	Document:	IMAS practical work
	Course:	ETSE-URV, 2020-21

Release	**Date**	**Description**
1.0	19-SEP-2020	Functional analysis
1.1	05-OCT-2020	Technical details

Table of contents

1. General overview

The main goal of the practical work of this course is to create a **fuzzy agent-based decision support system (FA-DSS)**, which permits to decide about a certain inputs using fuzzy inference engines embedded in particular agents.

The system performs two main actions, to prepare the fuzzy agent-based system and to handle the requests from users in order. The initialisation is done according the requirements sent by the user. Once the agent-based is ready to work, the user can use it in order to evaluate different entries. As we shall explain later, one of the main goals is to create an open system and scalable system that could be used under different circumstances, and taking into account different strategies and topologies.

To illustrate the problem, imagine that we have a lawn watering system. Taking into account some weather variables, the system is able to specify the period of watering. The input parameters could be the soil humidity and the outside temperature, and the output is the watering duration. The inference engine activates the fuzzy rules contained in its knowledge base whose premises are under fuzzy inputs, and then it aggregates all the conclusions of these activated rules. Finally, the inference engine transforms the fuzzy decision to the domain of the output. In this system, there are different elements to take into account such as the definition of all variables (input and output), the knowledge base of rules, the function to evaluate the rules (t-norm), and the function to aggregate the consequents (t-conorm).

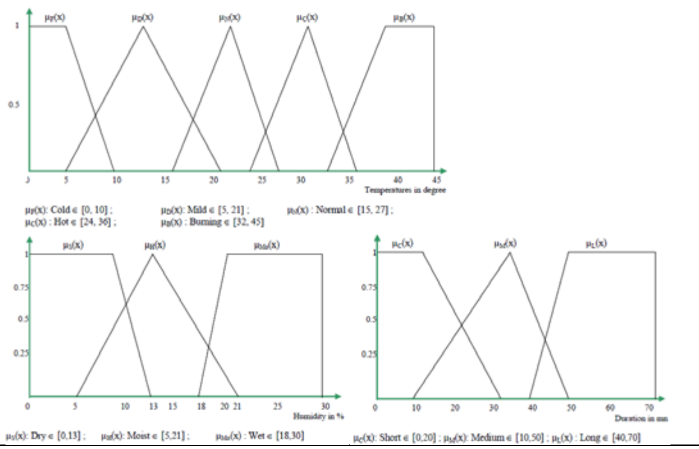
Variables	humidity x temperature → duration_period
Membership functions	
Rules	IF temperature is burning AND humidity is moist THEN the watering duration is average
Inference	Mamdani (t-norm minimum and t-conorm maximum)
Defuzzyfication function	Centre of area (CoG)
Decision	Input: (humidity, temperature) = (18, 38) Output: duration_period = 55 mn

Table 1: Elements of the fuzzy agents

Table 1 depicts the main elements included in a fuzzy system. The first element consists in the variables, distinguishing input and output variables. The next elements are the fuzzy sets associated to each variable which implement the semantic of the variable. Table 1 shows different examples using triagonal (e.g., $\mu_D(x)$) and trapezoidal functions (e.g., $\mu_P(x)$). The next element to define in any fuzzy inference engine is the knowledge base set of rules. This set can combine conjunction operators, disjunction, and negation operators.

The inference mechanism is defined by the t-norm and t-conorm. Typically, most of the systems use the pair defined by Mandami using the minimum of all inputs as t-norm, and maximum of all inputs as t-conorm. When the final consequent has been evaluated, the decision should create the final decision using the domain of the output variable using one of the available defuzzyfication functions, such as the centre of area.

2. Agents

As we previously introduced, the system include three different agents named *user* agent, *manager* agent and *fuzzy* agent. Figure 1 shows the interactions between these agents.

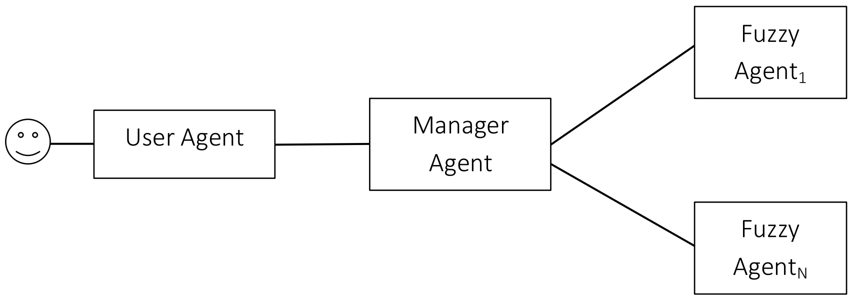


Figure 1: Agents of the system

The user agent is able to interact with the user in order to deal with the two main actions, starting the multiagent system, and answering the requests received from a user.

The manager agent is the gateway between the user agent and the set of fuzzy agents. This agent creates the fuzzy agents as requested, and then, it manages the requests received from the user agent and handled by the fuzzy agents.

The last type of agents is the fuzzy agents. These agents are created when required by the manager agent and embed a fuzzy inference engine. Each fuzzy agent is defined by a fuzzy set, a set of rules and a particular mechanism to handle with data and infer the final response. These agents represent a set of available resources that solve a problem.

3. Fuzzy-logic technical details

In the following section, some technical aspects about the implementation of the fuzzy agents will be explained. This course, all fuzzy-related aspects will be implemented using a third party library called jFuzzyLogic[1]. This is an open-source library that permits to create a fuzzy rule inference engine. The main goal of the practical work is to create an agent-based system, and this library offers all the issues involved with the creation and management of fuzzy logic.

[1] URL: http://jfuzzylogic.sourceforge.net/html/index.html

Fuzzy Control Language

Fuzzy control language (FCL) is a specification that permits to define fuzzy control system[2]. As figure 2 shows, any FCL file contains the elements that one fuzzy agent requires to decide according further inputs.

```
FUNCTION_BLOCK
VAR_INPUT
   <variable name> REAL; (* RANGE(<variable minimum value> .. <variable maximum value>) *)
END_VAR
VAR_OUTPUT
   <variable name> REAL; (* RANGE(<variable minimum value> .. <variable maximum value>) *)
END_VAR
FUZZIFY <variable name>
   TERM <term (or set) name> := <points that make up the term> ;
END_FUZZIFY
DEFUZZIFY valve
   METHOD: <defuzzification method>;
END_DEFUZZIFY
RULEBLOCK <ruleblock name>
   <operator>:<algorithm>;
   ACCUM:<accumulation method>;
   RULE <rule number>: IF <condition> THEN <conclusion>;
END_RULEBLOCK
END_FUNCTION_BLOCK
```

Figure 2: FCL details

[2] The complete FCL specification can be found here: http://jfuzzylogic.sourceforge.net/doc/iec_1131_7_cd1.pdf

Fuzzy inference system

Once an agent has been configured with a particular FCL specification, the agent is able to handle with particular entries. This inference is performed by the fuzzy inference system[3] (FIS) created by jFuzzyLogic. It permits to specify the values to evaluate, run the fuzzy system taking into account the rules and particular settings to aggregate the data, and at the end, it offers mechanisms for getting the particular results (fuzzy or defuzzified).

```
// the agent will receive the name of the FCL to load
// fis is the fuzzy inference System
FIS fis = FIS.load(fileName,true);
...
// values received
fis.setVariable("variableA", value_variable_A);
fis.setVariable("variableB", value_variable_B);
fis.evaluate();
fis.getVariable("output").getLatestDefuzzifiedValue();
```

Figure 3: FIS main operations

```
domain1
1,2
3,4
0,10
```

Figure 4: inputs from users

As figure 3 depicts, first of all, all agents require loading the configuration file, and then, they are able to apply this FIS with the incoming inputs. It's important to note that one agent will handle one particular FIS. In addition, figure 4 shows how users will transmit their requests through a particular file. This file provides the name of the domain of work, and then, a sequence of input values. As a result, the manager agent will transmit to the user agent the answers per each input.

4. FA-DSS system architecture

In the following, all steps of the FDSS will be explained in detail.

- The first step is to create the two first (and required) agents, one user agent and one manager agent. The user agent is ready to receive inputs from the user (initialisation or decision). Meanwhile, the manager agent is waiting for the first message from its user agent.
- The user agent permits to specify the action to perform from a real user (human): *initialisation* (*I*) or *evaluation* of a set of inputs given (*D*). In both cases, the input also specifies the name of a file where settings (see appendix 1) or data are included.
  - As a suggestion, these actions can be formatted such as `<action>_<file>`. For instance, the command `I_configuration1.txt` permits to *initialise* the system according the configuration given in *configuration1.txt* file. The command

[3] For more information, please visit http://jfuzzylogic.sourceforge.net/html/java.html

`D_requests_tipper.txt` permits to *evaluate* (or *decide* outputs) according the input pairs given in *requests_tipper.txt* file.

- At first, the human through his/her user interface, orders the action `I` (e.g. `I_configuration1.txt`). The user agent begins the transaction with the manager agent attaching the details of the simulation.
- The manager agent receives the *I*-order with the name of the file where settings are. It reads this file, creates the required fuzzy agents and finally, replies to the user agent when all agents are ready to work. These details contain the number of fuzzy agents to create, and the required FCL for each one.
- When a fuzzy agent is created, it receives the name of the FCL to read (see the FCL example on appendix 2). At this point, the fuzzy agent is able to create its particular FIS, and waits for further messages (requests).
- Now, the human, through his/her user interface, orders the action *D* (e.g. `D_request_tipper.txt`). This order is accompanied by the name of a file with a set of inputs to be evaluated (see file format). The user agent begins the transaction with the manager agent.
- The manager receives the *D*-order from its user agent. First of all, the manager reads the file with data to be valuated, and then starts a coordination mechanism with the fuzzy agents in order to create the final decision. The manager agent sends the set of rows to evaluate to the *required* set of fuzzy agents.
- The coordination mechanism between the manager and classifiers ends up with the composition of the final decision for each row to be decided (aggregation method specified in the configuration file).
- When all answers and decisions have been aggregated, the manager agent creates a new file with all these results, and the name of this new file is enclosed with the final response to the user agent.
    - The format and content of this file is open to enclose all information useful to understand and explain the results given by the fuzzy set of agents
- At the end, the user receives the final message from the manager agent informing that the process of decision has been finished and displaying where the results can be found.

- At this point, the user can request another *I*-order or another *D*-order by changing the provided files. If the user requests another initialisation, it could require the creation or not of new fuzzy agents to deal with that settings.

5. Minimum requirements

The FA-DSS is open to include different design decisions.

In the following, a list of minimum requirements to accept the practical work is listed:

- The system should be able to interact with a human user (through keyword) in order to specify which action wants to run, initialise (I) or decide (D).
- The FA-DSS has to be configured through a configuration file. A set of fuzzy agents should be created as required.
- It is required to run the FA-DSS with the basic and medium configurations provided in appendix 1.
- The FA-DSS should implement at least one aggregation method (e.g., average).
- The FA-DSS should provide a coordination mechanism between the manager and the classifiers in order to compose the final decision in the prediction stage.

6. Useful links and recommended readings

Official GIT repository: https://gitlab.com/disern/IMAS-URV-2020_21

Information about jFuzzyLogic:

- Example of FCL, http://jfuzzylogic.sourceforge.net/html/example_fcl.html
- Overview of the architecture of jFuzzyLogic, https://sci2s.ugr.es/sites/default/files/ficherosPublicaciones/1644_CingolaniAlcala_IJCIS_jFuzzyLogic.pdf.
- Set of examples showing functions of jFuzzyLogic: https://www.programcreek.com/java-api-examples/?api=net.sourceforge.jFuzzyLogic.FIS

The configuration files could be written using an XML-based notation or using java-like properties:

- In this link, you can find information about how to handle properties' files, https://www.mkyong.com/java/java-properties-file-examples/.
- There are some alternatives to handle XML files. In previous courses of IMAS, the JAXB was used (https://docs.oracle.com/javaee/7/api/javax/xml/bind/JAXBContext.html).

As you work in a team, it is useful to use well-known and widely-used tools such as:

- As IDE, Eclipse[4] is a good alternative. Any version since Oxygen would be valid.
    - jFuzzyLogic offers a plugin to colour FCLs files, see all membership functions as well as code completion.
- JDK 1.6[5] or newer.
- It is recommended the use of Maven (https://maven.apache.org/) to create the workspace. JADE (https://jade.tilab.com/developers/maven/) are supported.
- It is highly recommended to use a logging library to trace all the entities of the FA-DSS. In the following, two alternatives are listed, the Apache Log4j[6] (or its new release log4j 2) and the Java Logger[7].

It is **mandatory** to track and maintain the project on GIT server. You can use GitHub[8], GitLab[9] or BitBucket[10].

You can find the skeletons of past practical works at https://gitlab.com/disern/imas-skeleton-18-19, https://gitlab.com/FadiHassan/imasproject, or https://github.com/jordiae/IMAS-MAI. These examples show the dynamic creation of agents, handle with communication behaviours, to read XML-based configuration files, to pass information between agents using serialized objects, as well as to organize the packages in these kinds of projects.

[4] URL: https://www.eclipse.org/
[5] URL: https://www.oracle.com/java/technologies/javase-java-archive-javase6-downloads.html
[6] URL: https://logging.apache.org/log4j/2.x/
[7] URL: https://examples.javacodegeeks.com/core-java/util/logging/java-util-logging-example/
[8] URL: https://github.com/github
[9] URL: https://about.gitlab.com/
[10] URL: https://bitbucket.org/

7. Appendix 1: configuration files

A configuration file contains all the parameters required in any simulation. It could be written using an XML-notation or using a properties-based notation. Both alternatives are accepted.

Basic simulation

This simulation is defined on **configuration1.txt**. It uses 3 fuzzy agents with 2 different configurations under the tipper domain.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SimulationSettings>
   <title>IMAS_basic_simulation</title>
   <application>tipper</application>
   <fuzzyagents>3</fuzzyagents >
   <fuzzySettings>fcl1,fcl2,fcl2</fuzzySettings>
   <aggregation>average</aggregation>
</SimulationSettings>
```

```
urv.imas.title=IMAS_basic_simulation
urv.imas.application=tipper
urv.imas.fuzzyagents=3
urv.imas.fuzzySettings=fcl1,fcl2,fcl2
urv.imas.aggregation=average
```

Medium simulation

This simulation is defined on **configuration2.txt** and it uses 4 fuzzy agents using 4 different fuzzy configurations (fcl3, fcl4, fcl5, fcl6) about the quality of service domain.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SimulationSettings>
   <title>IMAS_medium_simulation</title>
   <application>qualityservice</application>
   <fuzzyagents>4</fuzzyagents >
   <fuzzySettings>fcl3,fcl4,fcl5,fcl6</fuzzySettings>
   <aggregation>average</aggregation>
</SimulationSettings>
```

```
urv.imas.title=IMAS_medium_simulation
urv.imas.application=qualityservice
urv.imas.fuzzyagents=4
urv.imas.trainingSettings=fcl3,fcl4,fcl5,fcl6
urv.imas.aggregation=average
```

Premium simulation

It permits the FA-DSS dealing with both simulations at the same time. The user requests two different initialisations, and combines requests from both domains. This simulation permits testing all coordination aspects among all agents as well as permits testing all synchronisation aspects between the manager and fuzzy agents.

8. Appendix 2: FCL files

The basic and medium simulations require configuring different agents using different FCL configurations. These files should be created based on two existing examples.

The basic simulation requires two different files about tipper domain (named fcl1 and fcl2). These files should be based on **tipper.fcl** that is already put on the repository (see code below), by changing the membership functions (in red) as well as the rules contained in the knowledge base (in blue).

```
/*
        Example: A tip calculation FIS (fuzzy inference system)
        Calculates tip based on 'servie' and 'food'
                                        Pablo Cingolani pcingola@users.sourceforge.net
*/

FUNCTION_BLOCK tipper   // Block definition (there may be more than one block per file)

VAR_INPUT                       // Define input variables
        service : REAL;
        food : REAL;
END_VAR

VAR_OUTPUT                      // Define output variable
        tip : REAL;
END_VAR

FUZZIFY service         // Fuzzify input variable 'service': {'poor', 'good' , 'excellent'}
        TERM poor := (0, 1) (4, 0) ;
        TERM good := (1, 0) (4,1) (6,1) (9,0);
        TERM excellent := (6, 0) (9, 1);
END_FUZZIFY

FUZZIFY food                    // Fuzzify input variable 'food': { 'rancid', 'delicious' }
        TERM rancid := (0, 1) (1, 1) (3,0) ;
        TERM delicious := (7,0) (9,1);
END_FUZZIFY

DEFUZZIFY tip   // Defzzzify output variable 'tip' : {'cheap', 'average', 'generous' }
        TERM cheap := (0,0) (5,1) (10,0);
        TERM average := (10,0) (15,1) (20,0);
        TERM generous := (20,0) (25,1) (30,0);
        METHOD : COG;           // Use 'Center Of Gravity' defuzzification method
        DEFAULT := 0;           // Default value is 0 (if no rule activates defuzzifier)
END_DEFUZZIFY

RULEBLOCK No1
        AND : MIN;                      // Use 'min' for 'and' (also implicit use 'max' for
'or' to fulfill DeMorgan's Law)
        ACT : MIN;                      // Use 'min' activation method
        ACCU : MAX;                     // Use 'max' accumulation method

        RULE 1 : IF service IS poor OR food IS rancid THEN tip IS cheap;
        RULE 2 : IF service IS good THEN tip IS average;
        RULE 3 : IF service IS excellent AND food IS delicious THEN tip IS generous;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

The medium simulation requires four different FCL files about quality of service domain (named fcl3, fcl4, fcl5 and fcl6). These files should be created based on the provided file named **MamdaniQoSManyRules.fcl**, by changing the membership functions and/or by changing the rules.

9. Appendix 3: data files

Data files contain the requests about decisions that users transmit to the AF-DSS. The format is as follows: the first row contains the name of the domain where it applies, and then a sequence of inputs following the rules of the input variables. It is assumed that all input values are well formatted and inside the range of the variables.

The repository contains two examples named **requests_tipper.txt** and **requests_quality.txt** for dealing with the tipper and quality of service domains respectively. You can use these files for testing your practical works as well as to analyse the performance and extract statistics of the global behaviour.