the Jess language. The Jess code that characterises the agent's beliefs and behaviour must be defined in a Jess file that is read by the Java stub of the agent (the JavaStubSkeleton code), and is executed by it.

The jessInJADE package provides two classes, jessAg and jessBhv. The most important of them is the jessAg class, since every Jess agent must be defined by means of a Java class that extends the jessAg class. The class jessAg, in turn, extends the JADE Agent class by adding to it the capability to exchange messages with any JADE agent, while the class jessBhv defines the behaviour of a Jess agent, which is a cyclic behaviour.

In order to integrate a Jess piece of code into a JADE agent, we provide the skeleton of a Jess agent: the "JavaStubSkeleton" code, once opened in a text editor and completed in the two points marked by "/********** COMPLETE THE CODE HERE **********/", and saved with the name chosen for this class, behaves like a JADE agent that executes a piece of Jess code (namely, behaves like a "jess agent" in our terminology).

Note once again that the Java classes that can be found in this package allow the integration of a Jess piece of code into a Jade agent, but all the procedural behaviour of the agent must be defined in the Jess language, not in the Java one.

For this reason, we concentrate on the functions that the jessAg class provides, and that can be used from inside a Jess description of the behaviour of an agent. In the following, we explain which features are available from inside the Jess language, once integrated into JADE by means of the jessInJADE package. For more information on the Jess language, the reader should refer to the Manual of Jess.

1.1 Some useful information to know about Jess in JADE

- **this**: keeps a reference to the Java object that represents the JessAg agent currently under execution on the JADE platform (recall that a JessAg agent, is also a JADE agent), so that it is possible to use (from the Jess program loaded by the JessAg agent) all the functionalities available to an agent that executes in a JADE platform.
- In order to retrieve this reference, it is necessary to invoke the Jess **fetch** function. For example, with the **(bind ?agent (fetch this))** predicate call, the Jess variable **?agent** is unified with a reference to the JessAg agent.
- In order to print the local name of the agent running on the JADE platform, on the screen, the predicate call **(printout t "My name is " (?agent getLocalName))** can be used.
- Finally, **(bind ?content (?*msg* getContent))** allows to unify the **?**content Jess variable, with the content of the message maintained in the **?*msg*** global variable (that, in any moment, stores the last message received by the agent).

1.2 Jess functions available to the jessAg agent

- **send ACLMessage:** this function sends an ACLMessage to an agent running in a JADE platform. Its input is the fact **ACLMessage** whose template is predefined in any JessAg. The slots defined for the ACLMessage fact are the same as the one present in a JADE ACLMessage:

- communicative-act	**- protocol**
- sender	**- language**
- receiver (a multislot)	**- ontology**
- reply-with	**- content**
- in-reply-to	**- encoding**
- envelope	**- reply-to**
- conversation-id	**- reply-by.**

The **sender** slot is not actually read by the function since the agent's aid is known by the agent and directly inserted in the message to send. The slots that are necessary in order to be effectively able to send a message to an agent are: **communicative-act**, **receiver** and **content.** An example of how the function can be used is:

(send (ACLMessage (communicative-act **REQUEST**) (receiver **?addr**)
(protocol **"He"**) (conversation-id **?cid**) (content **?content**)))

The arguments of the slots **cannot be** the result of a function call but **must be a variable or a string**. The performative of the message must be specified in capital letters, the receiver must be specified by a string (or many strings separated by blank spaces) representing the GUID of an agent in JADE.

- **receive**: this function returns **a reference to the first ACLMessage** available in the mail box of the agent: it returns **nil** if no message is available.

- **newcid:** this function returns **a string** made by concatenating the local name of the agent in the JADE platform with a randomly generated number.

- **search_roles ?aid:** this function returns **the list of the roles** (if any) registered with the DF by the agent whose AID is given in input to the function, otherwise it returns **nil**. The list returned is a Jess multifield.

- **is_perf ?n ?perf:** this function takes in input a number and a string: it returns **TRUE** if the number represents the JADE performative given in input as a string, otherwise it returns **FALSE**.

- **is_role_in ?role ?aid:** this function takes in input a string representing a role and a JADE AID: it returns **TRUE** if the role is registered with the DF by the agent identified by the AID, otherwise it returns **FALSE**. If the agent has not registered any role with the DF, this function returns **FALSE** even if the role in input is nil.

- **get_aid_role ?role:** this function takes in input a string representing a role: it returns **the**

AID of the agent that has registered that role with the DF. If no agent registered the role with the DF, the function returns **nil**.

- **my_addr ():** this function returns the string representing the JADE GUID of the agent.

- **unquote (?str):** this function removes the quote symbols from the input string.

- get_cid (?msg): This function returns the conversation-id of the ACLMessage given in input.

- **get_perf (?msg):** This function returns the performative of the ACLMessage given in input.

- get_sender (?msg): This function returns the JADE AID of the agent that sent the ACLMessage given in input.

- get_prot (?msg): This function returns the content of the protocol slot of the ACLMessage given in input, in other words the role played by the agent that sent the message.

- get_content (?msg): This function returns the content of the ACLMessage given in input.

- wait_msg (?perf ?role): This function keeps reading the messages in the mail box of the agent until a message satisfies the following conditions: its performative is equal to the first input parameter and its protocol is equal to the second input parameter. The message is stored in the global variable ?*msg*. The ordered fact whose name is message and has two fields - the performative of the message read and the role of the agent that sent it - is asserted in the knowledge base belonging to the agent.

- cont (?perf ?role): This function returns TRUE if there are no messages in the mail box of the agent or if the message read is not the one expected, that is with the given performative and the given role as protocol

- **fetch_addr (?role):** This function keeps asking the DF for the address of the agent that registered the role given as first input parameter. It returns the string representing the GUID given by the DF.

- have_addr (?role): This function checks if the ordered fact named 'address' is present in the knowledge base with the given input as field and returns a multivariable containing the TRUE value followed by the address if it was found, otherwise the multivariable returned only contains the FALSE value

- read_addr (?role): This function gets the address of the given role from the ordered fact named 'address' in the knowledge base of the agent

2. Creating a Jess agent and integrating it into JADE

There are two pieces of software that the developer must edit and complete, in order to integrate a Jess agent into Jade: JavaStubSkeleton and JessAgentSkeleton, both available in the directory DCaseLP\jessInJADE.

2.1 JavaStubSkeleton

The JavaStubSkeleton is the one that integrates the Jess piece of code defined when editing and completing the JessAgentSkeleton, and integrates it into JADE. There are three points in the file that must be changed, namely those (in pink) that state the name of the class, the role(s) played by instances of the class, and the name of the Jess file containing the code of the Jess agent. Also, the name of the file must be changed (see comment in green) so that it becomes equal to the name given to the class, with ".java" extension (see the jessInJADE-Tutorial for an example).

```
/********** CHANGE THE NAME OF THIS FILE, SO THAT ITS NAME BECOMES EQUAL TO THE NAME THAT YOU ASSIGN
TO THE CLASS
(SEE "COMPLETE THE CODE HERE" BELOW), with .java extension ********************/
import jessInJADE.*;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import jess.Jesp;
import jess.JessException;
import jess.Rete;

/********** 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 --write here the name of the agent class-- extends jessInJADE.JessAg
{ protected void setup()
  { try
    { super.setup();
      DFAgentDescription dfd = new DFAgentDescription();
      /********** 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(--write here the role --);
      DFService.register(this,dfd);
      eng.executeCommand(ACLTempl());
      JSend sen = new JSend();
      eng.addUserfunction(sen);
      eng.store("this",this);
      eng.executeCommand("(import jade.domain.DFService)");
      eng.executeCommand("(import jade.domain.FIPAAgentMan"
                    +"agement.DFAgentDescription)");
      eng.executeCommand("(import jade.core.AID)");
      eng.executeCommand("(import jade.lang.acl.*)");
      eng.executeCommand("(import jade.util.leap.*)");
      eng.executeCommand("(import java.util.Iterator)");

      /* The receive function returns a reference to the first message
        available in the mail box of the agent: it returns the Jess
        nil value if no message is available. */

      String str = "(deffunction receive () (bind ?msg ((fetch this) "
              +"receive)) (if (eq nil ?msg) then (return nil) else"
              +" (return ?msg)))";
      eng.executeCommand(str);
```

```
/* The newcid function returns a string made by concatenating the
   local name of the agent in the JADE platform with a randomly
   generated number. */

str = "(deffunction newcid () (return (str-cat ((fetch this)"
    +" getLocalName) (random))))";
eng.executeCommand(str);

/* The search_roles function returns a list of the roles (if any)
   registered to the DF by the agent whose AID is given in input
   to the function. */

str = "(deffunction search_roles (?aid) (bind ?dfd (new jade.domain."
    +"FIPAAgentManagement.DFAgentDescription)) (?dfd setName ?aid)"
    +" (bind $?res (call jade.domain.DFService search (fetch this)"
    +" ?dfd)) (if (not (eq $?res nil)) then (bind ?res0 (nth$ 1 $?"
    +"res)) (bind ?list (?res0 getAllProtocols)) (bind $?arr nil) "
    +"(if (eq FALSE (?list hasNext)) then (return nil) else (while (?li"
    +"st hasNext) do (bind $?arr (create$ (?list next) $?arr))) "
    +"(bind $?fin (subseq$ $?arr 1 (- (length$ $?arr) 1))) (return"
    +" $?fin)) else (return nil)))";
eng.executeCommand(str);

/* The is_perf function takes in input a number and a string: it
   returns TRUE if the number represents the JADE performative
   given by the string. */

str = "(deffunction is_perf (?n ?perf) (if (eq ?n nil) then "
    +"(return FALSE) else (if (eq ?perf nil) then (return FALSE) "
    +"else (if ((call jade.lang.acl.ACLMessage getPerformative ?n)"
    +" equals (?perf trim)) then (return TRUE) else (return FALSE)"
    +"))))";
eng.executeCommand(str);

/* The is_role_in function takes in input a string representing a
   role and a JADE AID: it returns TRUE if the role is registered
   with the DF by the agent represented by the AID.
   If the agent has not registered any role with the DF, this function
   returns FALSE even if the role is the Jess nil value. */

str = "(deffunction is_role_in (?rol ?aid) (if (eq ?rol nil) then "
    +"(return FALSE) else (if (eq ?aid nil) then (return FALSE) "
    +"else (bind $?list (search_roles ?aid)) (if (eq $?list nil) "
    +"then (return FALSE) else (if (member$ ?rol $?list) then "
    +"(return TRUE) else (return FALSE))))))";
eng.executeCommand(str);

/* The get_aid_role function takes in input a string representing a
   role: it returns the AID of the agent that has registered the role
   with the DF.
   If no agent registered the role with the DF, the function returns
   the Jess nil value. */

str = "(deffunction get_aid_role (?rol) (if (eq ?rol nil) then (retu"
    +"rn nil)) (bind ?dfd (new jade.domain.FIPAAgentManagement.DFAge"
    +"ntDescription)) (?dfd addProtocols ?rol) (bind $?res (call jad"
    +"e.domain.DFService search (fetch this) ?dfd)) (if (not (eq $?r"
    +"es nil)) then (if (>= (length$ $?res) 1) then (bind ?res0 (nth"
    +"$ 1 $?res)) (return (?res0 getName)) else (return nil)) else ("
    +"return nil)))";
eng.executeCommand(str);

/********** 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 = "--write here the path to the jess file containing the jess code for this agent--";
```

```
      FileReader fr = null;
      try
      { fr = new FileReader(jessFile);}
      catch(FileNotFoundException exc)
      { title = "ERROR CAUSED BY AGENT "+getLocalName();
        errTxt = " The main program of the Jess agent cannot"
               +" be found.";
        write_msg(title,errTxt);
      }
      Jesp j = new Jesp(fr,eng);
      j.parse(false);
      addBehaviour(new JessBhv(this));
    }
    catch (Exception err)
    { title = " ERROR CAUSED BY AGENT "+getLocalName();
      errTxt = "";
      if(err.getClass().getName().equals("jess.JessException"))
      { errTxt = "Error: "+((JessException) err).getMessage()
               +" While parsing: "+((JessException) err).getProgramText();
      }
      else errTxt = err.getMessage();
      write_msg(title,errTxt);
    }
  }
}
```

2.2 JessAgentSkeleton

This file contains other functions available to the Jess agent; it must be completed with both the behaviour and the initial facts of the agent in the places marked by pink comments (see the jessInJADE-Tutorial for an example, and refer to the Jess manual for learning how to write Jess programs).

```
(defglobal ?*addr* = nil)
;This global variable is used to store the address of an agent.

(defglobal ?*msg* = nil)
;This global variable is used to store the last message read
;by the agent.

(deffunction my_addr () "It returns the string representing
 the JADE GUID of the agent."
 (return ((fetch this) getName)))

(deffunction unquote (?str) "It removes the quote symbols from
 the input string."
 (return (?str substring 1 (- (?str length) 1))))

(deffunction get_cid (?msg) "It returns the conversation-id of the
 ACLMessage given in input."
 (return (unquote (?msg getConversationId))))

(deffunction get_perf (?msg) "It returns the performative of the
 ACLMessage given in input."
 (return (call jade.lang.acl.ACLMessage getPerformative
 (?msg getPerformative))))

(deffunction get_sender (?msg) "It returns the JADE AID of the agent
 that sent the ACLMessage given in input."
 (return (?msg getSender)))

(deffunction get_prot (?msg) "It returns the content of the protocol
```

```
 slot of the ACLMessage given in input, in other words the role
 played by the agent that sent the message."
 (return (?msg getProtocol)))

(deffunction get_content (?msg) "It returns the content of the
 ACLMessage given in input."
(return (?msg getContent)))

(deffunction wait_msg (?perf ?role) "It keeps reading the messages in
 the mail box of the agent until a message satisfies the following
 conditions: its performative is equal to the first input parameter
 and its protocol is equal to the second input parameter. The message
 is stored in the global variable ?*msg*. The ordered fact whose name
 is message and has two fields - the performative of the message read
 and the role of the agent that sent it - is asserted in the knowledge
 base belonging to the agent."
 (bind ?*msg* (receive))
 (while (eq TRUE (cont ?perf ?role)) do (bind ?*msg* (receive)))
 (assert (message ?perf ?role)))

(deffunction cont (?perf ?role) "It returns TRUE if there are no
 messages in the mail box of the agent or if the message read is not
 the one expected, that is with the given performative and the given
 role as protocol."
 (if (eq nil ?*msg*) then (return TRUE) else
 (if (eq ?perf (get_perf ?*msg*)) then
 (bind ?roleRead (get_prot ?*msg*))
 (bind ?fstChar (new java.lang.String (?roleRead charAt 0)))
 (if (eq 0 (?fstChar compareToIgnoreCase "\""))
  then (bind ?roleRead (unquote ?roleRead)))
 (if (eq ?role ?roleRead) then (return FALSE) else (return TRUE))
  else (return TRUE))))

(deffunction fetch_addr (?role) "It keeps asking the DF for the
 address of the agent that registered the role given as first input
 parameter. It returns the string representing the GUID given by the DF."
 (bind ?aid (get_aid_role ?role))
 (while (eq nil ?aid) do (bind ?aid (get_aid_role ?role)))
 (return (?aid getName)))

(defquery addr (declare (variables ?x)) (address ?x))

(deffunction have_addr (?role) "It checks if the ordered fact named
 'address' is present in the knowledge base with the given input
 as field and returns a multivariable containing the TRUE value
 followed by the address if it was found, otherwise the multivariable
 returned only contains the FALSE value."
 (bind ?elem (run-query addr ?role))
 (if (?elem hasNext) then (bind $?el (((?elem next) fact 1) get 0))
 (if (> (length$ $?el) 0) then (bind ?adr (nth$ 1 $?el))
 (return (create$ TRUE ?adr)) else (return (create$ FALSE)))
 else (return (create$ FALSE))))

(deffunction read_addr (?role) "It gets the address of the
 given role from the ordered fact named 'address' in the knowledge
 base of the agent."
 (bind $?res (have_addr ?role))
 (if (> (length$ $?res) 1) then (return (nth$ 2 $?res))
 else (return nil)))

; ************************** RULES ****************************

;add the rules that govern the behaviour of your agent here

; ************************** MAIN *****************************

;add the rules initial beliefs of your agent here
(reset)
```

2.3 Integrating a Jess Agent into JADE

Let us suppose that the JessAgentSkeleton has been completed and has been saved with name **jessFoo.clp**, in the directory **dirFoo**, and the JavaStubSkeleton has been edited, completed so to define the javaFoo class and to integrate the jessFoo.clp program, and saved with name **javaFoo.java** in the directory dirFoo.

The MAS developer must now compile the javaFoo.java file by calling **javac javaFoo.java** from the dirFoo directory.

At this point, the javaFoo agent is ready to be integrated into JADE by calling
java jade.Boot javaFooInstance:javaFoo

In this way, an instance of the **javaFoo** class named **javaFooInstance**, that behaves in the way prescribed by the **jessFoo.clp** Jess program, has been created and integrated into the JADE platform!

2.4 Making Jess and tuProlog agents interact

There is nothing special to do in order to make Jess and tuProlog agents communicate, since the communication works, no matter the language in which agents are implemented. The jessInJADE tutorial provides an example where a tuProlog agent interacts with a Jess agent, while the tuPInJADE tutorial provides an example of the integration of tuProlog and Jade agents.