International Journal of Computational Intelligence Systems
Vol. **12(1)**; 2019, *pp.* 204–214
DOI: https://doi.org/10.2991/ijcis.2019.125905646; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Design of Fuzzy Controllers for Embedded Systems With JFML

J.M. Soto-Hidalgo[1,*], A.Vitiello[2], J.M. Alonso[3], G. Acampora[4], J. Alcala-Fdez[5]

[1]*Department of Electronics and Computer Engineering, University of Córdoba, Rabanales Campus Córdoba, 14071, Spain*
[2]*Department of Computer Science, University of Salerno, Fisciano, 84084, Italy*
[3]*Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), University of Santiago de Compostela, 15782, Spain*
[4]*Department of Physics Ettore Pancini, University of Naples Federico II, Naples, 80126, Italy*
[5]*DaSCI Research Institute, University of Granada, Granada, 18071, Spain*

ARTICLE INFO

Article History
Received 29 Jul 2018
Revised 28 Dec 2018
Accepted 22 Jan 2019

Keywords
Fuzzy Rule-Based Systems
JFML
Embedded systems
IEEE Std 1855-2016
Open source software
Open hardware

ABSTRACT

Fuzzy rule-based systems (FRBSs) have been successfully applied to a wide range of real-world problems. However, they suffer from some design issues related to the difficulty to implement them on different hardware platforms without additional efforts. To bridge this gap, recently, the IEEE Computational Intelligence Society has sponsored the publication of the standard IEEE Std 1855-2016 which is aimed at providing the fuzzy community with a well-defined approach to model FRBSs in a hardware-independent way. In order to provide a runnable version of an FRBS that is designed in accordance with the IEEE Std 1855-2016, the open source library Java Fuzzy Markup Language (JFML) has been developed. However, due to hardware and/or software limitations of embedded systems, it is not always possible to run an IEEE Std 1855-2016 FRBS on this kind of systems. The aim of this paper is to overcome this drawback by developing a new JFML module that assists developers in the design and implementation of FRBSs for open hardware–embedded systems. In detail, the module supports several connection types (WiFi, Bluetooth, and USB) in order to make feasible running FRBSs in a remote computer when, due to hardware limitations, it is not possible that they run locally in the embedded systems. The new JFML module is ready for Arduino™ and Raspberry Pi, but it can be easily extended to other hardware architectures. Moreover, the new JFML module allows to automatically generate runnable files on Arduino™ or Raspberry Pi in order to support nonexpert users, that is, users without specific knowledge about embedded systems or without strong programming skills. The use of the new JFML module is illustrated in two case studies.

1. INTRODUCTION

Fuzzy rule-based systems (FRBSs) are rule-based systems, where fuzzy sets and fuzzy logic are used as tools for representing different forms of knowledge about the problem at hand, as well as for modeling the interactions and relationships existing between the related variables [1]. Thanks to their capability of dealing with uncertainty and vagueness, FRBSs have been successfully applied to a wide range of problems such as classification and regression [2–4]. Surely, one of the most active and mature research fields in the context of FRBSs is the area of fuzzy logic controllers (FLCs). Different from control systems based on complex mathematical models such as differential equations, FLCs use a collection of linguistic rules to model the human expertise and skills related to a given application domain [5]. In consequence of provided benefits, FLCs have been successfully used in different domain applications such as, for instance, mobile robot navigation [6], medical diagnosis [7], nonlinear rotary chain pendulum [8], and cement manufacturing plant [9].

Unfortunately, in spite of their evident benefits, the design activity of FRBSs has always been affected by strong 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 [10]. To overcome this weakness of FRBSs, recently, IEEE Computational Intelligence Society (IEEE-CIS) has sponsored the publication of a standard for FRBSs, namely IEEE Std 1855-2016 [11], capable of modeling FRBSs in an hardware-independent way. In detail, this standard defines a new W3C eXtensible markup language (XML)-based language, named Fuzzy Markup Language (FML), aimed at providing the fuzzy community with a unique and well-defined tool allowing fuzzy system design on different hardware architectures without additional efforts [5].

However, since an FLC develped in accordance with the IEEE Std 1855-2016 realizes a static view of a FRBS, tools to change this static view to a computable version are necessary. Recently, the open source Java library JFML[1] has been developed to bridge this gap

[*] *Corresponding author. Email:* jmsoto@uco.es

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

[12]. This library provides designers of FRBSs with a fully functional and complete implementation of the IEEE Std 1855-2016. Thanks to JFML, the fuzzy community can take profit of an open source software tool for designing and sharing FBRSs in accordance with the IEEE Std 1855-2016, and thus without requiring any additional porting task (hardware or software).

In spite of JFML benefits, FLCs and FRBSs, in general, implemented by means of JFML library could be not suitable to be run on some hardware systems such as embedded systems. In detail, an embedded device is a strongly specialized piece of hardware meant for one or very few specific purposes and it is usually included (embedded) within another object or as a component of a larger system [13]. The main components of an embedded device are an actuator, a sensor, an embedded processor, and often a communication network. In general, embedded devices are characterized by a limited set of hardware and/or software functionalities. Among the hardware limitations, it is worth noting the limits in processing performance, power consumption, and memory, whereas, software limitations could be few applications and no operating system or limited [14]. The difficulty to run FRBSs in embedded systems arises just from these limitations. Hence, the need of developing new tools to overcome this issue. Although some libraries allow using fuzzy logic on embedded devices such as Matlab for Arduino[2] and eFLL[3], currently there is no open source software infrastructure to support FLCs in accordance to the IEEE Std 1855-2016 for embedded systems [15, 16].

The main aim of this paper is to bridge this gap by enhancing the JFML library with a new module which makes easier the implementation and deployment of FLCs on embedded systems. When FLCs run in embedded systems, they need to be connected to a set of sensors/actuators to obtain input values and provide suitable output values to achieve a good performance of the systems. Therefore, the new JFML module allows to assign sensors/actuators to linguistic variables of the knowledge base in order to enable a FLC design completely independent from the specific hardware/software constraints that characterize the given architecture where the system is to be deployed. In addition, a new communication protocol between JFML and embedded systems is included within this module to enable both wireless communications (WiFi and Bluetooth) and the serial port. This protocol allows to run FLCs on a remote computer, when they cannot run on the embedded systems due to their limited capacity (e.g., low computational power).

In order to offer a software easily modifiable to suit the user needs and to be used as the basis for new products in different scenarios, it is important to embrace the open software and hardware models [17]. These models make it easier for other researchers to share knowledge and promote trade through the open exchange of designs. In this paper we focus on two of the most popular and well-known embedded systems in the open hardware model, Arduino™ [18] and Raspberry Pi [19]. The new JFML module provides developers with a full assistance in the design of FLCs for these two types of embedded systems. It is worth noting that this module allows to implement FLCs in Arduino boards and Raspberry Pi in an intuitive way and without the need for any additional tailored code. The potential of the new module is illustrated with two case studies: 1. an FLC that manages the wall-following behavior of a mobile robot and 2. an FLC that manages the ventilation system for a refrigerating chamber.

The rest of this paper is arranged as follows: Section 2 presents briefly both the FLCs and the embedded systems under consideration. It also introduces the main characteristics of the JFML library. Section 3 describes the new JFML module for embedded systems. Section 4 goes in depth with the two case studies. Finally, Section 5 points out some concluding remarks.

2. PRELIMINARIES

In this section, we first put in context the FLCs as a specific model of FRBSs. Then we describe the main characteristics of the embedded systems Arduino™ and Raspberry Pi. Finally, we present the open source JFML library.

2.1. Fuzzy Logic Controllers

FLCs can solve complex control situations using rules by defining automatic systems that behave like human experts in a particular domain [9, 20–22]. In these systems, a linguistic control strategy based on expert knowledge can be converted into an automatic control strategy on the basis of fuzzy rules. A fuzzy control rule is a conditional statement in which the antecedent is a condition in its application domain, the consequent is a control action to be applied in the controlled system. FLCs usually present the architecture shown in Figure 1 where the fuzzification stage translates nonfuzzy inputs (usually crisp values from the sensors) into fuzzy inputs, while the defuzzification stage does just the opposite with the outputs (usually crisp values to the actuators). A fuzzy inference engine processes fuzzy inputs and produces fuzzy outputs. With that aim, an inference mechanism interprets the given inputs in accordance with the knowledge base (i.e., the definition of all related fuzzy variables) and the rule base (i.e., the set of all fuzzy relations among the variables defined in the knowledge base).

2.2. Embedded Systems: Arduino™ and Raspberry Pi

Embedded systems are strongly specialized pieces of hardware meant for specific purposes that are embedded as part of a complete device [23]. Since they are dedicated to specific tasks,

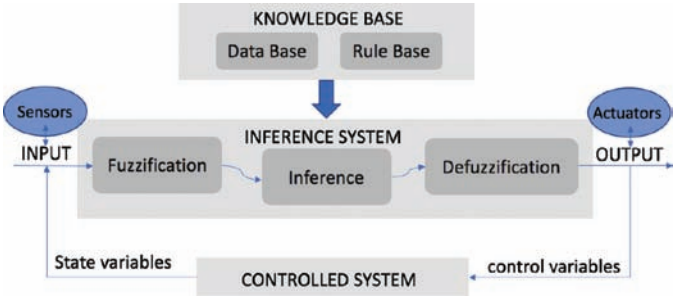

Figure 1 Common architecture of a fuzzy logic controller (FLC).

[2]http://playground.arduino.cc/Interfacing/Matlab

[3]https://github.com/zerokol/eFLL

engineers can optimize them to reduce the size and cost of the product and increase the reliability and performance. Because of this, in recent years the embedded industry has experienced exponential growth and embedded systems can control many devices. It is estimated that 98% of all microprocessors are manufactured as part of an embedded system. Embedded systems are commonly found in applications such as medical, commercial, industrial, or automotive applications.

One of the most used embedded system is Arduino^{TM} [18] due to its great simplicity and usability. Namely, it is a microcontroller board with a set of digital and analog input/output pins that may be connected to various expansion boards andother electronic components (e.g., motors, light sensors, or microphones). This board offers several communication interfaces (USB, WiFi, and Bluetooth) for connecting to personal computers or other computational systems. The microcontroller can be programmed using a dialect of the programming languages C and C++ in files with extension *.ino*.

Arduino^{TM} is the first widespread Open Source Hardware project (with license Creative Commons Attribution Share-Alike) and it is a popular tool for Internet of Things (IoT) product development as well as one of the most successful tools for STEM/STEAM education[4]. For instance, in [24], the design of a low-cost mobile robot based on Arduino was proposed as an alternative or complementary educational tool in labs, classrooms, e-learning, and massive open online courses (MOOC). In [25], an adaptive FLC based on Arduino DUE was presented to manage a DC motor with flexible shaft. In [13], Acampora *et al.* proposed an extension of the IEEE Std 1855-2016 [26] to allow the definition of an FLC in a fully interoperable manner with Arduino architectures.

Raspberry Pi [19] is a tiny and affordable computer that is usually used to learn programming skills in educational systems or as embedded computer in controllers. This is a single-board computer built on a single circuit board, with a microprocessor, memory, input/output, and other features required of a functional computer. Several communication interfaces are available depending on the selected Raspberry Pi model. Secure digital cards are used to store the operating system (e.g., Raspbian orUbuntu) and program memory in either SDHC or MicroSDHC sizes. The main programming languages are Python and Scratch, but it supports other languages.

The use of Raspberry Pi exhibits a steady growing in the literature due to its great usability and low price. For instance, in [27], Jayapriya *et al.* implemented a FLC in a Raspberry Pi to manage the charging and discharging of battery units for a wind powered microgrid. In [28], a FLC is implemented on a Raspberry Pi located at a remote location away from the plant to manage liquid levels through a wireless network. In [29], a fuzzy classifier is implemented on a Raspberry Pi to detect weed in sugarcane fields through a low-cost mobile robot.

2.3. The JFML Library

In the year 2000, the Fuzzy Control Language (FCL) was defined in the norm IEC61131-7 of the International Electrotechnical Commission [30] with the aim of providing engineers with a common and well-defined understanding of the basic means for integrating fuzzy controllers into control problems, and facilitating the exchange of controllers between different programming languages and software (increasing the usability and the interoperability of the available software). Recently, the IEEE-CIS has sponsored the publication of a new standard for fuzzy logic systems (FLSs), named IEEE Std 1855-2016 [26]. This standard defines the new FML language based on W3C XML with the aim of extending the advantages of IEC61131-7 to other types of problems (e.g., classification or regression). It exploits the advantages of XML to represent FLSs.

JFML is a new open source Java library ready to design and to use type-1 FLSs according to the IEEE Std 1855-2016. It allows to use all the fuzzy inference systems (Mamdani, TSK, Tsukamoto, and AnYa) enclosed in the XML Schema Definition (XSD) of the standard, including all the membership functions, fuzzy operators, defuzzifiaction methods, and so on, which are considered in the standard (see [26] for more information). However, researchers may need to use other elements that are not included in the current definition of the XSD (e.g., type-2 or intuitionistic fuzzy systems). For this reason, JFML includes custom methods (named according to the pattern *custom_name*) for all the elements indicated in the XSD, enabling a way to extend the library conforming to the updates of the standard without requiring to modify the language grammar itself. Moreover, the modular design of JFML based on the same labeled tree structure as the standard allows to include future changes modifying only the corresponding part of the library. JFML can also bind W3C XML documents and Java representations thanks to the API of the Java Architecture for XML Binding (JAXB). The JFML library uses JAXB to provide a fast and convenient way for reading and writing FLSs according to the standard. Finally, in order to make easier the interoperability of the library with other available software, JFML includes a module to import/export FLSs from/to three of the most widespread and widely used formats for FLSs: FCL [30], PMML [31], and Matlab FIS [32].

This library is distributed as open source software under the terms of the GNU Public License GPLv3[5] and it is hosted in the public hosting GitHub[6], which provides several tools (e.g., bug tracker) to take advantage of the open source policy. Moreover, JFML has a web page associated[7] (see Figure 2) with a complete documentation and a good variety of examples. JFML provides the fuzzy community with a well-defined tool for designing and sharing fuzzy systems without requiring any additional, hardware and/or software, porting task.

3. NEW JFML MODULE FOR EMBEDDED SYSTEMS

In this section we introduce the new JFML classes which assist developers in the implementation and deployment of fuzzy controllers for embedded systems. In the context of embedded systems, this assistance is particularly important due to the variety of applications, communication protocols, software library dependencies,

[4]http://stemtosteam.org/

[5]https://www.gnu.org/licenses/quick-guide-gplv3.html

[6]https://github.com/sotillo19/JFML

[7]http://www.uco.es/JFML

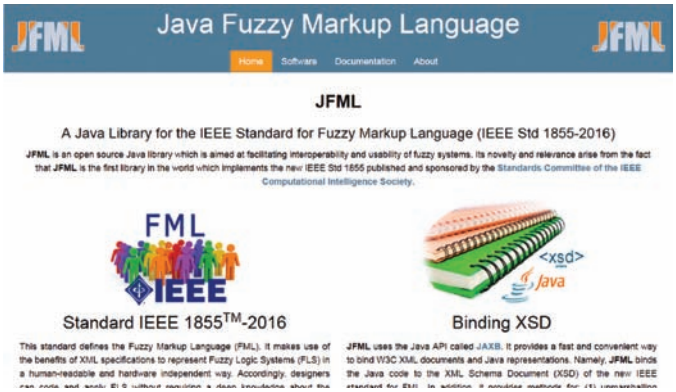

Figure 2 | Web page associated to Java Fuzzy Markup Language (JFML).

and low-level programming tools that are required. This module deals with all these issues into the JFML. As a result, FLCs developed with JFML in accordance with the IEEE Std 1855-2016 can be integrated into different types of hardware architectures without any software library dependencies or specific hardware programming tasks.

In the following subsections we describe the main features of this new module. First, in Section 3.1 we give an overview of the module design. Second, in Section 3.2, the communication protocol for the bidirectional communication between embedded systems and the JFML is detailed. Finally, the Arduino-based and Raspberry Pi-based implementations are presented in Section 3.3.

3.1. General Design

Figure 3 shows the main classes in this module:

1. *Embedded System*: Main class responsible for defining both the characteristics and the type of connection with the embedded system. It requires a name, a type of connection, and a baud rate for establishing the connection with the JFML core. It has also associated a list of *EmbeddedVariable* objects which are in charge of the association between variables of the knowledge base and sensors/actuators. In addition, this class defines an abstract method to create a runnable file according to the architecture design requirements of the embedded system. Thanks to this file, a user without specific knowledge about the embedded system architecture or programming language could run a FLC on it.
2. *Embedded Controller*: This class is in charge of coordinating all the embedded systems connected to the JFML by means of a defined protocol. As many embedded systems as the designer desires could be included in a unique *EmbeddedController* instance. This class has associated a list of *EmbeddedSystem* objects and an instance of the *FuzzyInferenceSystem* class that is responsible for the fuzzy inference engine, which could run in different computational systems thanks to the distributed capacity of the protocol. In addition, this class provides methods for running during a certain period of time or indefinitely by using the proposed communication protocol (Section 3.2).
3. *Embedded Variable*: This class enables the association between sensors and variables from the knowledge base. Sensors are usually associated to variables by means of the input/output ports of the embedded system. Reading or writing values from the sensors are mapped to the variable domain or viceversa. For instance, if the domain of a sensor were defined in the interval [0, 255] and the domain of the variable were defined in [0, 1], then actual values would be mapped accordingly.

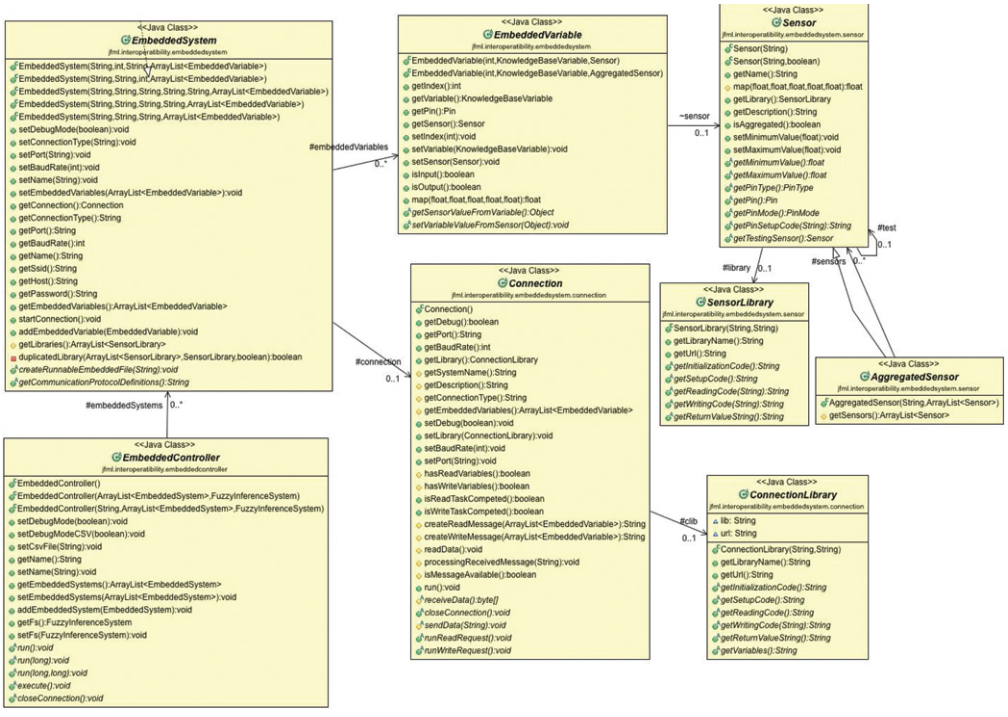

Figure 3 | Java Fuzzy Markup Language (JFML) module for embedded systems (main class diagram).

4. *Sensor* and *Sensor Library*: These classes are responsible for defining the characteristics of the different sensors. They include the type of sensor (analog or digital), the domain of the sensor (maximum and minimum read and write values), as well as associated operating libraries. Each sensor can have associated a library in order to manage the operating code by including methods for generating the initialization code, reading and writing sensors, and so on.
5. *Aggregated Sensor*. This class defines an association between different sensors with the aim of representing input/output in a unique way by the union of several sensors. To do this, a list of *Sensor* objects is included in this class. Notice that, this class allows us to preprocess the sensor values in order to generate high-level input variables and thus providing the information that is more relevant and meaningful to model the system [33]. This pre-processing task provides a great flexibility to the design since each sensor does not need to be associated with a specific input variable of the system.
6. *Connection* and *Connection Library*. The *Connection* class specifies a bidirectional connection method that allows connecting embedded systems with the core of the JFML via USB, WiFi, or Bluetooth. The *ConnectionLibrary* allows to include additional code to connect the embedded systems if it were required.

3.2. Communication Protocol

The new module allows FLCs to run on any embedded system but taking advantage of the computational capacity of other systems apart from the embedded ones, that is, the fuzzy inference process (in the core of JFML) can be executed either on the embedded system itself or on another remote computer. To do that, a communication protocol for both connecting embedded systems to a remote computer with the JFML and exchanging messages is defined. The connection is directly handled by JFML and transparent to the developer (no matter if JFML is integrated in the embedded system or run apart in a remote computer). This protocol manages the communication between the JFML and the embedded systems by means of read/write message requests. Sensors connected to embedded systems can send or receive values from or to the JFML thanks to this protocol iteratively or during a single iteration. In the following, we summarize the behavior of the protocol:

- First, the controller (*EmbeddedController* class) sends a broadcast message to all embedded systems for establishing a connection within the JFML. Each embedded system receives a message for reading and/or writing values from the connected sensors/actuators depending on the association between the sensors/actuators and the variables. For instance, if a sensor is associated to an input variable, the embedded system gets a value from the sensor. However, if it is associated to an output variable, the embedded system sets a value to the actuator. This value is sent to the JFML as a message according to the protocol definition.
- Second, the controller waits for a response (a message with the value from the sensors) from all the embedded systems that are connected to. If the controller receives response from all of them and a timeout is not exceeded, the values of the variables of the knowledge base are updated with the received values (coming from the sensors), otherwise, an error message is displayed.
- Third, the JFML makes the fuzzy inference taking into account the updated values to the input variables. Subsequently, the output values are obtained by the defuzzification method indicated in the FLC and, finally, they are sent to the corresponding embedded systems (those that have associated sensors/actuators with output variables). The sensors or actuators associated to output variables are set according to the results of the fuzzy inference. Finally, the controller waits for an ACK response from all the embedded systems indicating the results of the fuzzy inference is set correctly into the corresponding sensors or actuators, otherwise, an error message is displayed.

3.3. Arduino-Based and Raspberry Pi-Based Implementations in JFML

As we have mentioned before, Arduino^{TM} and Raspberry Pi are two of the most popular embedded systems due to their great simplicity, usability and multiple connections with sensors/actuators. In this section, Arduino-based (see Section 3.3.1) and Raspberry Pi-based (see Section 3.3.2) implementations in JFML are described. Finally, Section 3.3.3 presents a collection of sensors (already managed by the new JFML module) which correspond to some of the most commonly used solutions based on Arduino^{TM} and Raspberry Pi.

3.3.1. Arduino-based implementation

On the basis of the classes design structure proposed in Section 3.1, a new class *EmbeddedSystemArduino* which extends the abstract class *EmbeddedSystem* is implemented. Arduino boards can be connected to serial ports (USB or Bluetooth) or WiFi and used for designing FLCs in a simple way. These different connections are taken into account by the *EmbeddedSystemArduinoUSB, EmbeddedSystemArduinoBluetooth,* or *EmbeddedSystemArduinoWIFI* classes which extend the *EmbeddedSystemArduino* class.

The *EmbeddedSystemArduino* class implements the abstract method for creating automatically a runnable file (*.ino*) according to the Arduino architecture design. This file includes the code to start and run Arduino-based systems. It requires two functions: 1. *setup()* which is called once when an Arduino starts after power-up or reset and it is mainly used to initialize variables, input and output pin modes, and other libraries needed in the program, and 2. *loop()* which is called repeatedly by a program loop in the main program. This function is always called after the function setup().

Each variable of an FLC, which is associated to a sensor or actuator, as well as the related libraries, is automatically included in the code of the *setup()* method as well as the pin mode (input or output) and the pin number that is connected. Finally, the generated *.ino* file can be loaded directly into the Arduino's ROM by using, for instance, the Arduino IDE enabling Arduino board to run FLCs. Notice that this file could be edited by the users in case they needed to add or modify their own code.

3.3.2. Raspberry Pi-based implementation

Similar to the Arduino-based implementation, a new class *EmbeddedSystemRaspberry* which extends the abstract class *EmbeddedSystem* is implemented. In the same way, the classes *EmbeddedSystemRaspberryUSB, EmbeddedSystemRaspberryBluetooth* or *EmbeddedSystemRasperryWIFI*, which extend the *EmbeddedSystemRaspberry* class, are in charge of defining the different connections between the Raspberry Pi and the JFML.

The class *EmbeddedSystemRaspberry* implements an abstract method to automatically create a runnable file (*.py*) according to the Raspberry Pi architecture design. In this case, unlike the executable *.ino* file, the *.py* file does not contain the *setup()* and *loop()* functions for configuring the parameters and running iteratively the code, respectively. To achieve a similar behavior to the Arduino board allowing iterations, a *main()* function has been implemented, which contains a while loop similar to the *loop()* function in the case of Arduino. All configuration aspects such as connection pins or baud rate are declared before the *main()* function. Finally, the generated *.py* file can be executed by the command *python file_name.py*. Notice that this file can be edited by the users.

3.3.3. Collection of sensors

A collection of sensors of the most commonly used Arduino-based and Raspberry Pi-based solutions is already available within the new JFML module. Users can add more sensors and/or actuators by accessing to the programming code and extending the *Sensor* class. Table 1 shows a list of seven sensors and five actuators that are already integrated in the JFML. This list includes classic sensors [34] such as the temperature and humidity sensor DHT22, the ultrasonic sensor HC-SR04, the motion sensor H-SR501, the Gas sensor MQ-2, the light sensor LDR, and the accelerometer/gyroscope MPU6050; and actuators such as LEDs, DC motor and driver motor. Notice that sensors and actuators can be digital and/or analog. Their nature depends on the requirements of the designer.

Table 1 Sensors/actuators implemented in the Java Fuzzy Markup Language (JFML).

Sensor	Type	Digital	Analog
DHT22	Temperature and humidity	X	
HC-SR04	Distance (ultrasound)	X	
HC-SR501	Motion	X	
LDR	Light	X	X
MPU6050	Acceler. and gyroscope	X	
MQ-2	Gas	X	X
H206	Encoder	X	
Actuator	**Type**	**Digital**	**Analog**
DC Motor	Motor	X	X
L298N	Driver	X	X
LED Colour	Led	X	
LED PWM	Led	X	
SG90 Motor	Servo motor		X

4. CASE STUDY

JFML offers a complete implementation of the IEEE Std 1855-2016 and capability to import/export FLCs in accordance with other standards and software. Users can run and exchange FLCs avoiding the need to rewrite available pieces of code or to develop new software tools to replicate functionalities that are already provided by other software. These capabilities are extended to the design and implementation of FLCs on embedded systems with the module presented in this paper.

In this section we present two case studies which illustrate the potential of the new JFML module for embedded systems. First, FLCs are used to manage the wall-following behavior of a mobile robot. Second, a FLC manages the ventilation system of a refrigerating chamber.

4.1. Fuzzy Control of the Wall-Following Behavior in a Mobile Robot

In mobile robotics, fuzzy controllers are commonly considered for producing the wall-following behavior, which is frequently used to explore unknown indoor environments and to navigate between two points in a map. These controllers are in charge of preserving a suitable distance from the robot to the wall (d_{wall}) while the robot moves as fast as possible, avoiding abrupt changes in the trajectory movements and velocity. In [35], the authors designed a wall-following FLC for a Nomad 200 robot making use of genetic algorithms. It is made up of four input variables: relationship between the distance to the right and d_{wall} (RD), relationship between the distance to the left and to the right (DQ), orientation respect to the wall (O), and linear velocity with respect to the maximum linear velocity of the robot (V). This fuzzy controller has two output variables: linear acceleration (LA) and angular velocity (AV) with respect to the maximum linear acceleration and angular velocity of the robot, respectively. Table 2 shows the domain of each variable. This FLC is available as one of the examples provided with the software JFML (*./Examples/XMLFiles/RobotMamdani.xml*).

We have developed a homemade mobile robot with low-cost components for this case study. The architecture of this robot consists of an Arduino MEGA 2560 connected via USB with a Raspberry Pi, five ultrasonic sensors HC-SR04, one accelerometer/gyroscope sensor MPU6050, a servo SG-90, and two DC drive motors with matching wheels and driver H-Bridge L298N (see Figure 4). In this case, a Raspberry Pi is used as a remote computer where the

Table 2 Variables for the wall-following behavior fuzzy logic controller (FLC).

Input Var.	Domain	Output Var.	Domain
RD	[0.0, 3.0]	LA	[-1.0, 1.0]
DQ	[0.0, 2.0]	AV	[-1.0, 1.0]
O	[-45, 45]		
LV	[0.0, 1.0]		

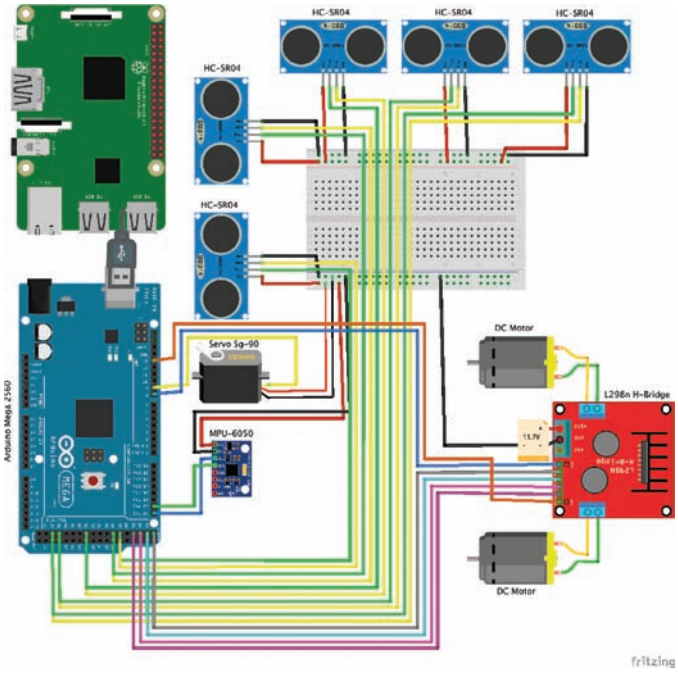

Figure 4 | Hardware architecture of our mobile robot.

fuzzy inference is performed by taking the input values (from ultrasonic and accelerometer/gyroscope sensors) which are connected to the Arduino board. In other words, the FLC is embedded in the Arduino board but the fuzzy inference is carried out in the Raspberry Pi (where JFML is installed).

Notice that the sensor values are not directly used as inputs of the FLC. They are low-level input variables that do not provide by themselves information that is relevant and meaningful to the FLC. For this reason, these low-level values are aggregated in JFML by means of the *AggregatedSensor* class in order to generate high-level input variables for the FLC. Likewise, the FLC outputs are sent to other aggregated sensors to calculate the values that will be sent to the DC motors and to the servo for the linear acceleration and the angular velocity, respectively.

The necessary steps to run an FLC with the new JFML module are as follows: The first step is to read the description of the FLC from the related FML/XML document. The second step is to instantiate the sensors/actuators that are connected with the Arduino board. The third step is to generate the aggregated sensors for the input and output values. The fourth step is to map the input/output variables with the sensors/actuators and to include them in a list. The fifth step is to create the embedded system (including the name, port, rate, etc.) and to generate the *.ino* file. The following Java code performs all the steps listed above:

```
// First step: Loading the FLC from a FML file
File fml;
fml=new File("./Examples/XMLFiles/RobotMamdani.xml");
FuzzyInferenceSystem fs = JFML.load(fml);

// Second step: Creating the sensors/actuators
KnowledgeBaseVariable rd = fs.getVariable("rd");
KnowledgeBaseVariable dq = fs.getVariable("dq");
KnowledgeBaseVariable o = fs.getVariable("o");
KnowledgeBaseVariable v = fs.getVariable("v");
KnowledgeBaseVariable la = fs.getVariable("la");
KnowledgeBaseVariable av = fs.getVariable("av");
Sensor rdSF = new ArduinoHC_SR04(rd.getName()+
"front",
ArduinoPin.PIN_40, ArduinoPin.PIN_41, 10, 200,
true, 3, true, true);
Sensor rdSF2 = new ArduinoHC_SR04(rd.getName()+
"front2",
ArduinoPin.PIN_48, ArduinoPin.PIN_49, true, 3,
false,true);
Sensor rdSR = new ArduinoHC_SR04(rd.getName()+
"right",
ArduinoPin.PIN_50, ArduinoPin.PIN_51, true, 3,
false,true);
Sensor rdSR2 = new ArduinoHC_SR04(rd.getName()+
"right2",
ArduinoPin.PIN_32, ArduinoPin.PIN_33, true, 3,
false, true);
Sensor dqSL = new ArduinoHC_SR04(dq.getName()+
"left",
ArduinoPin.PIN_30, ArduinoPin.PIN_31, true, 3,
false,true);
Sensor oS = new ArduinoMPU6050(o.getName(), -45,
45, true);
Sensor laS = new ArduinoH_BRIDGE_L298N(
la.getName(), ArduinoPin.PIN_6, ArduinoPin.PIN_7,
ArduinoPin.PIN_4, ArduinoPin.PIN_2, ArduinoPin.PIN_1,
ArduinoPin.PIN_5, -1, 1, 40, 70, 15);
Sensor avS = new ArduinoSERVO(av.getName(),
ArduinoPin.PIN_9, -1, 1, 45, 135, 45, true);

// Third step: Creating the aggredated sensores
ArrayList<Sensor> sensorsRD = new ArrayList<>();
sensorsRD.add(rdSF);
sensorsRD.add(rdSF2);
sensorsRD.add(rdSR);
sensorsRD.add(rdSR2);

AggregatedSensor rdAgg, dqAgg, oAgg;
AggregatedSensor avAgg, vAgg;
ArrayList<Sensor> sensorsDQ, sensorsO;
ArrayList<Sensor> sensorsAV;

rdAgg = new ArduinoAggregatedSensorRD(
rd.getName(), sensorsRD, 0, 3, 6, 50, true);
sensorsDQ = new ArrayList<>();
sensorsDQ.add(dqSL);
dqAgg = new ArduinoAggregatedSensorDQ(
dq.getName(), sensorsDQ, 0, 2);
sensorsO = new ArrayList<>();
sensorsO.add(rdSR);
sensorsO.add(rdSR2);
oAgg = new ArduinoAggregatedSensorO(
o.getName(), sensorsO, -45, 45);
sensorsAV = new ArrayList<>();
sensorsAV.add(avS);
vAgg = new ArduinoAggregatedSensorV(
v.getName(), -1, 1, 50);
avAgg = new ArduinoAggregatedSensorAV(
av.getName(), sensorsAV, -1, 1, 18);

// Fourth step: Mapping sensors/actuators
ArrayList<EmbeddedVariable> eVar = new ArrayList<>();
eVar.add(new EmbeddedVariableArduino(0,rd,rdSF));
eVar.add(new EmbeddedVariableArduino(0,rd,rdSF2));
eVar.add(new EmbeddedVariableArduino(0,rd,rdSR));
eVar.add(new EmbeddedVariableArduino(0,rd,rdSR2));
eVar.add(new EmbeddedVariableArduino(0,rd,rdAgg));
```

```
eVar.add(new EmbeddedVariableArduino(1,dq,dqSL));
eVar.add(new EmbeddedVariableArduino(1,dq,dqAgg));
eVar.add(new EmbeddedVariableArduino(2,o,oS));
eVar.add(new EmbeddedVariableArduino(2,o,oAgg));
eVar.add(new EmbeddedVariableArduino(3,v,vAgg));
eVar.add(new EmbeddedVariableArduino(4,la,laS));
eVar.add(new EmbeddedVariableArduino(5,av,avS));
eVar.add(new EmbeddedVariableArduino(5,av,avAgg));

// Fifth step: Creating embedded system
EmbeddedSystem robot;
robot = new EmbeddedSystemArduinoUSB
("RobotMamdani","USB_PORT", 9600, eVar);

// Creating the .ino file
robot.createRunnableEmbeddedFile("RobotMamdani.ino");
```

The generated *.ino* file has to be written in the Arduino board with the IDE provided by the company, and then the board can be connected to the Raspberry Pi through a serial USB connection. Finally, the embedded system is associated to the FLC and we can run the system with the next Java code:

```
ArrayList<EmbeddedSystem> embd_R = new ArrayList<>();
embd_R(robot);

EmbeddedController controller;
controller = new EmbeddedControllerSystem(embd_R,fs);
controller.run();
```

Notice that the embedded system named *embd_R* is included in a list because several Arduino boards can be used to connect with other sensors/actuators. As depicted in Figure 5 (the robot trajectory is highlighted in red), our mobile robot has been tested in the ground level of the Research Centre for Information and Communications Technologies of the University of Granada (CITIC-UGR). We have evaluated a wide range of paths and contours (straight walls of different lengths, followed and/or preceded by a number of concave and convex corners, gaps, etc.) which correspond to the usual situations that a mobile robot may face during normal navigation. The d_{wall} considered in this case study was 40 cm. We can see qualitatively how the robot follows a smooth trajectory despite the noise produced by ultrasound sensors. This noise is mainly due to the specular reflection caused by the surface material of the obstacles (wooden doors, skirting boards, etc.). Moreover, small bugs also appear due to the fact that the wheels can slip, the turned angle is not exactly the one measured by the motors of the robot, and so on. In spite of all these disturbances, the tested fuzzy controller got an average distance of 40.5 cm with a standard deviation of 3 cm and an average speed of 42.9 cm/s, thus exhibiting a good performance in the test environment. A demonstration video of this case study and additional complementary illustrative material can be found at the JFML website (http://www.uco.es/JFML/documentation).

4.2. Ventilation System for a Refrigerating Chamber

In this case study, an FLC has been designed to manage a ventilation system for a refrigerating chamber in order to show the use of the three communication mechanisms (WiFi, Bluetooth, and USB) and the two embedded systems (Arduino^{TM} and Raspberry Pi) available in the JFML. This kind of fuzzy controller is commonly considered in the food-processing industry due to the fact that the chilled and frozen food should be conserved in specific conditions to maintain their nutritional properties. Fuzzy controllers can preserve the original cooling conditions while avoiding abrupt changes in the environment. In this case, we have considered three input variables: the indoor temperature of the chamber (*temp*), the indoor humidity of the chamber (*hum*), and the opening rate of the door (*prox*). Notice

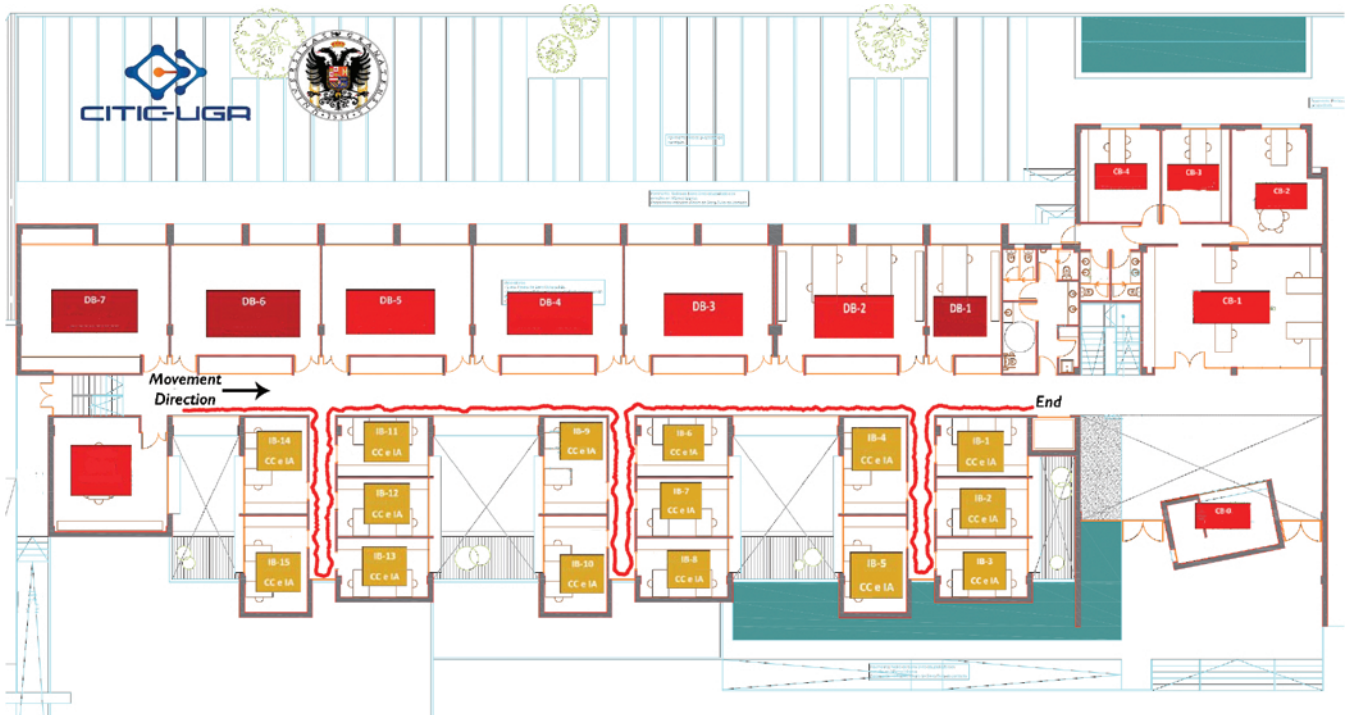

Figure 5 Test-bed environment of our mobile robot.

that the opening rate is measured as the distance from the door to the back wall of the chamber. The output variable is the power of the ventilation system (*power*). Table 3 shows the domain of each variable. The designed FLC is available as one of the examples provided with the software JFML (*./Examples/XMLFiles/Chamber.xml*).

Table 3 Variables in the ventilation system.

Input Var.	Domain	Output Var.	Domain
Temp (C)	[−40.0, 15.0]	Power (%)	[0.0, 100.0]
Hum (%)	[0.0, 100.0]		
Prox (cm)	[0.0, 200.0]		

Figure 6 shows the hardware architecture for this embedded system. It is made up of three modules (an Arduino UNO board, a NodeMCU, and a Raspberry Pi), two temperature and humidity sensors DHT22, a proximity sensor HCSR04, and a DC motor. The sensors DHT22 are connected to the NodeMCU and the Arduino UNO to measure the temperature and the humidity, respectively. The sensor HCSR04 is connected to the Raspberry Pi for measuring the proximity. In addition, the DC motor is connected to the Arduino UNO to control the FAN power of the chamber. Moreover, a remote personal computer (where JFML is installed) is used to manage the whole system in real-time. The JFML library assists us to run the related FLC and to infer the output values from the input values. The computer is connected to the NodeMCU via WiFi, to the Arduino Uno via USB, and to the Raspberry Pi via Bluetooth.

The steps that are required to configure and run the whole system are as follows: The first step is to read the description of the FLC from the FML/XML document. The second step is to create the sensors/actuators that are connected with the Arduino board.The third step is to map the input/output variables with the sensors/actuators

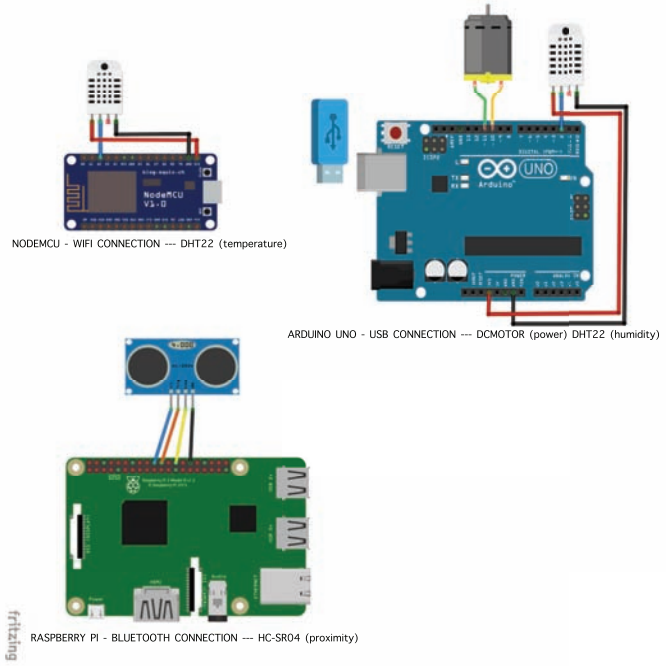

Figure 6 Hardware architecture for the ventilation system.

and to include them in a list. The fourth step is to create the embedded system (including the name, port, rate, etc.) and to generate the *.ino* file. The following Javacode carries out all these steps:

```
// First step: Loading the FLC from a XML file
File fml=new File("./Examples/XMLFiles/Chamber.xml");
FuzzyInferenceSystem fs = JFML.load(fml);

// Second step: Creating the sensors/actuators
KnowledgeBaseVariable temp, hum, prox, power;
temp = fs.getVariable("temperature");
hum = fs.getVariable("humidity");
prox = fs.getVariable("proximity");
power = fs.getVariable("power");

Sensor tempS, humS, proxS, powerS;
tempS = new ArduinoDHT22_temperature(temp.getName(),
ArduinoPin.PIN_D2);
humS = new ArduinoDHT22_humidity(hum.getName(),
ArduinoPin.PIN_2);
powerS = new ArduinoDCMOTOR_PWM(power.getName(),
ArduinoPin.PIN_10, ArduinoPin.PIN_11);
proxS = new RaspberryHC_SR04(prox.getName(),
RaspberryPin.GPIO25, RaspberryPin.GPIO7);

// Third step: Mapping sensors/actuators
ArrayList<EmbeddedVariable> eVarWIFI;
eVarWIFI = new ArrayList<>();
eVarWIFI.add(new EmbeddedVariableArduino
(0, temp, tempS));

ArrayList<EmbeddedVariable> eVarUSB;
eVarUSB = new ArrayList<>();
eVarUSB.add(new EmbeddedVariableArduino
(0, hum, humS));
eVarUSB.add(new EmbeddedVariableArduino
(1, power, powerS));

ArrayList<EmbeddedVariable> eVarBluetooth;
eVarBluetooth = new ArrayList<>();
eVarBluetooth.add(new EmbeddedVariableRaspberry
(0, prox, proxS));

// Fourth step: Creating the embedded systems
EmbeddedSystem arduinoWIFI;
arduinoWIFI = new EmbeddedSystemArduinoWifi
("arduinoWIFIDHT22TEMP", "IP_ADDRESS",
"SSID", "PASSWORD", eVarWIFI);
EmbeddedSystem arduinoUSB;
arduinoUSB = new EmbeddedSystemArduinoUsb
("arduinoUSB-DHT22HUM", "USB_PORT", 9600, eVarUSB);
EmbeddedSystem rpiBluetooth;
rpiBluetooth = new EmbeddedSystemRaspberryBluetooth
("rpiBluetooth-HCSR04","BLUETOOTH_PORT", 9600,
eVarBluetooth);

//Creating the .ino and .py files
arduinoWIFI.createRunnableEmbeddedFile
(fml.getName()+"WIFI");
arduinoUSB.createRunnableEmbeddedFile
(fml.getName()+"USB");
rpyBluetooth.createRunnableEmbeddedFile
(fml.getName()+"BLUETOOTH");
```

The generated *.ino* files have to be written in the corresponding Arduino boards with the IDE provided by the company. The *.py* file has to be executed by the command “python file.py” in the

Raspberry Pi. Finally, we can execute the controller that synchronizes all the embedded systems (Arduino UNO, NodeMCU and Raspberry Pi) and runs the FLC, in this case in the external computer. The following Java code corresponds to this final step:

```
ArrayList<EmbeddedSystem> boards;
boards = new ArrayList<>();
boards.add(arduinoWIFI);
boards.add(arduinoUSB);
boards.add(rpiBluetooth);

EmbeddedController controller;
controller=new EmbeddedControllerSystem(boards, fs);
controller.run();
```

Notice that the embedded systems are included in a list because we have used two Arduino boards and one Raspberry Pi in the design. The different communication mechanisms available in the module allow us to place the sensors over appropriate settings according to the system, obtaining relevant information to manage the system and with low noise from sensors. However, the low-cost components used in the hardware architecture have a certain degree of imprecision in this environment, therefore some parameters of the FLC may need to be tuned in order to obtain a good performance in unknown environments.

5. CONCLUSIONS

In this paper we present a new module of the open source JFML library. It allows to connect FLCs with two of the most used embedded systems in the open hardware model, Arduino™ and Raspberry Pi. The new JFML module maps the input/output variables of a FLC with a large set of sensors/actuators through these embedded systems in a completely independent way apart from the specific hardware/software constraints that usually characterize the architectures where such system is to be deployed. Moreover, this module integrates a new communication protocol between JFML and the embedded systems by means of several communication mechanisms (WiFi, Bluetooth, and USB). This communication protocol allows to perform the fuzzy inference in an external personal computer, overcoming the limited computing capacity that typically characterizes some embedded systems. Thanks to this protocol, an FLC can be distributed in computer network environments which is a key issue in real-time applications.

We have shown two case studies in order to illustrate the potential of the new JFML module: 1. a wall-following FLC defined according to the IEEE Std 1855-2016 to manage a mobile robot and 2. a FLC to manage a ventilation system for a refrigerating chamber.

We have also illustrated how to make use of the new module in a Java program. In detail, we can map the input/output variables of the FLC with the sensors and the actuators, generate runnable files for Arduino™ board and Raspberry Pi, and manage the developed systems by performing the fuzzy inference either locally in the embedded system or remotely in an external computer while communication can be established through several communication mechanisms (USB, Bluetooth, or WiFi). These communication mechanisms allow to place the embedded systems over appropriate settings for the system, and obtaining the information more relevant to manage the system. Moreover, sensor values can be grouped to generate high-level input variables that provide information that is more relevant and meaningful to model the systems under consideration.

JFML supports three of the most popular and widely used formats for FLCs (i.e., FCL, PMML, and Matlab FIS), making possible with this module to implement a wide range of FLCs designed with other software tools on Arduino™ and Raspberry Pi. Moreover, the open software and hardware models make it easier to share knowledge, promote trade through the open exchange of designs, improve the level of application of FLCs to industry, and to promote the teaching of basic computer science in schools and in developing countries.

As well as the whole JFML, this new module is continuously updated and improved. Currently, we are working on the incorporation of new embedded systems, sensors, and actuators, and on the development of a graphical user interface to provide a framework friendly and easy to use.

In addition, we consider that the IoT paradigm represents one the most active research trends in the area of computation. Therefore, we are mainly focusing our future research in this line, facing IoT challenges by extending opportunely the JFML modules to enable fuzzy programmers to approach these novel scenarios in a simple, intuitive, and direct way. In this sense, a very interesting application of this paradigm addresses the area of recommender systems, which in combination with IoT, are currently successful solutions for facilitating online access to the information that fits user preferences and needs in overloaded search spaces [36]. For example, housing design in safe, adaptive environments, integrated with technologies for senior citizens life support [37], or web personalization [38] are hot related challenges.

ACKNOWLEDGMENTS

This paper has been supported in part by the Spanish Ministry of Economy and Competitiveness (Projects TIN2017-89517-P, TIN2015-68454-R, TIN2017-84796-C2-1-R, and TIN2017-90773-REDT) and the Andalusian Government. In addition, Jose M. Alonso is Ramon y Cajal Researcher (RYC-2016-19802). Financial support from the Galician Ministry of Education (grants ED431F 2018/02, GRC2014/030 and accreditation 2016-2019, ED431G/08), co-funded by the European Regional Development Fund (ERDF/FEDER program), is also gratefully acknowledged.

REFERENCES

[1] L. Magdalena, Fuzzy rule-based systems, in: J. Kacprzyk, W. Pedrycz (Eds.), Springer Handbook of Computational Intelligence, Springer, 2015, pp. 203–218.
[2] R. Alcala, M.J. Gacto, J. Alcala-Fdez, Evolutionary data mining and applications: a revision on the most cited papers from the last 10 years (2007–2017), Wiley Interdiscip. Rev. Data Mining Knowledge Discov. 8(2) (2018), e1239.
[3] J. Alcalá-Fdez, R. Alcalá, S. Gonzalez, Y. Nojima, S. Garcia, Evolutionary fuzzy rule-based methods for monotonic classification, IEEE Trans. Fuzzy Syst. 25(6) (2017), 1376–1390.
[4] H. Zuo, G. Zhang, W. Pedrycz, V. Behbood, J. Lu, Fuzzy regression transfer learning in takagi-sugeno fuzzy models, IEEE Trans. Fuzzy Syst. 25(6) (2017), 1795–1807.

[5] G. Acampora, Fuzzy markup language: a XML based language for enabling full interoperability in fuzzy systems design, in: A. Giovanni, L. Vincenzo, L. Chang-Shing, W. Mei-Hui (Eds.), On the Power of Fuzzy Markup Language, Springer, Berlin, 2013, pp. 17–31.
[6] H. Zermane, H. Mouss, Internet and fuzzy based control system for rotary kiln in cement manufacturing plant, Int. J. Comput. Intell. Syst. 10(1) (2017), 835–850.
[7] S. El-Sappagh, J.M. Alonso, F. Ali, A. Ali, J.-H. Jang, K.-S. Kwak, An ontology-based interpretable fuzzy decision support system for diabetes mellitus diagnosis, IEEE Access. 6 (2018), 37371–37394.
[8] E. Aranda-Escolástico, M. Guinaldo, M. Santos, S. Dormido, Control of a chain pendulum: a fuzzy logic approach, Int. J. Comput. Intell. Syst. 9(2) (2016), 281–295.
[9] X. Xiang, C. Yu, L. Lapierre, J. Zhang, Q. Zhang, Survey on fuzzy-logic-based guidance and control of marine surface vehicles and underwater vehicles, Int. J. Fuzzy. Syst. 20(2) (2018), 572–586.
[10] G. Acampora, V. Loia, A. Vitiello, Distributing fuzzy reasoning through fuzzy markup language: an application to ambient intelligence, in: A. Giovanni, L. Vincenzo, L. Chang-Shing, W. Mei-Hui (Eds.), On the Power of Fuzzy Markup Language, Springer, Berlin, 2013, pp. 33–50.
[11] G. Acampora, B. di Stefano, A. Vitiello, IEEE 1855^{TM}: the first IEEE standard sponsored by IEEE Computational Intelligence Society, IEEE Comput. Intell. Mag. 11(4) (2016), 4–6.
[12] J.M Soto-Hidalgo, J.M. Alonso, G. Acampora, J. Alcala-Fdez, JFML: a java library to design fuzzy logic systems according to the IEEE std 1855-2016, IEEE Access. 6 (2018), 54952–54964.
[13] G. Acampora, A. Vitiello, Extending IEEE std 1855 for designing Arduino^{TM}-based fuzzy systems, in IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), July 2017, pp. 1–6.
[14] T. Noergaard, Embedded systems architecture: a comprehensive guide for engineers and programmers, Newnes, 2012.
[15] J. Alcalá-Fdez, J.M. Alonso, A survey of fuzzy systems software: taxonomy, current research trends and prospects, IEEE Trans. Fuzzy Syst. 24(1) (2016), 40–56.
[16] A.H. Altalhi, J.M. Luna, M.A. Vallejo, S. Ventura, Evaluation and comparison of open source software suites for data mining and knowledge discovery, Wiley Interdiscip. Rev. Data Mining Knowledge Discov. 7(3) (2017), e1204.
[17] J.M. Pearce, Open-Source Lab: How to Build Your Own Hardware and Reduce Research Costs, first ed., Elsevier, Amsterdam, 2013.
[18] Arduino Project, Open-source electronic prototyping platform enabling users to create interactive electronic objects, 2019. https://www.arduino.cc/.
[19] Raspberry Pi Foundation, Tiny and affordable computer that you can use to learn programming through fun, practical projects, 2018. https://www.raspberrypi.org/
[20] J.M. Alonso, C. Castiello, C. Mencar, The role of interpretable fuzzy systems in designing cognitive cities, in: E. Portmann, M.E. Tabacchi, R. Seising, A. Habenstein (Eds.), Designing Cognitive Cities. Studies in Systems, Decision and Control, Springer Nature Switzerland AG, 2019, pp. 131–152.
[21] M. Muñoz, E. Miranda, P. Sánchez, A fuzzy system for estimating premium cost of option exchange using mamdani inference: derivatives market of mexico. Int. J. Comput. Intell. Syst. 10(1) (2017), 153–164.
[22] B. Zhang, C. Yang, H. Zhu, P. Shi, W. Gui, Controllable-domain-based fuzzy rule extraction for copper removal process control, IEEE Trans. Fuzzy Syst. 26(3) (2018), 1744–1756.
[23] M. Barr, Embedded systems glossary, Neutrino Technical Library. Retrieved 21 April, 2007, https://barrgroup.com/Embedded-Systems/Glossary
[24] F. López-Rodríguez, F. Cuesta, Andruino-A1: low-cost educational mobile robot based on Android and Arduino, J. Intell. Robot. Syst. 81(1) (2016), 63–76.
[25] A. Aziz Khater, M. El-Bardini, N.M. El-Rabaie, Embedded adaptive fuzzy controller based on reinforcement learning for dc motor with flexible shaft, Arab. J. Sci. Eng. 40(8) (2015), 2389–2406.
[26] IEEE-SA Standards Board, IEEE standard for fuzzy markup language, IEEE Std. (2016), 1855–2016.
[27] M. Jayapriya, S. Yadav, A.R. Ram, S. Sathvik, R.R. Lekshmi, S. Selva Kumar, Implementation of fuzzy based frequency stabilization control strategy in Raspberry Pi for a wind powered microgrid, Procedia Comput. Sci. 115 (2017), 151–158.
[28] R.G.J. Anduray, S.M.Z. Irigoyen, Development of a fuzzy controller for liquid level by using raspberry pi and internet of things, in IEEE Central America and Panama Student Conference (CONESCAPAN 2017), Panama, 2017, pp. 1–5.
[29] M. Sujaritha, S. Annadurai, J. Satheeshkumar, S. Kowshik Sharan, L. Mahesh, Weed detecting robot in sugarcane fields using fuzzy real time classifier, Comput. Electron. Agr. 134 (2017), 160–171.
[30] International Electrotechnical Commission, Technical Committee Industrial Process Measurement and Control. IEC 61131-Programmable Controllers. IEC, 2000.
[31] A. Guazzelli, W.C. Lin T. Jena. PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics Paperback CreateSpace Independent, second ed., Publishing Platform, 2012.
[32] MathWorks, Fuzzy logic toolbox-r2017b. https://www.mathworks.com/products/fuzzy-logic.html, 2017.
[33] I. Rodríguez-Fdez, M. Mucientes, A. Bugarín, Learning fuzzy controllers in mobile robotics with embedded preprocessing, Appl. Soft Comput. 26 (2015), 123–142.
[34] K. Karvinen, T. Karvinen, Getting Started with Sensors: Measure the World with Electronics, Arduino, and Raspberry Pi, Maker Media, Incorporated, San Francisco, 2014.
[35] M. Mucientes, R. Alcalá, J. Alcalá-Fdez, J. Casillas, Learning weighted linguistic rules to control an autonomous robot, Int. J. Intell. Syst. 24(3) (2009), 226–251.
[36] R. Yera, L. Martínez, Fuzzy tools in recommender systems: a survey, Int. J. Comput. Intell. Syst. 10 (2017), 776–803.
[37] M. Zallio, D. Berry, N. Casiddu, Adaptive environments for enabling senior citizens: an holistic assessment tool for housing design and iot-based technologies, in IEEE 3rd World Forum on Internet of Things (WF-IoT), Dec. 2016, pp. 419–424.
[38] V. Salonen, H. Karjaluoto, Web personalization: the state of the art and future avenues for research and practice, Telemat. Inform. 33(4) (2016), 1088–1104.