An approach to bridge the gap between ubiquitous embedded devices and JFML: A new module for Internet of Things

Francisco Javier Rodriguez-Lozano
Dpt. Electr. & Computer Engineering
University of Córdoba, Spain
email: fj.rodriguez@uco.es

Juan Carlos Gámez-Granados
Dpt. Electr. & Computer Engineering
University of Córdoba, Spain
email: jcgamez@uco.es

Oresti Banos
Dpt. Comp. Architecture & Technology
University of Granada, Spain
email: oresti@ugr.es

Jesús Alcalá-Fdez
Dpt. Computer Science and Artificial Intelligence
University of Granada, Spain
email: jalcala@decsai.ugr.es

Jose Manuel Soto-Hidalgo
Dpt. Comp. Architecture & Technology
University of Granada, Spain
email: jmsoto@ugr.es

***Abstract*—Internet of Things enables sensors and actuators to share heterogeneous data between different devices. Such data can be used to create intelligent systems to control diverse structures available in houses, cities, or industrial environments among others. In this context, one of the most used approaches to handle these intelligent systems is based on Fuzzy Rule-Based Systems (FRBS) due to their suitability for addressing complex data and managing their imprecision. However, most of the current developments in this area are usually ad-hoc solutions limited by the intercommunication between FRBS and IoT devices. This results into significant challenges in reusing these solutions to solve latent problems. To bridge this gap, a new module for the open source library JFML is proposed to offer a complete implementation of an IoT infrastructure to develop intelligent IoT solutions based on the IEEE std 1855-2016. Moreover, a case study with real IoT devices is presented to showcase the use of the proposed module.**

I. Introduction

Internet of Things (IoT) is a paradigm where physical objects (such as sensors, small microcontrollers, or embedded devices) connected to the Internet interact with each other to support intelligent decision making in everyday life. These objects are able to perceive or understand surrounding data and to work together to help making decisions in different domains.

IoT exploits the underlying technologies of these traditional devices, transforming them into more versatile devices, minimizing computing resources, and costs [1], [2]. Indeed, IoT applications range from the more common smart homes to work scenarios such as transportation, healthcare or industrial [3]–[5].

Data captured by IoT things may be quite diverse and imprecise. Hence, there is the need for introducing a methodology to generate decisions in the IoT context by using imprecise information. Fuzzy Rule-Based Systems (FRBS) represent a suitable methodology to address this task [6], providing interpretable models from complex systems by expressing their skills and expertise through simple linguistic variables and IF-THEN rules [7]. However, the design activity of FRBS is affected by difficulties related to the implementation of a same system on different hardware architectures, each one characterized by a proper set of electrical/electronic/programming constraints, which strongly emerge in a context such IoT where the interconnected things rely on a great number of heterogeneous hardware devices.

To overcome design difficulties, IEEE-CIS published the standard IEEE std 1855-2016 [8] for fuzzy systems aimed at providing the fuzzy community with a unique and well-defined tool allowing a fuzzy system design completely independent from the specific hardware/software. In order to make this standard operative, an open source Java library, named JFML [9], was also developed to provide designers of FRBS with a fully functional and complete implementation of the IEEE std 1855-2016.

Thanks to the benefits provided by the standard and its implementation in JFML, the FRBS designers have an open source software tool [10] available for designing and sharing FRBS without any additional porting task (hardware or software). However, FRBS developed making use of the JFML library are not suitable for solving a large number of IoT problems due to the fact that the many solutions for communication between devices and FRBS are ad-hoc, so they are difficult to replicate in other problems. For this reason, in this paper we present a new module for JFML based on a publisher/subscriber architecture that allows users and designers to implement FRBS in an easy and intuitive way on the basis of the IoT paradigm.

This paper is organized as follows. Section II provides some preliminary concepts related to the Internet of Things architecture and the JFML library. Section III describes the general design of the proposed module for the communication between embedded devices and JFML. Section IV presents

978-1-6654-4407-1/21/$31.00

a study case to show the applicability of the developed module using real devices. Finally, the main conclusions are highlighted in Section V.

II. Preliminaries

This section shows an overview of the main concepts as well as the communications protocols necessary to understand the proposal of this work.

A. Internet of Things architecture

There are multiple approaches [11], [12] to defining the underlying architecture of IoT, but one of the more generic architectures consists of three layers [13]: *Perception Layer*; *Network Layer*; *Application Layer*. Each layer is represented in Fig. 1 and they are described as follows:

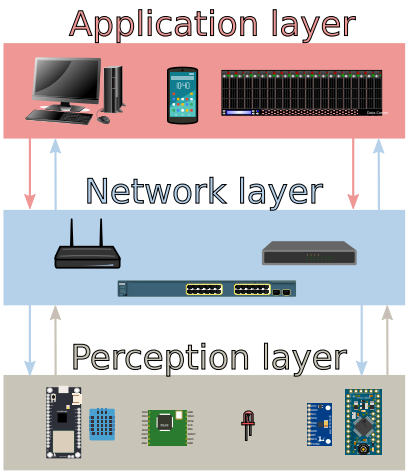

Fig. 1. Three-layer IoT architecture

1) *Perception Layer*: this layer corresponds to the lowest level of the three-layer architecture. All physical devices are considered here, especially sensors and actuators focused on collecting data and processing information to carry out actions on actuators. In addition, this layer is responsible for transmitting the information from the devices to the upper layers. For this reason, this layer is also often referred to as the *sensor layer*.
2) *Network Layer*: this represents the middle layer of the IoT architecture and is also known as the *transmission layer*. The purpose of this layer is to receive the data from the lower and upper layers and transmit the information to the devices and applications divided into these layers, specifying the routes to carry out the communication.
3) *Application Layer*: this is the highest layer of the IoT architecture. The objective of this layer is to receive the information provided by the network layer and use it in services and applications. Since server-hosted applications are typically used in this layer, it is also known as the *business layer*.

B. Message Queue Telemetry Transport protocol

Message Queue Telemetry Transport (MQTT) [14] is a machine-to-machine protocol based on the ISO / IEC 20922: 2016 standard. This protocol can be used at the network layer of the Internet of Things architecture detailed in the previous section. Additionally, the protocol enables microcontrollers to send data collected from sensors through all three layers of the IoT architecture. Internally, MQTT establishes TCP/IP-based connections where the connections remain open to be reused. The mechanics of this protocol are based on a publisher/subscriber architecture (see Fig. 2), where three different elements can be identified: publisher, broker, and subscriber.

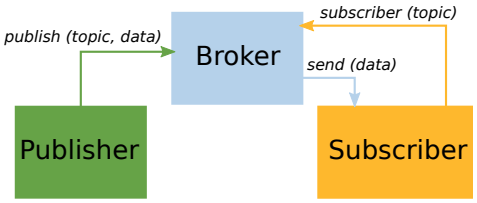

Fig. 2. MQTT architecture

- *Publisher*: this element is responsible for creating a topic and publishing the data associated with a "topic". Both the data and the topic are published in an intermediate element called a *broker*.
- *Broker*: this element acts as a server that receives and sends the communications between the publisher and the *subscriber*.
- *Subscriber*: this element makes connections with the broker to obtain the data that the publishers sent to the *broker*.

Although a single publisher and subscriber appear in Fig. 2, there may be several. For this reason, the inclusion of the broker in the scheme makes the protocol bidirectional, because at any time during the communication, publishers can become subscribers and vice versa.

This protocol plays an important role in IoT because it can be used in devices with low computational resources that could have sensors and actuators connected. Also, in terms of communications, MQTT can not only secure connections using SSL/TLS, username and password authentication, or certificate authentication, but it can also establish three different levels of quality of service (QoS): QoS 0 unacknowledged, QoS 1 acknowledged, and QoS 2 assured.

C. IEEE std 1855-2016 and JFML

The IEEE std 1855-2016 [8] was introduced by the IEEE-CIS in 2016 proposing the syntax of a new language called Fuzzy Markup Language (FML). FML was created to represent Fuzzy Logic Systems (FLS), and specifically, Fuzzy Rule Based Systems (FRBS) in a human-understandable language. This is possible because FML shares the syntax with Extensible Markup Language (XML), another famous human-readable language. Furthermore, FML provides components

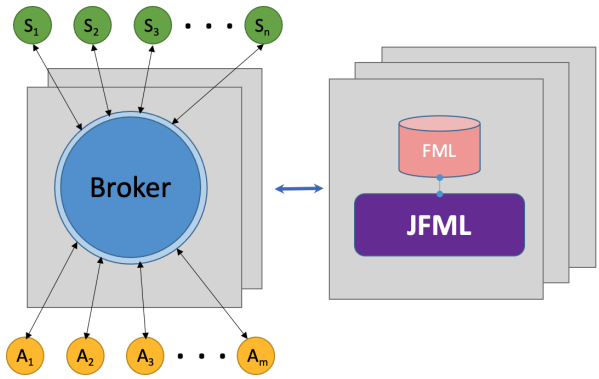

Fig. 3. General infrastructure design of IoT-JFML

to use various inference methods such as AnYa [15], or well-known methods such as Tsukamoto and Takagi-Sugeno-Kang (TSK) [16] and Mamdani [17]. In addition to these features, FML improves the IEC 61131-7 FCL [18] solving the limitation of proprietary formats and making FLS or FRBS usable in other systems, such as embedded systems.

However, the IEEE std 1855-2016 only introduced the FML syntax and its elements, but there was no available a library to design the FLS. For this reason, in 2018 an open-source library was published to design FLS according to IEEE std 1855-2016 under the name of JFML [9].

JFML is a library implemented in Java, although it is also available in Python 3.x through Py4JFML [19]. JFML provides not only an implementation of IEEE std 1855-2016, but also modules to import and export the generated FLS to other widely used formats such as Predictive Model Markup Language (PMML) or the format used by Matlab Fuzzy Logic Toolbox among others. As this library is based on the FML language, the five main components that define an FLS are also present: fuzzy knowledge base; fuzzy rule base; inference technique; fuzzification subsystem; defuzzification subsystem. Also, due to the library's package scheme, it is possible to update the library's functionality with the introduction of new modules.

III. IoT-JFML Infrastructure

A. General design

This section details the general scheme of the new proposed JFML module to solve the problem of communication between FRBS and embedded devices under the IoT paradigm. The main elements included in this design are sensors/actuators, brokers, JFML instances, and FML files representing expert knowledge according to the IEEE std 1855-2016.

A graphical representation of the main infrastructure is shown in Fig. 3. The main elements are described in the following:

- $S_1, ...S_n$: these elements represent the different sensors that provide data. For example, temperature sensors, humidity sensors, ultrasound sensors, cameras, etc.
- $A_1, ...A_n$: these elements represent the actuators that receive data and perform actions. For example, a fan, a buzzer, a light indicator, a robotic arm, etc.
- $Broker$: this element is in charge of managing data from/to sensors and actuators. For example, an MQTT server instance.
- $JFML$: this element represents devices capable of using the JFML library. For example, embedded systems, a computer, or a server in the cloud where JFML can be run.
- FML: this element contains a repository of FML files where the knowledge base, the rule base and the fuzzy inference model according to the IEEE std 1855-2016 are detailed.

B. Integration with JFML

In general, sensors/actuators provide data that pass through the brokers which are used for the JFML to make the inference according to the expert knowledge represented in the FML files. This architecture allows communication between all the ubiquitous elements thanks to the broker and the wireless capabilities. The behavior of this communication procedure and the integration with JFML is summarized below:

1) Sensors provide data so they publish them into "input" topics. In consequence, they must be associated to input variables. For example, the sensor S_1 publishes data into the topic "*input/S1*", S_2 publishes data into the topic "'*input/S2*", etc.
2) JFML is subscribed to all input topics to receive input data from the sensors and to assign them to the input variables. These input variables are defined in the FLS (represented in the FML files according to the IEEE std 1855-2016). For example, JFML is subscribed to the topics "'*input/S1*", "'*input/S2*', etc. to receive data from the sensors S_1 and S_2, respectively. These sensors are associated with the input variables, in consequence, all input variables receive data from sensors.
3) When all sensors have published their data and JFML has assigned these values to the input variables, the inference is carried out. Rules are fired according to the input values and the rule base defined in the FML files.
4) Once the inference process is finished, the output variables obtain values from the corresponding defuzzification method. Then, JFML publishes these values to "output" topics. For example, the value of the output variable *Actuator 1* is published by JFML into the topic "*output/A1*".
5) Actuators receive data so they are subscribed to "output" topics. As a consequence, they must be associated with output variables. For example, the actuator A_1 is subscribed to the topic "*output/A1*" to receive data from the output variable *Actuator 1*.

Fig. 4 illustrates, for the sake of simplicity, a general example of some classes for the IoT module in JFML. For example, the *JFML_MQTT_Subscriber* class is in charge of subscribing to the topics that are represented by means of

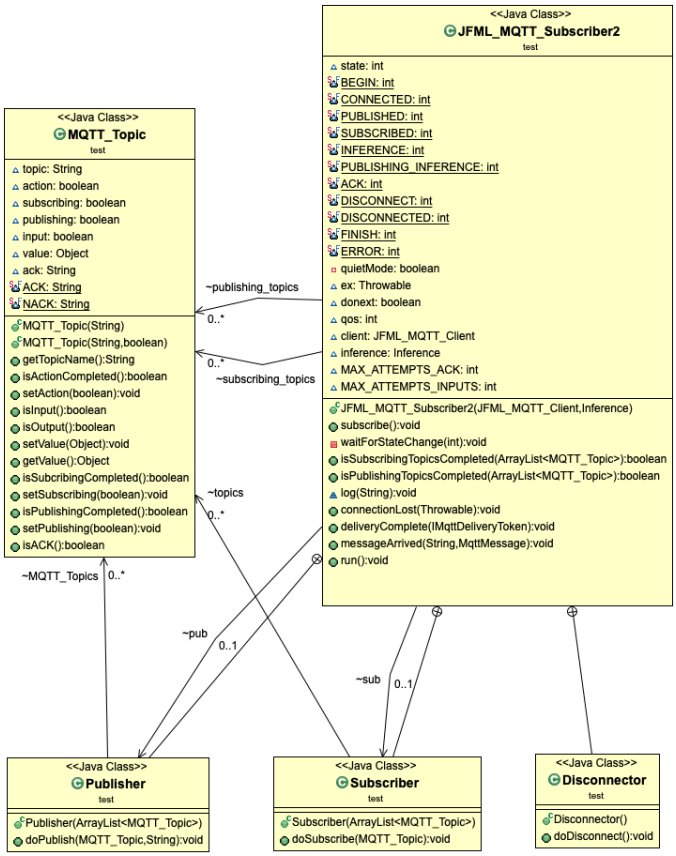

Fig. 4. Example of the class relation for the IOT module in JFML

the *MQTT_Topic* class. *JFML_MQTT_Subscriber* class defines methods to connect to the broker and a set of publishing and subscribing topics which are managed by the classes *Publisher* and *Subscriber*, respectively. The complete class diagram can be found at the official JFML web page[1].

IV. Case Study

In order to show the potential of the new JFML module, in this section we have used a FRBS to manage the temperature and humidity of a room by making use of several devices (Arduino, Rasperry Pi, ESP8266, DHT22 and a L298n with a DC motor connected). The objective is to automatically preserve the original comfort conditions defined by the user, avoiding any abrupt changes in the environment. In this case, we have selected the following devices (Fig. 5 shows the hardware architecture used in this case study):

- *Arduino yum:* this small microcontroller obtains temperature and humidity information through a DHT22 sensor. Once these values have been obtained, this device publishes these values in two topics (*"input/temperature" and "input/humidity"*) in the MQTT broker. These publications are made using the following code:
- *ESP8266:* this board has a DC motor connected as an actuator. The DC motor represents a fan that is controlled by the L298n circuit to digitally modulate its rotational speed. To get the values that control the fan speed, this board is subscribed to the *"output/fan"* topic of the MQTT broker using the following code:
- *Raspberry Pi 3 B:* this board performs the role of the MQTT broker. The broker implementation is based on the Mosquitto MQTT Broker server for Raspberry Pi devices [20] working as *a daemon process*.
- *Desktop computer:* this device performs both publication and subscription messages in the MQTT broker to obtain information from the sensor (*DHT22*) and to provide values for the actuator (*DC motor*). In addition, this device contains an instance of the JFML library and the FML repository to perform inference with the sensors data and provide a result for the actuator. The input and output values for the fuzzy system are obtained from the topics of the MQTT broker with the following code:

```
//...
//TOPICS FOR PUBLISH
const char* PUBLISH_TOPIC_TEMPERATURE = "input/temperature";
const char* PUBLISH_TOPIC_HUMIDITY = "input/humidity";

char msg[50];
//...
void publishMessages(){
    snprintf (msg, 50, "%f", read_humidity());
    client.publish(PUBLISH_TOPIC_HUMIDITY, msg);

    snprintf (msg, 50, "%f", read_temperature());
    client.publish(PUBLISH_TOPIC_TEMPERATURE, msg);
}
//...
```

Code 1. Arduino code for topics publication

```
//...
//TOPICS FOR SUBSCRIBE
const char* SUBSCRIBE_TOPIC_FAN = "output/fan";

char msg[100];

void callback(char* topic, byte* payload, unsigned int length) {
  float val;
  memcpy(msg, payload, length);
  //..
  val = atof(msg);

  if(strcmp(topic, SUBSCRIBE_TOPIC_FAN) == 0){
     //...
     // Fan spped
     analogWrite (ENA, val);
  }
}

void subscribeTopics(){
  client.subscribe(SUBSCRIBE_TOPIC_FAN);
}
```

Code 2. ESP8266 code for topic subscription

```
//...
ArrayList<MQTT_Topic> subscribing_topics = new ArrayList<
        MQTT_Topic>();
subscribing_topics.add(new MQTT_Topic("input/temperature"));
subscribing_topics.add(new MQTT_Topic("input/humidity"));

ArrayList<MQTT_Topic> publishing_topics = new ArrayList<MQTT_Topic
        >();
publishing_topics.add(new MQTT_Topic("output/fan", false));

//...
```

Code 3. Publication and subscription topics procedure in JFML

[1]http://www.uco.es/JFML/documentation

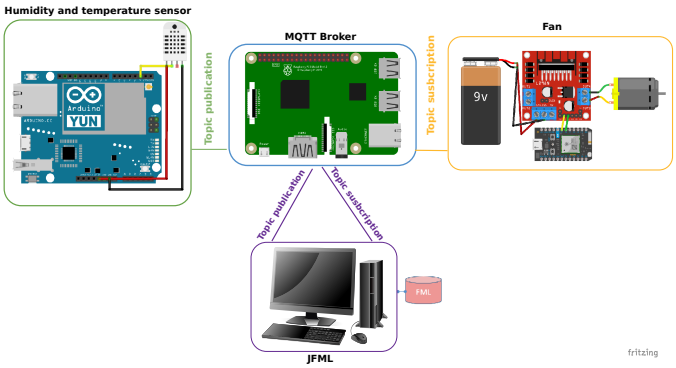

Fig. 5. General schema for the case study.

The complete code files and the FML file used in the next sections can be found at the official JFML web page.

A. Knowledge base definition

In this case, we have considered two input variables (temperature and humidity) and one output variable (fan):

- `temperature`: this variable represents the temperature in $°C$ provided by the DHT22. It is an input variable composed by the fuzzy terms *"very cold"*, *"cold"*, *"warm"*, *"hot"* and *"very hot"* in the domain $[-20.0, 100.0]$.
- `humidity`: this variable represents the relative humidity in percentage value provided by the DHT22. It is an input variable composed by the fuzzy terms *"dry"*, *"good"*, and *"damp"* in the domain $[0.0, 100.0]$.
- `fan`: this variable represents the speed of the fan in the domain $[0.0, 10.0]$. It is an output variable composed by the fuzzy terms *"low"*, *"medium"*, and *"high"*.

Fig. 6 shows the membership functions defined by an expert.

B. Rule base definition

After the knowledge base is declared, the definition of the FML file must be completed with the rule base. For this case study, the following 10 rules have been defined by an expert:

1) **IF** *temperature* IS *hot* **AND** *humidity* IS *dry* **THEN** *fan* IS *medium*
2) **IF** *temperature* IS *hot* **AND** *humidity* IS *damp* **THEN** *fan* IS *high*
3) **IF** *temperature* IS *very hot* **OR** *humidity* IS *dry* **THEN** *fan* IS *high*
4) **IF** *temperature* IS *warm* **AND** *humidity* IS *damp* **THEN** *fan* IS *medium*
5) **IF** *temperature* IS *warm* **OR** *humidity* IS *good* **THEN** *fan* IS *medium*
6) **IF** *temperature* IS *hot* **AND** *humidity* IS *good* **THEN** *fan* IS *high*
7) **IF** *temperature* IS *warm* **AND** *humidity* IS *low* **THEN** *fan* IS *low*
8) **IF** *temperature* IS *cold* **AND** *humidity* IS *good* **THEN** *fan* IS *low*
9) **IF** *temperature* IS *very cold* **AND** *humidity* IS *good* **THEN** *fan* IS *low*
10) **IF** *temperature* IS *very cold* **AND** *humidity* IS *dry* **THEN** *fan* IS *low*

C. Exemplary cases

This subsection shows some examples of different values provided by the devices detailed in Fig. 5. In these examples, the Arduino yum publishes temperature and humidity data. Then, the JFML library makes a subscription using the developed module, receives the data, makes the inference (*using Mandani as the inference model*), and publishes the result in the broker. Finally, the ESP8266 subscribes to the topic posted by JFML and adjusts the speed of the fan.

- *Case 1*: in this example, the temperature and humidity acquired by JFML from the broker are $38.2°C$ and 70.0% respectively. These variables are the inputs for the fuzzy system (*defined in the FML file*), and when the inference is finished, a value of 8.952371 (*corresponding to the fuzzy term "high"*) is provided for the fan variable. This value is published in the broker and then it is read by the ESP8266. In addition, from the output provided by JFML, it can be seen that the activated rule to make the decision is number 2:

```
RESULTS
 (INPUT): temperature=38.2, humidity=70.0
 (OUTPUT): fan=8.952371
 (ACTIVATED RULES):
  RULE 2: (0.75555545) IF temperature IS hot AND humidity IS very damp
          THEN fan IS high
```

Case 1. Hot temperature and dump room

- *Case 2*: in this case, the temperature and humidity are $30°C$ and 5% respectively, and two rules are activated to set the fan speed to a medium value. In this case, rules number 1 and 3 are activated:

```
RESULTS
 (INPUT): temperature=30.0, humidity=5.0
 (OUTPUT): fan=7.277178
 (ACTIVATED RULES):
    RULE 1: (1.0) IF temperature IS hot AND humidity IS dry THEN fan IS
            medium
    RULE 3: (1.0) IF temperature IS very_hot OR humidity IS dry THEN
            fan IS high
```

Case 2. Hot temperature and dry humidity

- *Case 3*: this is the last case and shows a situation where the ambient temperature is *cold* and the humidity is equal to 56%. Therefore, according to the output provided by JFML, a value of 3.8072174 is published to the broker to set the fan to *low* speed. For these input values, the triggered rules to make the decision are number 5 and 8 as detailed below:

```
RESULTS
 (INPUT): temperature=5.0, humidity=56.0
 (OUTPUT): fan=3.8072174
 (ACTIVATED RULES):
  RULE 5: (1.0) IF temperature IS warm OR humidity IS good THEN fan IS
          medium
  RULE 8: (0.625) IF temperature IS cold AND humidity IS good THEN fan
          IS low
```

Case 3. Cold temperature and good humidity

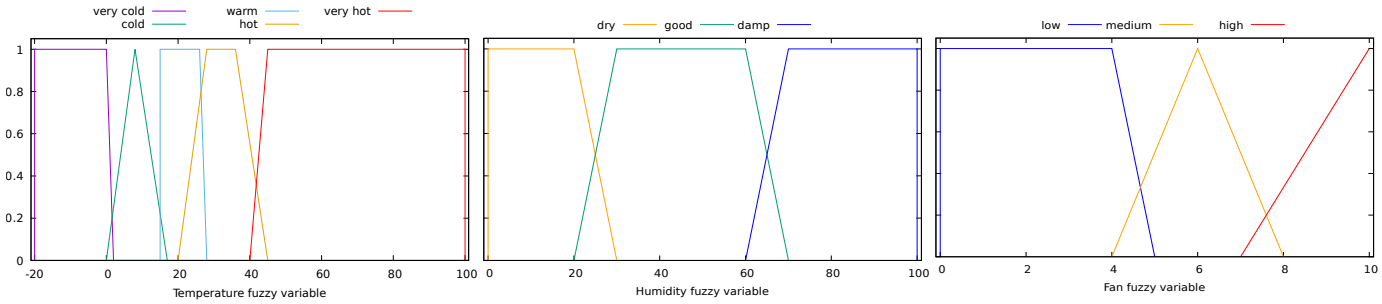

Fig. 6. Membership functions defined on the input and output variables

This new module allows us to perform a fluid communication between the controller and the sensors and actuators, enabling us to place them on the appropriate settings according to the system to obtain relevant information from the sensors with low noise and to manage the actuators accurately.

V. Conclusions

This work has presented a new module for the JFML library ready for the IoT paradigm to solve the communication limitation between FRBS and ubiquitous embedded systems. Communications in the proposed module follow a publisher/-subscriber architecture that allows devices to send and receive data through an intermediary broker. The implementation used in the proposed module is based on MQTT, a publisher/-subscriber protocol that performs efficient communications in most low-computational IoT devices.

In order to showcase the utility of the proposed module, a case study has been carried out. The case study shows the potential advantages of the proposed module allowing heterogeneous devices to share data between them, reducing the gap between IoT devices and fuzzy logic systems. More specifically, a temperature and humidity sensor was used to collect data and a DC motor (fan) as an actuator. The information of these devices is encoded in an easy and intuitive way in the proposed module as input/output variables to work with a FRBS described in a FML file. Finally, the developed module is available in the JFML library allowing its use to the scientific community, developers, or any potential user.

Acknowledgements

This research has been partially supported by the Spanish Ministry of Science, Innovation and Universities and the European Regional Development Fund - ERDF under project PGC2018-096156-B-I00 *Recuperación y Descripción de Imágenes mediante Lenguaje Natural usando Técnicas de Aprendizaje Profundo y Computación Flexible*, and by the ERDF/Regional Government of Andalusia (P18-RT-2248).

References

[1] H. Ko and S. Pack, "Neighbor-aware energy-efficient monitoring system for energy harvesting internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5745–5752, jun 2019.

[2] J. A. Ansere, M. Kamal, E. Gyamfi, F. Sam, M. Tariq, and A. Mohammed, "Energy efficient resource optimization in cooperative internet of things networks," *Internet of Things*, vol. 12, p. 100302, dec 2020.

[3] P. Spachos, "Towards a low-cost precision viticulture system using internet of things devices," *IoT*, vol. 1, no. 1, pp. 5–20, feb 2020.

[4] B. K. Chae, "The evolution of the internet of things (IoT): A computational text analysis," *Telecommunications Policy*, vol. 43, no. 10, p. 101848, nov 2019.

[5] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[6] L. Magdalena, *Fuzzy Rule-Based Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 203–218.

[7] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[8] G. Acampora, B. di Stefano, and A. Vitiello, "IEEE 1855^{TM}: The first IEEE standard sponsored by IEEE Computational Intelligence Society," *IEEE Computational Intelligence Magazine*, vol. 11, no. 4, pp. 4–7, 2016. [Online]. Available: https://standards.ieee.org/findstds/standard/1855-2016.html

[9] J. M. Soto-Hidalgo, J. M. Alonso, G. Acampora, and J. Alcala-Fdez, "JFML: A java library to design fuzzy logic systems according to the IEEE Std 1855-2016," *IEEE Access*, vol. 6, no. 1, pp. 54 952–54 964, 2018. [Online]. Available: http://www.uco.es/JFML/

[10] J. Alcalá-Fdez and J. M. Alonso, "A survey of fuzzy systems software: Taxonomy, current research trends and prospects," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 1, pp. 40–56, 2016.

[11] I. Woungang, S. K. Dhurandher, and A. Visconti, "Internet of things design, architectures and protocols," *Internet of Things*, p. 100267, jul 2020.

[12] M. Burhan, R. Rehman, B. Khan, and B.-S. Kim, "IoT elements, layered architectures and security issues: A comprehensive survey," *Sensors*, vol. 18, no. 9, p. 2796, aug 2018.

[13] M. R. Abdmeziem, D. Tandjaoui, and I. Romdhani, "Architecting the internet of things: State of the art," in *Robots and Sensor Clouds*. Springer International Publishing, aug 2015, pp. 55–75.

[14] MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[15] P. Angelov and R. Yager, "Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density," in *IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)*, 4 2011, pp. 62–69.

[16] T. J. Ross, *Fuzzy logic with engineering applications*, 4th ed. John Wiley & Sons, 2016.

[17] E. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Human-Computer Studies*, vol. 51, no. 2, pp. 135–147, aug 1999.

[18] International Electrotechnical Commission Technical Committee Industrial Process Measurement and Control, document IEC 61131,Programmable Controllers, 2000.

[19] J. Alcalá-Fdez, J. M. Alonso, C. Castiello, C. Mencar, and J. M. Soto-Hidalgo, "Py4jfml: A python wrapper for using the ieee std 1855-2016 through jfml," in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2019, pp. 1–6.

[20] Eclipse Mosquitto^{TM} An open source MQTT broker. (2021). Mosquitto man page. [Online]. Available: https://mosquitto.org/man/mosquitto-8.html.