SCHOOL OF SCIENCE AND ENGINEERING

PRECISION AGRICULTURE: MODELING AND SIMULATION

EGR 4402: Capstone Design

Spring 2018

Omar Bahri

Dr. Asmaa Mourhir
PRECISION AGRICULTURE: MODELING AND SIMULATION

Capstone Report

Student Statement:
I declare on my word of honor that I have applied ethics to the design process and in the selection of the final proposed design. And that, I have held the safety of the public to be paramount, and has addressed this in the presented design whenever may be applicable.

[Signature]
Omar Bahri

Approved by the Supervisor:

[Signature]  April 16, 2018
Dr. Asmaa Mourhir
ABSTRACT

In conventional agriculture, a wide variety of decisions are made with high uncertainty. Pesticides, fertilizers and irrigation water recommendations are made in a very generic manner, and do not consider the intra-field variability of parameters that can affect crop yield. This induces serious economic and environmental implications. Precision Agriculture aims to develop a sustainable agriculture that optimizes crop growth decisions by considering field variability and site-specific parameters. Research in this context is faced with a large set of problems. On the one hand, the absence of observations data calls for alternatives to empirical, mathematical models. On the other hand, crop growth management is characterized by its dynamism and by the involvement of different parts and entities; which makes it an inherently complex problem and difficult to model. This work aims to develop a decision support system for sustainable agriculture. The proposed platform aims to provide crop and site specific recommendations for the use of pesticides and fertilizers, irrigation duration and other. The complexity of the problem is overcome by the adoption of the Agent Based Modeling paradigm, and the scarcity of data is overcome by the use of soft computing techniques for modeling and simulation. An architecture of the platform is proposed and a case study is implemented and tested on a Fuzzy Inference System for the prediction of tomato crop irrigation duration. The results show that up to 37% savings in water consumption could be achieved compared to conventional programmed irrigation systems.
# Table of Contents

List of Figures .................................................................................................................. 7

List of Abbreviations ......................................................................................................... 9

1. Introduction.................................................................................................................... 10

2. Literature Background .................................................................................................. 12

   2.1. Multi-Agent Systems............................................................................................... 12

      2.1.1. Overview ........................................................................................................... 12

      2.1.2. Java Agent DEvelopment Framework ................................................................. 14

         2.1.2.1. Overview ....................................................................................................... 14

         2.1.2.2. Architecture ................................................................................................. 16

         2.1.2.3. Message Transport ....................................................................................... 17

         2.1.2.4. Agent Communication .................................................................................. 18

         2.1.2.5. Agent Execution ............................................................................................ 19

   2.2. Fuzzy Inference Systems ......................................................................................... 21

      2.2.1. Overview ............................................................................................................. 21

      2.2.2. Fuzzy Logic Formalism ....................................................................................... 22

         2.2.2.1. Membership Functions .................................................................................. 22

         2.2.2.2. Fuzzy Rules .................................................................................................... 22

         2.2.2.3. Fuzzy Inference ............................................................................................. 23

   2.2.3. JFuzzyLogic Library .............................................................................................. 25
3. Related Work .................................................................................................................. 28
   3.1. Multi-Agent Systems in Precision Agriculture ....................................................... 28
   3.2. Fuzzy Inference Systems in Precision Agriculture ................................................ 30
4. Platform Proposed ......................................................................................................... 31
   4.1. Platform Design ....................................................................................................... 31
       4.1.1. Physical Layer ................................................................................................. 31
       4.1.2. JADE Platform .............................................................................................. 32
   4.2. Usage Scenarios ....................................................................................................... 35
       4.2.1. New Inference Agent Scenario ....................................................................... 35
       4.2.2. New Site Scenario ......................................................................................... 36
       4.2.3. Site Monitoring Scenario ............................................................................... 38
5. Implementation - Case Study ........................................................................................ 39
   5.1. Platform Development ............................................................................................ 39
   5.2. Fuzzy Inference System Description ...................................................................... 40
   5.3. Platform Testing ...................................................................................................... 44
   5.4. Database Design ..................................................................................................... 45
   5.5. Inference Results .................................................................................................... 46
6. STEEPLE Analysis .......................................................................................................... 51
   6.1. Societal Factor ........................................................................................................ 51
   6.2. Technical Factor ...................................................................................................... 51
List of Figures

Figure 2.1.1.1: FIPA Reference Model .............................................................. 13
Figure 2.1.2.1: JADE Architecture ................................................................. 16
Figure 2.1.2.3.1: JADE Messaging Architecture ........................................... 18
Figure 2.1.2.5.1: JADE Support for Content Languages and Ontologies ........ 19
Figure 2.1.4.1: Agent Thread Path of Execution ............................................. 20
Figure 2.2.1.1: Trapezoidal MF Example ...................................................... 22
Figure 2.2.3.1: Fuzzification example ............................................................. 23
Figure 2.2.3.2: Rule Implication Methods ....................................................... 24
Figure 2.2.3.3: Max Aggregation .................................................................... 24
Figure 2.2.3.4: COG Defuzzification .............................................................. 25
Figure 2.3.1: Tipper FCL – Variables Definition ............................................ 26
Figure 2.3.2: Tipper FCL – Fuzzification ........................................................ 26
Figure 2.3.3: Tipper FCL – Defuzzification .................................................... 27
Figure 2.3.4: Tipper FCL – Fuzzy Rules .......................................................... 27
Figure 4.1.1: Platform Architecture ............................................................... 31
Figure 4.2.1.1: New Inference Agent Diagram .............................................. 36
Figure 4.2.2.1: New Site Use Case Diagram ................................................... 37
Figure 4.2.3.1: Site Monitoring Sequence Diagram ...................................... 38
Figure 5.1.1: Concepts of the Ontology .......................................................... 40
Figure 5.2.1: Soil Moisture Membership Functions ...................................... 41
Figure 5.2.2: Temperature Membership Functions ...................................... 41
Figure 5.2.3: Solar Radiation Membership Functions .................................... 42
Figure 5.2.4: Irrigation Time Membership Functions .................................... 42
Figure 5.2.5: Tomato Irrigation FCL .............................................................. 43
Figure 5.3.1: Platform Execution ................................................................. 44
Figure 5.3.2: Sniffer Agent: Messages Exchanged ........................................ 45
Figure 5.4.1: Database Design - ERD ............................................................. 46
Figure 5.5.1: Results Ordered by moisture_value ......................................... 47
Figure 5.5.2: Results Ordered by irrigation_time ......................................... 48
Figure 5.5.3: Temperature, Moisture and Irrigation Duration Surface Plot…………………………49
Figure 5.5.4: Solar Radiation, Moisture and Irrigation Duration Surface Plot…………………………49
Figure 5.5.5: Water Consumption Comparison…………………………………………………………50
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM</td>
<td>Agent-Based Modeling</td>
</tr>
<tr>
<td>ACC</td>
<td>Agent Communication Channel</td>
</tr>
<tr>
<td>AMS</td>
<td>Agent Management System</td>
</tr>
<tr>
<td>AOP</td>
<td>Agent-Oriented Programming</td>
</tr>
<tr>
<td>COG</td>
<td>Center Of Gravity</td>
</tr>
<tr>
<td>DF</td>
<td>Directory Facilitator</td>
</tr>
<tr>
<td>FCL</td>
<td>Fuzzy Control Language</td>
</tr>
<tr>
<td>FCM</td>
<td>Fuzzy Cognitive Maps</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundation for Intelligent Physical Agents</td>
</tr>
<tr>
<td>FIS</td>
<td>Fuzzy Inference System</td>
</tr>
<tr>
<td>FLC</td>
<td>Fuzzy Logic Controller</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IMTP</td>
<td>Internal Message Transport Protocol</td>
</tr>
<tr>
<td>IOT</td>
<td>Internet Of Things</td>
</tr>
<tr>
<td>JADE</td>
<td>Java Agent DEvelopment Framework</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent Systems</td>
</tr>
<tr>
<td>MIDP</td>
<td>Mobile Information Device Profile</td>
</tr>
<tr>
<td>MTP</td>
<td>Message Transport Protocol</td>
</tr>
<tr>
<td>MTS</td>
<td>Message Transport Service</td>
</tr>
<tr>
<td>PA</td>
<td>Precision Agriculture</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>WSAN</td>
<td>Wireless Sensor and Actuator Network</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
</tbody>
</table>
1. Introduction

In conventional farming, agricultural fields are treated uniformly. The intra-field variability in crops is not considered and the recommendations for the use of fertilizers, pesticides and irrigation water are generic. Such practices prevent the crop from achieving optimal yields, and thus incur negative impact on the farmers from an economic perspective. Moreover, the overuse of the three mentioned parameters results in serious implications for the environment. On one hand, pesticides and fertilizers can be carried by the wind to farther areas, affecting public health. They also represent a threat to streams and ground water, as they are easily transported by rain waters and absorbed by the soil [8, 9]. On the other hand, the overuse of water results in wasting one of the most precious environmental resources. Estimations of water wasted by the agricultural sector point to around 2.500 billion liters a year, which amounts to around 60% of the total used [10-12].

Precision Agriculture (PA) aims to support sustainable agriculture [10]. It aims to develop more efficient and site-specific crop management practices in order to improve the yield and decrease the environmental footprint. By exploiting knowledge about the specific crop, the soil properties and other environmental parameters, crop growth strategies could be proposed for each specific case [13]. In a study presented by Mourhir et al., it has been proven that even by cutting Potassium, Phosphorous and Nitrogen quantities by half, the final yield would not be affected by more than 10% [14].

Wireless sensor networks (WSN) are an important enabling technology in today’s world. They provide the opportunity to acquire real-time data from the environment in which they are deployed. They are used in a wide variety of applications [15, 16], and have also proved to be a very useful internet of things (IOT) solution for precision agriculture systems [6, 12, 17]. Agent-based modeling (ABM) is a relatively recent software paradigm. It is used for the development of highly distributed platforms. Recently, a growing interest has emerged in the research community for exploiting WSN using multi-agent systems (MAS) [18]. Different precision agriculture platforms that integrate MAS and WSN have been proposed. These include systems that rely on some
specific inference technique and make decisions to improve crop management practices. Most of these systems are application specific. They are intended to solve a certain problem and are not designed in a way that supports reusability and extendibility [19]. Examples of this platforms include the one proposed by Isern et al. in which yield is simulated based on different irrigation timings, and recommendations are provided for the most efficient one [17]. Another multia-gent platform has been proposed by Villarubia et al. [6]. It aims to optimize the irrigation process in order to maximize the crop yield, by predicting the irrigation time based on real sensor data.

In this research, we aim to contribute to the development of a decision support platform for the adoption of sustainable agriculture. We aim in particular to develop a platform that supports the integration of different modeling and simulation techniques, and that can be extended to address new issues as needed. We have adopted ABM for its ability to model complex problems where different entities are involved. Farmers, environmentalists, policy makers and experts in agriculture are all parties with separate, conflicting preferences and goals, which need to be represented independently. The ABM paradigm provides the ability to model this autonomy and sets the ground for collaboration through the concept of intelligent agents. MAS are also particularly useful for real time monitoring of wireless sensor and actuator networks (WSAN). Last but not least, MAS provide strong support for distributed computing. As to the inference part, the platform proposed in this work aims to overcome the difficulty to build exact mathematical models caused by the scarcity of data, by providing support for the use of soft computing techniques. As a first step in this direction, we have tested the system using a Fuzzy Inference System (FIS). However, other solutions, such as Fuzzy Cognitive Maps (FCM) could be easily integrated, as the platform incorporates a functionality that allows the creation of new “Inference Agents” based on experts’ knowledge. The platform also allows the addition of new agricultural sites as needed.

The rest of this report is organized as follows. In Section 2, we will introduce some literature background on MAS and FIS. In Section 3, we will present some related works on the use of MAS and FIS in PA. In Section 4, the architecture of the proposed platform is described, along with some usage scenarios. In Section 5, we will present the implementation of a case study and the
results of its execution. Finally, we will present an analysis of the seven STEEPLE aspects of this work in Section 6.

2. Literature Background

2.1. Multi-Agent Systems

2.1.1. Overview

Agent-Oriented Programming (AOP) has been introduced in 1990-1993 by Yoav Shoham [20] as a new programming paradigm centered on the concept of agents. It aims to solve complex problems where different actors are involved, and where distributed computing is required, by introducing Artificial Intelligence (AI) concepts into the field of distributed systems. Although AOP has received important interest from the scientific and research community for the last two to three decades, it is only recently that it started being employed in commercial solutions [5]. Such domains where Multi-Agent Systems (MAS) are applied include process control [3], power engineering [21, 22], transportation and logistics [23], PA [6, 17, 24, 25], video games, etc.

As its name indicates, a multi-agent system is a collection of agents. There have been plenty of attempts in the literature to define the notion of agent. Each definition differs according to the area of application of agent systems and the manner in which they are used. Thus, there is no standard definition of the concept. However, it is commonly believed that agents can be either physical or virtual (software agents), and that each agent represents an autonomous entity characterized by its reactivity, proactivity and social ability [26]. Autonomous here means the ability to act independently of any external interference in order to solve complex and long term tasks, and to keep a certain extent of self-control [5, 27]. Reactivity points to the fact that an agent should always be able to react to external events, such as a change in the environment. Being proactive means that agents also have the ability to take initiatives without any kind of stimulus. Finally, a social agent can interact with other agents in order to achieve a common goal. Each agent should be able to receive messages and to start a communication with other agents [5, 26].
The use of MAS has known an increased interest following the growth of Internet resources and of the dependence on distributed computing. Between the years 2000 and 2005 alone, the value of MAS has gone from $112 million to $1.3 billion [7]. This important development called for the standardization of the realm of MAS. Indeed significant efforts have been provided by industrial and research organizations for this purpose [28], the main challenges being enabling agents to easily find each other and handle interactions [29]. In particular, the Foundation for Intelligent Physical Agents (FIPA) has emerged as the most important player in this standardization project. It is an international non-profit organization that aims to improve the development of agent-based solutions [7], and to enable their interoperability with non-agent technologies [30]. In 2005, it has earned the acceptance of the Institute of Electrical and Electronics Engineers (IEEE) [30]. FIPA specifications do not aim to unify the internal implementation details of MAS. The objective is rather to build a common framework that would organize the external aspects of MAS in order to promote interoperability [31]. A list of 26 standard FIPA specifications can be accessed on the official website of the organization. Among these, three are most important, namely: the FIPA Abstract Architecture Specification, the FIPA Agent Management Specification and the FIPA SL Content Language Specification. Figure 2.1.1.1 illustrates the reference model for the development of agent platforms proposed by FIPA, as presented in Ahmad Farooq’s paper [7].

![Figure 2.1.1.1: FIPA Reference Model](image)

The main components of this model are the Agent Communication Channel (ACC) which enables the intercommunication between agents, the Agent Management System (AMS) which is responsible for the management of agents’ creation, registration and deregistration, and the
Directory Facilitator (DF) which plays the role of a yellow-pages directory. Agents declare their services to the DF and use it to find other services they need.

There are several MAS development environments. However, we have selected the Java Agent Development Framework (JADE), which is considered as the most used MAS framework in research and industry. There are many reasons behind this choice. In the next paragraph are the most relevant ones.

A fundamental requirement for our platform is the ability to serve multiple agricultural sites located in different places. The sensors data gathering is achieved locally in each site, and is communicated to a base station through radio transmission. Therefore, part of the platform that monitors a specific site should be hosted near the WSAN. This is easily solved by the architecture of JADE platforms that is based on the use of agent containers. Each container represents an environment on which multiple agents can live and, more importantly, each container can be deployed on a different physical host. Thus, a container is created and deployed in each site. Then, agents responsible for monitoring the WSAN are deployed on the container. Moreover, the communication between different containers is achieved in a transparent way to the developer, i.e. developers handle communication between agents living in remote hosts in a similar manner as between agents sharing the same host. In the following section, we will present an overview of the JADE environment in order to provide the reader with a more concise understanding of the advantages offered by the framework.

2.1.2. Java Agent Development Framework
2.1.2.1. Overview

JADE is a software framework for the development of agent applications. It was started by Telecom Italia in 1998 and went open source in 2000 [5]. The JADE framework is fully developed in Java and is compliant with the FIPA specifications for interoperability between intelligent MAS. It aims to facilitate the development of MAS and guarantee standard compliance by proposing a set of ready to use and customizable core functionalities [5]:

14
• A FIPA-compliant platform where the AMS, the DF and the ACC are activated automatically at start-up.

• A fully distributed platform where different containers can be deployed on different hosts, possibly on different machines. On each host, agents live and run on their separate threads, and are capable to communicate with each other. Moreover, each agent can still execute more than one thread in a parallel way.

• An efficient transport mechanism for asynchronous messages where messages exchanged between agents of the same platform are transferred without marshalling and unmarshalling procedures, as they are encoded as Java objects. Furthermore, messages exchanged with agents living on different platforms are automatically converted, transparently to the developer, to FIPA compliant syntaxes.

• A simple and effective agent life-cycle management. Newly created agents are automatically assigned a unique global identifier and a transport address, and are registered with the AMS. Agent life cycle can be locally and remotely managed through simple APIs and graphical tools.

• A collection of graphical tools useful for monitoring and debugging purposes.

• A support for ontologies and content languages where ontologies and content encoding are checked automatically. Programmers have the ability to select desired content languages and ontologies, and develop their own.

• A library of ready-to-use FIPA interaction protocols and of customizable Java classes for development of application specific protocols.

• Support for J2ME platform and deployment in wireless environments.

• An extensible kernel that allows the extension of the functionalities of the platform by programmers.
2.1.2.2. Architecture

JADE platforms can be distributed over several containers hosted in different machines. Containers represent the JADE runtime environment in which agents live. A special container, the Main Container, is mandatory for any JADE platform. It is the first one to be launched, and all other containers have to register with it when created for them to join the platform. Figure 2.1.2.2.1 illustrates two typical JADE platforms, where one is distributed on three containers.

The main container should also host the AMS and DF defined by the FIPA specifications. JADE implements these as agents instantiated on the main container upon creation. The AMS agent is responsible for monitoring the entire platform. All agents register with it automatically once created and are assigned their AIDs from it. Platform management operations are possible only through the AMS agent. Agents wishing to manage their life cycle, to create or kill other agents or containers, etc. have to request it from the AMS agent. As to the DF agent, it is commonly referred to in the JADE community as the yellow pages agent. Agents register the services they offer with the DF agent, and use it to search for agents that provide specific services.

Figure 2.1.2.2.1: JADE Architecture [4]
2.1.2.3. Message Transport

A Message Transport Service (MTS) is one of the three fundamental services that should be implemented by an agent platform according to FIPA specifications. The MTS is the service responsible for exchanging messages within the same platform, and with other platforms [5].

For intra-platform communication between agents, the messaging architecture implemented by JADE is based on the platform’s internal message transport protocols (IMTP). In this case, the message transport mechanism that reduces delivery time the most given the addresses of the agents involved is selected automatically by the platform. When the sender agent and the receiver agent are both in the same container (i.e. running on the same JVM), the message is cloned and the object reference is sent to the receiver. When the two agents do not live in the same container, the message is transmitted using the Java Remote Method Invocation (RMI) method [1, 5, 31].

As to inter-platform communication of messages, it is achieved using the ACC specified by FIPA, which is distributed across all the platform’s containers in the JADE implementation. All the standard Message Transport Protocols (MTP) specified by FIPA are implemented by JADE. An HTTP-based MTP is started by default on the main container. However, programmers may start a different MTP, or even prevent the default one from being started. Moreover, each container of the platform can be launched with any number of MTPs. The platform then chooses the best one according to the situation. JADE predefined a set of MTPs, among which IIOP-based MTP and HTTP-based MTP. Figure 2.1.2.3.1 illustrates the JADE messaging architecture.
2.1.2.4. Agent Communication

Agent communication in JADE is implemented in compliance with the FIPA specifications, and is one of the most important features of the framework. It is based on asynchronous message passing. Messages are received by agents in their corresponding message queue, or “mailbox”. When a message is pushed into this queue, the agent is notified. Then, it is up to the developer to define when to pick the message.

Messages in JADE are instances of the `jade.lang.acl.ACLMessage` class. They are composed from [5]:

- A sender.
- A list of receivers.
- A performative, or communicative act, which indicates the intention of sending the message. The most used performatives are: REQUEST, INFORM, PROPOSE and CFP.
- The content of the message.
- A content language understood by the sender and the receiver in order to be able to parse and encode the content.
- An ontology (optional) that indicates a vocabulary that defines the symbols used in the content.
Other fields to control concurrent conversations and set timeouts.

The content of the ACLMessage is usually quite complex. Simply storing it as a string is not really convenient to parse it and decode it. The content could also be sent as a Java Object. However, this would require the sender to convert the content from its internal representation to the corresponding ACL content expression, and the receiver agent to do the conversion the other way around, in addition to check that it is semantically meaningful. In order to solve these issues, JADE proposes a mechanism based on content languages and ontologies. It works as depicted in Figure 2.1.2.5.1.

![Figure 2.1.2.4.1: JADE Support for Content Languages and Ontologies [5]](image)

This is performed by a `jade.content.ContentManager` object, which delegates the conversions and checks to an ontology responsible for the semantic validation, and a content language code responsible for the conversion into strings or bytes sequences.

Ontologies are nothing but a representation of the elements involved in the MAS. There are three main types of ontologies according to the JADE content reference model: **Predicates** which take either a true or false value, and that give an information about the status of the domain of discourse, **Concepts** which represent a complex entity of the domain of discourse, and **Agent Actions** which, as their name indicates, define actions to be performed by agents [5].

2.1.2.5. Agent Execution
As we have mentioned previously, an agent must be autonomous, meaning that it should not only wait for external stimulation, but should have the ability to initiate a new communicative act. This is why each JADE agent has its own thread of control. Also, each agent has the ability to carry different activities simultaneously, among which engaging in multiple conversations with other agents. In JADE, agent tasks (the jobs that agents have to do) are implemented through “behaviours”. All the behaviours of an agent share the same thread, and are ran according to a non-preemptive round-robin policy. However, programmers have the option to create special behaviours that run on their own threads.

Figure 2.1.2.5.1: Agent Thread Path of Execution [5]
From a developer perspective, creating an agent is equivalent to writing the code for a class that extends that the jade.core.Agent class. The class should implement the `setup()` method, that is executed when the agent is initiated for the first time. Creating a behaviour is done by instantiating an object of the class jade.core.behaviours.Behaviour. Then, the behaviour must be added to the agent’s thread of execution by calling the method `addBehaviour()`. JADE API offers large set of ready to use behaviours, such as the OneShotBehaviour, the CyclicBehaviour, etc. [5, 31]. Figure 2.1.2.4.1 depicts the path of execution of an agent thread.

2.2. Fuzzy Inference Systems

In this section, an overview of FISs is presented, along with descriptions of the fuzzy logic formalism and the fuzzy inference process. Then, the library used in the implementation of the case study is described.

2.2.1. Overview

Expert systems aim to use human expertise in order to solve complex problems [32]. An expert system should have the ability to emulate and quantify natural language in order to build rules based on abstract expert mental images. These rules constitute the knowledge base of the system. Expert systems should also incorporate an inference engine that acts on the rules of the knowledge base [33]. On another hand, fuzzy logic provides methods to interpret data qualitatively and quantitatively through linguistic variables and membership values.

Fuzzy logic has been introduced by Zadeh in 1984 [34]. It has been proposed in contrast to crisp or Boolean logic, and provides a method for handling partial truth where variables have a truth value between 0 and 1 [35]. In this way, it becomes possible to express vague and imprecise knowledge through the use of membership functions. Membership functions consist in sets that divide the domain of discourse of a variable (fuzzy sets) and linguistic descriptions. Furthermore, fuzzy logic provides an inference process that works on top of relationship expressions between cause concepts and effect concepts. These expressions are named fuzzy rules and are defined by experts in the specific domain of application.
FISs are among the most used applications of fuzzy logic [36]. It is basically a combination of expert systems and fuzzy logic. An FIS is therefore an expert system that applies fuzzy logic concepts for the development of its knowledge base and inference engine [30]. There are many domains of applications of FIS. They have proved to be very useful for solving complex problems where a concise mathematical solution cannot be applied, thanks to their ability to model vagueness and ambiguity [37]. Among the many fields of applications of FIS, we can cite power electronics [38], civil engineering [39] and PA [37]. In what follows, we will briefly describe the main concepts behind fuzzy logic and FIS.

2.2.2.  Fuzzy Logic Formalism

2.2.2.1. Membership Functions

A membership function is a function that represents a fuzzy set. Membership functions allow the determination of the membership value (a value between 0 and 1 that represents the degree of belonging to a fuzzy set) of members of a fuzzy set. These functions are usually chosen based on experts’ knowledge. They could be represented in a triangular, trapezoidal, sigmoid or Gaussian shape. Following is an example of a trapezoidal membership function and the different membership values a variable could take.

\[
\mu_A(x) = \begin{cases} 
0, & x < a \text{ or } x > d \\
\frac{(x - a)}{(b - a)}, & a \leq x \leq b \\
1, & b \leq x \leq c \\
\frac{(d - x)}{(d - c)}, & c \leq x \leq d 
\end{cases}
\]

2.2.2.2. Fuzzy Rules

Fuzzy rules are used as the linguistic representation of fuzzy knowledge. They are the statements that control the classification decision of input and output values. Fuzzy rules are made of three components: antecedents for describing the input variables of the system, linguistic connectors such as AND and OR for linking the antecedents, and a consequent following a THEN operator for describing the decision about the output variables. Following is the format of a fuzzy rule:
2.2.2.3. Fuzzy Inference

The fuzzy inference process is made of four steps, namely fuzzification, rule composition, rule implication, aggregation and defuzzification.

a) Fuzzification

Fuzzification is the process by which the fuzzy terms of input variables are mapped into membership values between 0 and 1. For each input variable, the corresponding fuzzy sets and membership functions are employed. The process consists simply in drawing a line from the input variable to the membership functions. The intersection point is the membership value. Figure 2.2.2.3.1 illustrates the fuzzification of x1 using three membership functions: Short, Medium and Tall.

![Fuzzification example](image)

\[ \mu_{(x = \text{short})} = 0.5 \]
\[ \mu_{(x = \text{medium})} = 0.2 \]

Figure 2.2.2.3.1: Fuzzification example [2]

b) Rule Composition and Implication
The Fuzzification process described above considers one input variable, i.e. one antecedent. However, there is a need to develop a belief about the truth of a consequent by propagating over multiple antecedents. This is usually achieved through a fuzzy t-norm operation. After rule implication, comes the rule implication process. It is through implication that the membership function of the rule consequent is correlated with the membership values of the rule antecedents. This is done through clipping or scaling. The first technique is the simpler. It consists in slicing the output variable’s membership function at a strength similar to the antecedent’s. The second is more expensive from a computational perspective. It consists in multiplying the membership function of the consequent by the strength of the rule of the antecedent, which allows the preservation of the shape of the output fuzzy set.

c) Aggregation

Aggregation is the process by which all rules’ consequents are combined into a single output fuzzy set. Several aggregation techniques have been proposed in the literature, which rely on t-conorm operators (e.g.: ‘Or’) and the “bounded product” [40]. However, Max Aggregation, which uses the
‘Max’ operator is the most widely used method. This method consists in solving overlaps between fuzzy terms by considering the maximum membership value, as shown in Figure 2.2.2.5.1.

d) Defuzzification

In many cases, the purpose of the application of an FIS is to generate an output in the form of a single crisp value. Defuzzification is the process needed to achieve this. Different methods have been proposed for this purpose, among which the “maximum”, the “mean of maxima” and the “height” methods [41]. However, the centroid (or center of gravity) method is the most used one. The center of gravity (COG) is computed according to the following formula:

$$\text{COG} = \frac{\int \mu_A(x) \cdot x \, dx}{\int \mu_A(x) \, dx}$$

![Figure 2.2.2.3.4: COG Defuzzification [2]](image)

2.2.3. \textit{JFuzzyLogic Library}

JFuzzyLogic is an open source fuzzy logic library, fully implemented in Java, which provides an API for the design and development of FISs, or fuzzy logic controllers (FLC). It is compliant to the standards specified by the IEC 61131 norm of the International Electrotechnical Commission. JFuzzyLogic is based on the Mamdani model proposed by Mamdani and Assilian [42], which is considered as the most used FIS. It aims to simplify the development of fuzzy systems in research and industrial environments [43, 44]. As will be explained below, Mamdani FISs use a min-max approach and provide a more intuitive output.
In order to use the jFuzzyLogic API, the FIS should be described in the standardized fuzzy control language (FCL). In FCL, the FIS is composed of blocks where the input and output variables, the fuzzy rules used for inference, the aggregation method used, and other FIS parameters are described. A “Hello World” example for FIS is provided in the documentation of the library. It solves the problem of calculating the tip in a restaurant. Figures 2.2.3.1 to 2.2.3.4 show the FCL code used for this problem. We will use it to briefly explore the structure of FCL files.

First of all, a function block should be defined, as there might be several in one file. Then, all input variables should be defined. In these case, all variables are real numbers.

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

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

// Define output variable
VAR_OUTPUT
  tip : REAL;
END_VAR

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

Figure 2.2.3.1: Tipper FCL – Variables Definition [3]

Figure 2.2.3.2: Tipper FCL – Fuzzification [3]
The next step is to define membership functions for input variables linguistic terms in FUZZIFY blocks. The functions can be described either as in the example, using the coordinates of the points that define them, or using keywords such as TRIAN and TRAP. These functions define how the input variables will be fuzzified.

Next, a DEFUZZIFY block is used to determine how the output variables will be defuzzified. This block should contain definitions of membership functions of the output linguistic terms. In this block, the method used for defuzzification is also specified, in this case center of gravity, and the default value that should be assigned when the variable is not activated by any rule.

Finally comes the RULEBLOCK where the fuzzy rules used for inference are defined. In addition, the activation and accumulation methods are specified at the level of this same block.

Figure 2.2.3.3: Tipper FCL – Defuzzification [3]

Figure 2.2.3.4: Tipper FCL – Fuzzy Rules [3]
3. Related Work

This section presents previous research and systems proposed in the context of PA using MASs and FISs.

3.1. Multi-Agent Systems in Precision Agriculture

Important research has been done in order to integrate the use of MAS in the field of PA. Different platforms have been proposed in the recent years, most of which deal with the irrigation aspect of PA. These platforms aim to take advantage of MAS’ ability to integrate the different actors of a system in a collaborative environment, in order to develop decision support tools for farmers.

The use of MAS in this context differs and the methodologies adopted are various. Gutta A. and Sajja P. have for example sought to support farmers seeking information on the web [45]. They proposed an intelligent multi-agent based framework where the cooperation of agents that retrieve relevant information from multiple farming databases allows the answering of farming queries in an effective and pertinent manner.

On another hand, many researchers have concentrated their efforts for the development of intelligent systems that would provide timely and precise support for farmers’ decisions. The platforms that have been proposed differ in regard to the factors taken into consideration, and to their specific purpose. But the overall goal is to maximize the yield while minimizing the water resources used for irrigation.

The system proposed by David Isern et al. considers the weather conditions in order to estimate the most efficient length of irrigation time and the specific time in which it should be initiated [17]. It simulates the yield based on different timings and proposes the most efficient one to the end user (the farmer). For this purpose, the system counts several agents that represent real entities involved in the irrigation of a field, among which a Sprinkler Agent, an Irrigation Agent and others. The simulation are controlled by a Controller Agent, which also receives the information of the garden to be simulated. All the knowledge required for MAS is encapsulated in a common ontology. Finally, the interaction with the end-user is achieved through a web interface.
Another platform that aims to optimize the irrigation process in order to maximize the crop has been proposed by Gabriel Villarubia et al. [6]. This multi-agent platform uses sensors to measure temperature, soil moisture around the plants, solar radiation, humidity, pH and wind, and integrates all this information in order to determine the exact volume of water and duration of irrigation required for each specific case. The system is composed of a PANGEA MAS and of several virtual organizations. The Information Fusion Organization has for objective to merge the information provided by the network of sensors and to provide this information to the other agents in a standard format. The Smart Irrigation Organization is the organization that is actually responsible for extracting the information provided by the agents embedded in the different sensors. The Control Center Organization is where the predictions happen based on the sensors’ information. The Application Interface Organization has for mission the adaptation of data from the other virtual organizations to the application layer. Finally, all the virtual organizations are managed by the PANGEA MAS. This system is in its turn made of several agents: a Database Agent that persists all the data in a database, an Information Agent that allows the communication and coordination between agents by holding information about all services provided, a Normative Agent responsible for security and authorizations, a Manager Agent for periodic overall management of functionalities, and an Organization Manager which deals with security of the virtual organizations and encryption of communication frames. The technique adopted for making predictions is a Fuzzy Logic based algorithm. The knowledge is provided by experts in the form of rules. The agents communicate via RESTful web services, where data is exchanged through JSON frames.

On their part, Wanyama T. and Far B. [46] have proposed a system made of three agents, namely the Irrigation Scheduler, the Weather Agent and the Plant Watering and Monitoring Agent. This system seeks to create an environment of collaboration between different irrigation systems in order to optimize community water usage. The Irrigation Scheduler is the agent that sets the irrigations schedule and collaborates with the other irrigation systems. The Weather Agent is responsible for reading weather information from the web and providing it to the other agents as well as human users. The Plant Watering and Monitoring Agent has the tasks of watering and monitoring the health of the plants. The schedule set by the Irrigation Scheduler Agent is computed using fuzzy logic and based on data provided by the Weather Agent and on the soil moisture data.
measured by the Watering and Monitoring Agent. The knowledge is stored as rules defined by experts.

### 3.2. Fuzzy Inference Systems in Precision Agriculture

There are several instances of application of fuzzy inference systems to PA, both for research purposes and industrial use.

Tremblay et al. have developed an FIS for optimizing the use of fertilizers in corn fields by reducing the quantities of Nitrogen used [47]. The FIS proposed recommends spatially variable applications of Nitrogen based on soil electrical conductivity, soil elevation and soil slope membership functions and fuzzy rules based on expert knowledge. The system was tested on corn crop, and on various soil and plant status conditions. The comparison of the results with the crop yield under standard variable-rate applications showed that the use of Nitrogen could be reduced to up to 25%.

Jones and Barnes have combined the use of fuzzy logic with remote sensing and crop models in order to develop a precision crop management framework [48]. The framework proposed is based on data collected in 1994 on a 770 ha cotton field. The crop model used is the CALifornia GOSym cotton growth model that predicts cotton growth based on soil water, meteorological variations and soil Nitrogen. The framework developed in this work assists users in selecting the best alternatives suggested by the crop model depending on her individual or corporate values and preferences, and the degrees of imprecision of each information source and of each alternative. Also, the framework assists the user in structuring the decision process and in examining the trade-offs between alternatives and interests.

Other works can be found in the literature, among which Badariah et al.’s where fuzzy logic has been used along support vector machines for the development of an automated system for sorting and grading of agriculture produce [49]. In this system, images of produced fruits are processed using image processing techniques. Then, they are sorted using support vector machine. Finally, they are graded using an FIS that takes as input the fruit’s length, area and size.
4. Platform Proposed

4.1. Platform Design

In what follows, we will present a description of the proposed platform. The architecture of the platform is illustrated in Figure 4.1.1.

![Figure 4.1.1: Platform Architecture](image)

**4.1.1. Physical Layer**

The physical part of the platform is distributed over the different agricultural sites monitored by the system. It includes wireless sensors, actuators and base stations. The hardware is site specific and the internal functional details are out of the scope of this research. In principle, the sensors should be able to collect data from their environment and communicate it to the base station. The base station should be able to receive the data and delegate it to the JADE agents for processing.
It should also be able to control the actuators in order to carry out the decisions of the MAS [18]. Communication in the WSAN is usually achieved through radio (e.g. Zigbee, IEEE 802.15.4) [50].

4.1.2. JADE Platform

The JADE MAS in our platform is distributed over multiple containers. As required by the JADE specifications, it contains a main container which keeps record of references and transport addresses of the other platform containers and of all the agents of the platform. In addition, the main container hosts two necessary agents for the management of the platform according to the FIPA specifications: the Agent Management System (AMS) and the Directory Facilitator (DF) agents. The AMS agent is responsible for monitoring the entire platform. All agents register with it automatically upon creation and are assigned their AIDs from it. Platform management operations are possible only through the AMS agent. Agents wishing to manage their life cycle, to create or kill other agents or containers, etc. have to request it from the AMS agent. As to the DF agent, it is commonly referred to in the JADE community as the yellow pages agent. Agents register the services they offer with the DF agent, and use it to search for agents that provide specific services. Another functionality offered by the DF agent, and of particular relevance to our platform, is the possibility for agents to subscribe to the DF in order to receive notifications each time a specified service is registered, deregistered or updated [5, 31].

The main container of our platform contains three other types of agents: a Monitor agent, a Knowledge Acquisition and Agent Creation (KAAC) agent, and Inference agents.

The Monitor agent is, as its name indicates, responsible for controlling the operation of the platform. It contains a JADE Threaded behaviour for every monitored site. A ThreadedBehaviour is simply a behavior running on its own dedicated thread (refer to section 2.1.2.5 for more information on JADE behaviours). It is responsible for requesting sensor observations from Site agents after the corresponding specified time interval, for checking the consistency, sanity and completeness of the observations, and for sending these observations to the appropriate Inference agent. Then, it receives the result of the inference from the Inference agent and sends the data (observations and inference result) to the Database agent for persistence, and the inference result to the site agent to act on it consequently.
The Monitor agent should also keep record of the addresses of all Site agents and Inference agents, and should be aware of the creation or deletion of agents of these two types. It does this by initiating a FIPA-Subscribe interaction protocol with the DF agent. On the setup() method of the Monitor agent, a behaviour is added where the interaction protocol is initiated through the instantiation of the SubscriptionInitiator class and where the services desired are specified.

There are many advantages for having a central monitor agent, in addition to the ones mentioned above. First, the Site agents deployed on the sites containers are hardware specific; meaning that they can be configured in different ways, depending on the technology and the technique used to integrate agents. Thus, we would like ideally to minimize the work done by these agents. The best way in which this can be achieved is by limiting their actions to merely being able to receive and communicate the observations values, and to receive the inference result and act on it. Also, the Site agents will be deployed on the farmers’ machines. The Monitor agent represents a level of abstraction for the users and allows keeping the core logic of the platform remotely. Another benefit is that the Site agents shall not be aware of the Inference agent appropriate for the application they require, nor keep track of its AID in case it is changed; the Monitor agent is the one responsible for this mapping. Last but not least, the Monitor agent is the agent responsible for keeping track of the Database agent’s address and of sending to it the data intended for persistence in an appropriate format (ontology). Hence, the Inference agents shall not be concerned with this, and adding or changing the Database agent would only implicate modifications at the level of the Monitor agent.

The KAAC agent is the agent that provides a GUI for the acquisition of knowledge from experts, for the creation of new sites to be monitored, and for the creation of new Inference agents. This agent takes input from its user and delegates the task of creating the agent containers and agents to the AMS agent, by calling the methods createAgentContainer() and createAgent() defined on this latter.

The Inference agents are the agents responsible for carrying out the inference process. These agents are application specific. Therefore, an agent should be created each time the platform is extended with a new application. Note here that we consider an existing application that uses a
different type of inference or knowledge a different application. The Inference agents are able to receive REQUEST messages from the Monitor agent, to get the input observations from the content of the messages, and to use the values in order to initiate the inference. Then, the Inference agent should reply to the Monitor agent with an INFORM message that includes the inference result in its content.

In addition to the main container, each monitored agricultural site hosts a container. Each site container contains a Site agent in addition to an Actuator agent and a Sensor agent for each wireless sensor.

Each Sensor agent is responsible for controlling the actions of its corresponding sensor. It should be able to receive REQUEST messages from the Site agent, get the observed data from the sensor, and send it back to the Site agent in an INFORM message.

Each Actuator Agent is responsible for controlling the actions of its corresponding actuator. It should at least be able to receive REQUEST messages from the Site agent and initiate the desired action at the level of the actuator according to the content of the REQUEST message.

The Site Agent is the agent that monitors the Sensor and Actuator agents. It has the ability to receive REQUEST messages from the Monitor agent asking for observations to be made, and to send in its turn a REQUEST message to each of its Sensor agents. Then, it should wait until it receives a response from all the Sensor agents and collects the contents of all responses, perform a sanity check and organize the data in a format understood by the Monitor Agent, and send an INFORM message to this latter with the data encoded in the content. The Site agent is also able to receive INFORM messages from the Monitor agent, with the inference result in the content. It then interprets the inference result and acts on it accordingly (sending a REQUEST to the appropriate Actuator agent, with the applicable directives).

The mapping between the hardware and the software, i.e. between the base station the sensors and the actuators, and the Site agent, the Sensor agents and the Actuator agents respectively is hardware specific. Some WSANs support a JVM version and have the ability to run a modified
version of JADE for devices supporting Mobile Information Device Profiles (MIDP) called JADE-LEAP [51]. However, most of the devices used for WSANs do not have enough computing power to support the direct execution of agents. There exist several techniques to overcome this issue [18, 52-54], some of which have been developed specifically for the JADE platform [55-57].

As to the **Database Agent**, it lives on a container deployed on the machine that hosts the database. It is responsible for receiving INFORM messages from the Monitor agent when a new Inference service is registered, and preparing the SQL queries necessary for the creation of the needed tables (application table and sensor observations tables for new types of observations). It should also be able to receive REQUEST messages from the Monitor agent carrying the observation values and corresponding inference results, and to prepare queries to persist each in the corresponding database table.

### 4.2. Usage Scenarios

In this section, three basic usage scenarios of the platform are described.

#### 4.2.1. New Inference Agent Scenario

One of the core requirements of the platform is its ability to be extended with new applications, i.e. by new Inference agents. Following is the description of the process of adding a new Inference agent. In this scenario, it is assumed that the agent class has already been implemented by the platform administrator. Figure 4.2.3.1.1 illustrates the scenario through a use case diagram.
The process begins by inputting the knowledge rules by the expert, using the KAAC agent. When this is done, the KAAC agent requests the AMS agent to create the Inference agent by calling the createAgent() method of this latter. The KAAC agent includes the name of the class of the new agent, a list of the inputs needed by the agent (sensor observations), and the knowledge rules required to carry out the inference. Then, the AMS agent takes over the responsibility of creating the Inference agent. Once created, the agent registers its service with the DF agent. It creates a ServiceDescription, fills its properties, and associates it using a DFAgentDescription instance before calling the DF method register(). After the registration of the service, the DF agent notifies the Monitor agent through the FIPA-Subscribe interaction protocol. Then, the Monitor agent takes record of the address of the created agent and sends an INFORM message to the Database agent. Finally, the Database agent prepares the execution of a query to create a table for this new type of application.

4.2.2. New Site Scenario

Another fundamental functionality supported by the platform is its ability to monitor new sites. The diagram on Figure 4.2.3.2.1 describes the process of adding a new one to the platform. It is
assumed that the WSAN hardware has already been configured to function with the JADE agents, and that the agents’ classes have been developed by the platform administrator.

First, the platform administrator specifies the IP and port on which the Site container should be hosted and the agent classes to create on this container to the AMS agent in a REQUEST message. The AMS agent creates the container and the agents specified. Once the agents are created and initiated, they register their services with the DF. Then, the DF agent notifies the Monitor agent of the registration of new Site service. When the Monitor agent receives the INFORM message from the DF agent, it instantiates a new ThreadedBehaviour for the new Site agent. It uses the properties of the newly registered service from the DF and stores them in the data store of the ThreadedBehaviour. These properties include the type of application required by the site so that the Monitor agent could map between the site and the appropriate Inference agent, the frequency of monitoring so that it starts the process of taking observations and making inference, and information about the sensors and actuators used which the Monitor agents sends in an INFORM

Figure 4.2.2.1: New Site Use Case Diagram
message to the Database agent. This latter prepares the queries for the creation of the necessary tables on the database.

4.2.3.  **Site Monitoring Scenario**

Once the platform is set up, the Monitor agent starts its control over the flow of events in the platform. We will briefly describe the sequence of events involved in monitoring one single agricultural site. This is depicted in a sequence diagram in Figure 4.2.3.1.

![Sequence Diagram](image.png)

**Figure 4.2.3.1: Site Monitoring Sequence Diagram**

The process starts when the Monitor agent sends a REQUEST message to the Site agent, asking for the observations from the specific sensors. The Site agent receives this message, and sends REQUESTs to the concerned Sensor agents. It then stays in a blocking mode, waiting for responses from all the Sensor agents. Each Sensor agent receives the REQUEST message from the Site agent, gets the data from the sensor, and sends it back in an INFORM message. When the Site Agent receives all the observations, it processes them to a format understood by the other agents of the platform and sends them to the Monitor agent as part of an INFORM message. The Monitor agent then performs the required checks on the observations data, and prepares a REQUEST message for the Inference agent corresponding to this type of application. Once sent, the message is received by the Inference agent which carries out the inference process and sends back the result in an INFORM message to the Monitor agent. This latter sends a REQUEST message containing all the
observations data and the inference result to the Database agent for persistence in the database. The Database agent extracts the data from the content of the message, prepares the SQL queries, and performs the persistence on the database. Meanwhile, the Monitor agent has also sent the inference result to the Site agent. The Site agent receives the INFORM message, processes its content, and decides whether it should perform an action. In such case, it sends a REQUEST to the actuator agent with the information it needs to perform the desired action.

5. Implementation - Case Study

5.1. Platform Development

In order to test the platform, we have developed a prototype that focuses mainly on the inference part. The platform implemented has been designed according to the proposed architecture. The KAAC agent has not been implemented, and the containers have all been deployed locally. We have tested the system by feeding observation values from .csv files to the Sensor agents. The observation values were organized in a way that would allow the assessment of the results of the Inference agent on different scenarios. The tested platform includes one site and one application: irrigation of tomatoes. The Inference agent incorporates an FIS based on the membership functions and fuzzy rules proposed in the work of Villarubia et al. [6].

An application specific JADE ontology has been created using the Bean-Ontology class, and used for the communication between agents. The concepts of the ontology are presented in Figure 5.1.1.
5.2. Fuzzy Inference System Description

An FIS has been developed using the jFuzzyLogic library based on the knowledge for tomato irrigation proposed by Villarubia et al. [6], and integrated into the Inference agent. The FIS determines the irrigation time based on three input variables: soil moisture, environmental temperature and solar radiation of the environment. The membership functions for the input
variables and the output are depicted in Figures 5.2.1/2/3/4, and the fuzzy rules are represented in Figure 5.2.5. The Inference agent should also set the irrigation time to zero when the moisture level is above 20%.

Figure 5.2.1: Soil Moisture Membership Functions [6]

Figure 5.2.2: Temperature Membership Functions [6]
Figure 5.2.3: Solar Radiation Membership Functions [6]

Figure 5.2.4: Irrigation Time Membership Functions [6]
Based on this knowledge, an FCL file has been developed for the FIS. A screenshot of this file is
presented in Figure 5.2.6.

5.3. Platform Testing

Once implemented, the platform is started and the agents are created on their corresponding containers as shown in Figure 5.3.1 (from the JADE RMI graphical tool).

![Figure 5.3.1: Platform Execution](image)

We have ran a Sniffer agent that tracks messages exchanged in a JADE based environment. The output given by this agent on the RMI GUI confirms that the platform behaves as expected. One can notice the similarity of Figure 5.3.2 with the sequence diagram presented in section 4.2.2.3.

The messages exchanged with “Other” agents are the ones involved in the agent discovery. They are exchanged with the DF agent.
5.4. Database Design

The database handled by the Database agent and needed for the persistence of observations data and inference results has been developed in MySQL. The design of the database is illustrated in an ERD, Figure 5.4.1.
5.5. Inference Results

The platform has been executed based on different observation scenarios in order to assess the behaviour of the Inference agent. Executing the following query on the database allows us to see these results.

```sql
SELECT moisture_value, temperature_value, solar_radiation_value, irrigation_time FROM tomatoirrigation i
JOIN moistureobservations m ON i.moisture_id = m.moisture_id
JOIN temperatureobservations t ON i.temperature_id = t.temperature_id
JOIN solarradiationobservations s ON i.solar_radiation_id = s.solar_radiation_id
```
Figure 5.5.1 depicts part of the results of the query ordered by moisture_value, and Figure 5.5.2 shows another part ordered by irrigation_time.

<table>
<thead>
<tr>
<th>moisture_value</th>
<th>temperature_value</th>
<th>solarradiation_value</th>
<th>irrigation_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.4652</td>
<td>27.0475</td>
<td>8734.95</td>
<td>0</td>
</tr>
<tr>
<td>20.4962</td>
<td>23.7394</td>
<td>2387.97</td>
<td>0</td>
</tr>
<tr>
<td>20.5932</td>
<td>17.0086</td>
<td>9298.92</td>
<td>0</td>
</tr>
<tr>
<td>20.9297</td>
<td>15.7637</td>
<td>6079.27</td>
<td>0</td>
</tr>
<tr>
<td>21.3646</td>
<td>20.5238</td>
<td>8300.98</td>
<td>0</td>
</tr>
<tr>
<td>21.3646</td>
<td>20.5238</td>
<td>8300.98</td>
<td>0</td>
</tr>
<tr>
<td>21.5526</td>
<td>33.9153</td>
<td>4570.15</td>
<td>0</td>
</tr>
<tr>
<td>21.7213</td>
<td>16.1246</td>
<td>10629.2</td>
<td>0</td>
</tr>
<tr>
<td>21.7213</td>
<td>16.1246</td>
<td>10629.2</td>
<td>0</td>
</tr>
<tr>
<td>21.9612</td>
<td>19.8827</td>
<td>4427.34</td>
<td>0</td>
</tr>
<tr>
<td>22.231</td>
<td>24.6025</td>
<td>4152.16</td>
<td>0</td>
</tr>
<tr>
<td>22.231</td>
<td>24.6025</td>
<td>4152.16</td>
<td>0</td>
</tr>
<tr>
<td>22.3268</td>
<td>16.7032</td>
<td>7315.1</td>
<td>0</td>
</tr>
<tr>
<td>22.3816</td>
<td>36.5117</td>
<td>10881.6</td>
<td>0</td>
</tr>
<tr>
<td>22.788</td>
<td>18.7904</td>
<td>8846.98</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5.5.1: Results Ordered by moisture_value
These results prove that the Inference Agent behaves as would be expected. We can see that when the moisture level is above 20%, the irrigation time is set to zero. Otherwise, the irrigation time changes according to the values of the input variables. We can notice that the irrigation time is not linearly dependent on the three input variables; there is no apparent pattern according to which it varies.

We have also generated surface plots in order to study the behaviour of the FIS. The first plot in Figure 5.5.3 shows how irrigation duration varies with respect to the temperature and soil moisture. The plot in Figure 5.5.4 illustrates the variation of irrigation duration with respects to soil moisture and solar radiation. The surface plots have been generated using GNU Octave software.

**Figure 5.5.2: Results Ordered by irrigation_time**

<table>
<thead>
<tr>
<th>moisture_value</th>
<th>temperature_value</th>
<th>solarradiation_value</th>
<th>irrigation_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.4968</td>
<td>42.2536</td>
<td>3530.51</td>
<td>15</td>
</tr>
<tr>
<td>16.0607</td>
<td>33.3356</td>
<td>10384.2</td>
<td>17.5467</td>
</tr>
<tr>
<td>0.282603</td>
<td>30.7421</td>
<td>10119.6</td>
<td>19.4316</td>
</tr>
<tr>
<td>2.0563</td>
<td>32.1916</td>
<td>7717.12</td>
<td>20.8261</td>
</tr>
<tr>
<td>5.13312</td>
<td>22.6254</td>
<td>979.41</td>
<td>21.8825</td>
</tr>
<tr>
<td>0.696867</td>
<td>26.7802</td>
<td>8628.26</td>
<td>23.2627</td>
</tr>
<tr>
<td>13.5328</td>
<td>30.4303</td>
<td>7380.63</td>
<td>26.2321</td>
</tr>
<tr>
<td>19.5491</td>
<td>15.713</td>
<td>7887.43</td>
<td>31.8394</td>
</tr>
<tr>
<td>8.72924</td>
<td>29.4154</td>
<td>5871.56</td>
<td>32.6585</td>
</tr>
<tr>
<td>18.1131</td>
<td>14.6344</td>
<td>6309.54</td>
<td>32.9928</td>
</tr>
<tr>
<td>17.412</td>
<td>15.5963</td>
<td>5390.85</td>
<td>33.8567</td>
</tr>
<tr>
<td>16.9989</td>
<td>24.3258</td>
<td>2731.31</td>
<td>34.5596</td>
</tr>
<tr>
<td>10.3365</td>
<td>27.3776</td>
<td>7402.36</td>
<td>36.2306</td>
</tr>
<tr>
<td>12.4668</td>
<td>29.0929</td>
<td>2548.47</td>
<td>36.7319</td>
</tr>
<tr>
<td>4.77307</td>
<td>27.2353</td>
<td>7161.42</td>
<td>38.4545</td>
</tr>
<tr>
<td>8.2928</td>
<td>26.9245</td>
<td>1193.1</td>
<td>38.5222</td>
</tr>
<tr>
<td>15.6486</td>
<td>15.7124</td>
<td>7290.8</td>
<td>38.5951</td>
</tr>
<tr>
<td>8.72839</td>
<td>26.2077</td>
<td>1842.55</td>
<td>38.7832</td>
</tr>
</tbody>
</table>
Figure 5.5.3: Temperature, Moisture and Irrigation Duration Surface Plot

Figure 5.5.4: Solar Radiation, Moisture and Irrigation Duration Surface Plot
The two surface plots show that irrigation duration is completely determined by temperature and solar radiation. This is due to water evaporation, which is caused by both high temperatures and light values. Irrigation duration is therefore reduced under these conditions, and increases under cooler temperatures and during the night. Soil moisture also has an important effect on irrigation duration. We can see in the two graphs that when the soil is wet, the irrigation duration is predicted as very low.

In the work by Villarubia et al. [6], an irrigation system based on the same FIS has been tested and compared to a conventional programed irrigation system. The results show that saving on irrigation water can reach up to 37% without affecting the final yield. Figure 5.5.5 shows the water consumption of the two systems over a period of 30 days.

Figure 5.5.5 : Water Consumption Comparison [6]
6. STEEPLE Analysis

6.1. Societal Factor

This platform will allow the social development of rural areas in Morocco and other countries, in which agriculture is the most vital sector, by developing the income of small to medium-scale farmers.

6.2. Technical Factor

This platform will be developed using the latest technologies. MAS and FIS are both recent technologies, on which research is still undertaken by the academic and industrial communities. JADE, the framework used for the development of the MAS is the most used MAS development environment, and is still considered to have the potential for wider use. The platform itself is developed in a way that permits extending it with other technologies in the future.

6.3. Environmental Factor

This platform will allow an important decrease in the use of fertilizers, irrigation water, etc. It will allow farmers to achieve a high yield without overusing fertilizers and water, thus reducing the pollution caused by formers and saving important quantities of the latter.

6.4. Ethical Factor

The platform will involve data from private agricultural sites of farmers. No unethical use of the data will be done.

6.5. Political Factor

The development, delivery and maintenance of this platform are not concerned with politics.

6.6. Legal Factor

No illegal use of farmers’ data shall be done.
6.7. Economic Factor

This platform aims to contribute to the development of the agricultural sector. It aims to increase the crop yield and reduce the expenses necessary for growing crops. Hence, the platform will highly benefit farmers using it and enhance their economic situation.

7. Conclusions and Future Work

In this work, we have proposed a PA decision support system for the adoption of sustainable agriculture practices by farmers. The proposed platform has been developed based on multi-agent modeling and integrates the use of soft computing techniques based on experts’ knowledge for inference and simulation. The platform can be used for different PA solutions that require the processing of knowledge using some inference method. The platform can also monitor several agricultural sites, with different crop types. The platform has been designed to be essentially extendable and scalable. Its architecture is based on the JADE framework, which is considered as the most used multi-agent platforms development environment. It is also compliant with FIPA standards. Thus, the platform can fully interoperate with other platforms. We have also developed a prototype platform and used it to test the proposed architecture. The application tested consists in controlling the irrigation of tomatoes using fuzzy inference. The results show that important savings on water consumption, up to 37%, can be achieved without affecting the crop yield.

In the future, we would like to work on further improvements to the architecture. In particular, we would refine the platform design and give more attention to the knowledge acquisition process by offering an ontology creation tool. Also, we would like to test the platform using other soft computing techniques such as FCMs, which are very useful in systems with feedback. Finally, we aim to develop a complete implementation of the platform including its integration with the physical layer, and test it in a distributed environment.
References


10. !!! INVALID CITATION !!! {}.


Appendix A

Monitor Agent setup() method.

```java
public class MonitorAgent extends Agent implements SmartAVocabulary {
    private ContentManager manager = (ContentManager) getContentManager();
    private Codec codec = new SLCodec();
    private Ontology ontology;

    private AID sensorsMonitorAgent;
    private AID inferenceAgent;
    private AID DBAgent;

    private AllSensorData lastObservations;

    protected void setup() {
        try {
            ontology = SmartAOntology.getInstance();
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }

        manager.registerLanguage(codec);
        manager.registerOntology(ontology);

        SequentialBehaviour sb = new SequentialBehaviour();
        sb.addSubBehaviour(new RegisterInDF(this));
        sb.addSubBehaviour(new MonitorPlatform(this, 600000));
        addBehaviour(sb);
    }
}
```

Monitor Agent MonitorPlatform behaviour.

```java
class MonitorPlatform extends TickerBehaviour {
    public MonitorPlatform(Agent a, long period) {
        super(a, period);
    }

    protected void onTick() {
        FSMBehaviour fsm = new FSMBehaviour();
        fsm.registerFirstState(new RequestObservations(myAgent), "Request Observations");
        fsm.registerLastState(new NotIrrigate(myAgent), "Not Irrigate");
        fsm.registerLastState(new Irrigate(myAgent), "Irrigate");
        fsm.registerTransition("Request Observations", "Not Irrigate", 1);
        fsm.registerTransition("Request Observations", "Irrigate", 0);
        addBehaviour(fsm);
    }
}
```
Monitor Agent \texttt{HandleInferenceDecision} behaviour.

```java
class HandleInferenceDecision extends SimpleBehaviour {
  private boolean finished = false;

  HandleInferenceDecision(Agent a) {
    super(a);
  }

  public void action() {
    ACLMessage msg = receive(MessageTemplate.MatchSender(setAndSetAID(INFERENCE_AGENT)));
    if (msg == null) {
      block();
      return;
    }
    if (msg.getPerformative() == ACLMessage.NOT_UNDERSTOOD) {
      System.out.println("\n\n\nResponse from agent: Inference Agent NOT UNDERSTOOD!" );
    } else if (msg.getPerformative() != ACLMessage.INFORM) {
      System.out.println("\nUnexpected msg from agent Inference Agent!");
    } else {
      try {
        ContentManager content = getContentManager().extractContent(msg);
        if (content instanceof Result) {
          Result result = (Result) content;
          if (!(result.getValue() instanceof InferenceDecision)) {
            System.out.println("Unable to decode response from Inference Agent");
          } else {
            Persist persist = new Persist();
            persist.setAllSensorData(lstObservations);
            persist.setInferenceDecision(((InferenceDecision)result.getValue()));
            sendMessage(ACLMessage.REQUEST, persist, DATABASE_AGENT);
            ontologies.elements.actions.Irrigate.irrigate(lstObservations.elements.actions.Irrigate());
            irrigate.setInferenceDecision(((InferenceDecision)result.getValue()));
            sendMessage(ACLMessage.REQUEST, irrigate, SENSORS_MONITOR_AGENT);
          }
        }
      } catch (Exception ex) { ex.printStackTrace(); }
    }
    finished = true;
  }

  public boolean done() {
    return finished;
  }
}
```

Behaviour used by agents to register services with DF
Methods used by agents to fetch agents from DF.
AID setAndGetAID(String agentType) {
    switch(agentType) {
    case SENSORS_MONITOR_AGENT:
        if (sensorMonitorAgent == null)
            sensorMonitorAgent = lookupAgent(SENORS_MONITOR_AGENT, sensorMonitorAgent);
        if (sensorMonitorAgent == null) {
            System.out.println("Unable to localize " + agentType + \nOperation aborted!" );
            return null;
        }
        return sensorMonitorAgent;
    case INFERENCE_AGENT:
        if (inferenceAgent == null)
            inferenceAgent = lookupAgent(INFERENCE_AGENT, inferenceAgent);
        if (inferenceAgent == null) {
            System.out.println("Unable to localize " + agentType + \nOperation aborted!" );
            doDelete();
            return null;
        }
        return inferenceAgent;
    case DATABASE_AGENT:
        if (DBAgent == null)
            DBAgent = lookupAgent(DATABASE_AGENT, DBAgent);
        if (DBAgent == null) {
            System.out.println("Unable to localize " + agentType + \nOperation aborted!" );
            return null;
        }
        return DBAgent;
    default:
        System.out.println("Invalid agent type: " + agentType + \nOperation aborted!" );
        return null;
    }
}

AID lookupAgent(String agentType, AID aid) {
    ServiceDescription sd = new ServiceDescription();
    sd.setType(agentType);
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.addServices(sd);
    try {
        DFAgentDescription[] dfs = DFServices.search(this, dfd);
        if (dfs.length > 0) {
            System.out.println("Localized Agent " + agentType);
            return dfs[0].getName();
        }
    }
    catch(Exception ex) {
        ex.printStackTrace();
        System.out.println("Failed searching in the DF!");
    }
    return null;
}