IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 27, 2018, accepted September 11, 2018, date of publication September 28, 2018, date of current version October 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2872777

JFML: A Java Library to Design Fuzzy Logic Systems According to the IEEE Std 1855-2016

J. M. SOTO-HIDALGO[1], (Member, IEEE), JOSE M. ALONSO[2], (Member, IEEE), GIOVANNI ACAMPORA[3], (Senior Member, IEEE), AND J. ALCALA-FDEZ[4], (Member, IEEE)

[1]Department of Electronics and Computer Engineering, University of Córdoba, 14071 Córdoba, Spain
[2]Centro Singular de Investigación en Tecnoloxías da Información, University of Santiago de Compostela, 15782 Santiago, Spain
[3]Department of Physics Ettore Pancini, University of Naples Federico II, 80126 Naples, Italy
[4]DaSCI Research Institute, University of Granada, 18071 Granada, Spain

Corresponding author: J. M. Soto-Hidalgo (jmsoto@uco.es)

This work was supported in part by the XXII Own Research Program (2017) of the University of Córdoba, in part by the Spanish Ministry of Economy and Competitiveness under Grants RYC-2016-19802 (Ramón y Cajal contract), TIN2017-84796-C2-1-R, TIN2014-56633-C3-3-R, TIN2014-57251-P, and TIN2015-68454-R, in part by the Andalusian Government under Grant P11-TIC-7765, in part by the Xunta de Galicia (accreditation 2016–2019), and in part by the European Union (European Regional Development Fund).

ABSTRACT Fuzzy logic systems are useful for solving problems in many application fields. However, these systems are usually stored in specific formats and researchers need to rewrite them to use in new problems. Recently, the IEEE Computational Intelligence Society has sponsored the publication of the IEEE Standard 1855-2016 to provide a unified and well-defined representation of fuzzy systems for problems of classification, regression, and control. The main aim of this standard is to facilitate the exchange of fuzzy systems across different programming systems in order to avoid the need to rewrite available pieces of code or to develop new software tools to replicate functionalities that are already provided by other software. In order to make the standard operative and useful for the research community, this paper presents JFML, an open source Java library that offers a complete implementation of the new IEEE standard and capability to import/export fuzzy systems in accordance with other standards and software. Moreover, the new library has associated a Website with complementary material, documentation, and examples in order to facilitate its use. In this paper, we present three case studies that illustrate the potential of JFML and the advantages of exchanging fuzzy systems among available software.

INDEX TERMS Fuzzy logic systems, IEEE std 1855-2016, fuzzy markup language, open source software, IEC61131-7.

I. INTRODUCTION

Fuzzy Logic Systems (FLSs) [1], [2] are likely to be one of the most important applications of the fuzzy set theory. FLSs are an extension of the classical systems in which fuzzy sets and fuzzy rules are used instead of the classical ones. Thanks to their capacity to include a priori knowledge, to manage uncertainty and vagueness, and to represent systems for which is not possible to obtain a mathematical model, FLSs have been successfully applied to a wide range of problems such as classification, control or regression [3]–[6] and in different domain applications such as, for instance, mobile robot navigation [7], medical diagnosis [8], non-linear rotary chain pendulum [9], cement manufacturing plant [10], etc.

A high number of proposals on FLSs have been published in the literature, however few researchers publish the software and/or source code associated with their papers and many inconsistencies are reported in the codes published in journals and books. This issue, along with the high complexity of some proposals, prevents the widespread use of FLSs [11]. In recent years, many software tools have been developed in order to facilitate the use of FLSs to solve real-world applications. Although some of them are commercially distributed, such the Fuzzy Logic Toolbox for Matlab [12] or the Fuzzy Logic add-on for Mathematica [13], a high number of free and open source software has been also developed by the scientific community to work with FLSs, such as Juzzy [14], FuzzyLite [15], or Fispro [16]. Notice that the open source model makes it easier for researchers the application of FLSs to new problems, which is crucial to extend FLSs to other disciplines and industry [17].

Some of these software allow to read, design, and write Mamdani type-1 fuzzy logic controllers [18] according to the

part 7 of the standard IEC 61131 (IEC61131-7) published by the International Electrotechnical Commission [19]. This standard defines the Fuzzy Control Language (FCL) with the aim of offering companies and developers a well-defined common understanding of the basic means with which to integrate fuzzy control applications in control systems, as well as the possibility of exchanging portable fuzzy control programs across different programming systems. This standard has a world-wide diffusion and is independent of system manufactures, which has many advantages: easy migration to and from several hardware platforms from different manufacturers, protection of investment at both training and application levels, conformity with the requirements of the Machinery Directive EN60204, and reusability of the developed application.

Recently, the IEEE Computational Intelligence Society (IEEE-CIS) has sponsored the publication of the new standard for FLSs (IEEE Std 1855-2016) [20]. This standard defines a new W3C eXtensible Markup Language (XML)-based language, named Fuzzy Markup Language (FML) [21], that extends the advantages of IEC61131-7 to other types of problems (e.g., classification or regression) and includes four different types of type-1 fuzzy inference systems: Mamdani and Assilian [18], Takagi-Sugeno-Kang (TSK) [22], Tsukamoto [23], and AnYa [24]. The main aim of this standard is to allow the exchange of FLSs across different programming systems and software in order to avoid the need to rewrite available pieces of code or to develop new software tools to include functions or modules that are available in other ones (increasing the usability of the available software). Moreover, this standard provides several extension mechanisms that ensure the viability of the specification, allowing to include new elements such as type-2 and intuitionistic fuzzy sets, new membership functions, and so on, in the scheme of the standard. These extensions should be forwarded to the IEEE 1855 committee for potential inclusion in a future release.

In this paper we present JFML,[1] a new open source Java library that allows to design FLSs according to the IEEE Std 1855-2016 in order to make the standard operative and useful for the research community. This library offers a complete implementation of the four fuzzy inference systems enclosed in the W3C XML Schema definition (XSD) of the standard: Mamdani, TSK, Tsukamoto, and AnYa. Additionally JFML includes a module to import/export FLSs in accordance with FCL documents (the standard IEC 61131-7), the Predictive Model Markup Language (PMML) [25], [26] and the proprietary format understood by the Matlab Fuzzy Logic Toolbox.

JFML has been designed following the hierarchical structure based on the concept of labeled tree used in the definition of FML and it includes the extension methods considered in the standard, facilitating the integration of changes in future releases of the standard. This library makes use of the Java Architecture for XML Binding (JAXB) to bind the XSD of the standard and the Java representations, providing a fast and convenient way for reading (unmarshalling) and writing (marshalling) FLSs according to the standard. Moreover, the new library has associated a website in order to facilitate the download of complementary material and documentation. It also brings to the research community all the benefits derived from working with open source software. Thanks to the benefits provided by JFML and IEEE Std 1855-2016, the community will have an open source software tool available for designing and sharing FLSs without any additional porting task (hardware or software).

This paper is arranged as follows. Section II summarizes the main features of the new IEEE Std 1855-2016. Section III makes a brief review of other libraries available in the literature for designing FLSs. Section IV describes JFML, its architectural components. Section V shows how JFML can be used to design FLSs for problems from different areas. Finally, in Section VI, some concluding remarks are made.

II. IEEE STANDARD 1855-2016 FOR FUZZY MARKUP LANGUAGE

The IEEE Std 1855-2016 is the first standard sponsored by the IEEE-CIS [27]. It was approved by the Standards Board of the IEEE Standards Association on January 2016. This standard defines an W3C XML-based language called FML with the aim of offering developers a well-defined common understanding of the basic means with which to design FLSs for different types of problems that can be exchanged across different programming systems and software (improving their interoperability), avoiding the need of recoding available codes and of developing new software tools to replicate functions or modules that are available in other ones (improving the usability of the available software).

Let us summarize some issues related to the IEEE Std 1855-2016 which are needed to understand the rest of the manuscript.

1) *FML labeled trees*. The elements (e.g., variables and rules) of FLSs are represented by a connected acyclic graph with different levels of granularity. In this graph, nodes and edges are associated with labels which describe their properties. As it is illustrated in Fig. 1, there are three kinds of node: rectangles represent structural nodes; round rectangles correspond to attributes; and ellipses are text nodes. The root node is named ''FLS''. It is connected with other two nodes: ''KB'' depicts the knowledge base and ''RB'' depicts the rule base. In addition, KB child nodes are related to fuzzy variables while RB child nodes correspond to rules. Accordingly, FLSs are represented by labeled trees which fit the hierarchical nature of FML.
2) *FML schema*. This schema is represented in a XSD file and it can be visualized/interpreted as a labeled tree where all nodes have at least two attributes: (1) the node name; and (2) the network address. The attribute network address allows developers to represent a

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

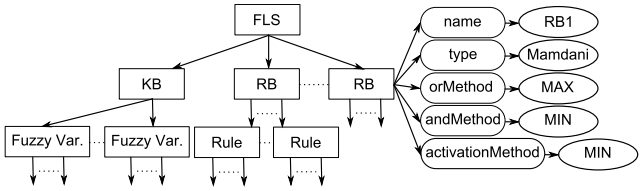


FIGURE 1. Example of FML labeled tree.

distributed FLSs in which the components can be located in different network addresses. For example, FLSs defined for applications in the Internet of Things.

3) *FML synthesis*. The standard suggests the use of eXtensible Stylesheet Language Translators for translating FLSs described in FML into general purpose programming languages such as Java or C/C++.
4) *FML conformance*. Two levels are considered:
   - *A strictly conforming FML* means the document which describes the FLS strictly complied with the FML schema. In consequence, it guarantees fully interoperability no matter the hardware platform or programming language considered.
   - *A conforming FML* admits the use of extensions, e.g., customized elements. In consequence, fully interoperability is not guaranteed.

The IEEE Standard 1855-2016 considers four type-1 FLSs: Mamdani [18], TSK [22], Tsukamoto [23], and AnYa [24]. Let us briefly introduce them. The interested reader is kindly referred to [28] for a thorough introduction to fuzzy sets and systems.

1) *Mamdani fuzzy systems*. The FLSs introduced by Zadeh [29] were first developed by Mamdani and Assilian [18]. IF-THEN rules relate n inputs and 1 output:

$$R_i: \; IF \; X_1 \; is \; A^i_{1,h} \; AND \; \ldots \; AND \; X_n \; is \; A^i_{n,m} \; THEN \; Y \; is \; B^i_o \tag{1}$$

The antecedent of rule R_i is a conjunction of linguistic propositions ($X_k \; is \; A^i_{k,j}$) where $k \in [1, n]$, and $A^i_{k,j}$ is the $j-th$ linguistic term defined for the $k-th$ input variable X_k. B^i_o is the $o-th$ linguistic term defined for the output Y. Given a universe of discourse U, each linguistic term is characterized by a fuzzy set $A = \{(x, \mu_A(x))|x \in U\}$, where $\mu_A(x) \in [0, 1]$ is the membership function (MF) which characterizes A in U. The Mamdani inference mechanism is usually called min-max because conjunction (*AND*) and implication (*THEN*) are normally implemented by the t-norm minimum, and the output accumulation is done by the maximum. Anyway, notice that Mamdani rules admit other operations, not only minimum and maximum. In addition, the defuzzification stage transforms an inferred fuzzy set into a single crisp value by means of a defuzzification method.

2) *Takagi-Sugeno-Kang fuzzy systems*. These FLSs are known as TSK systems [22]. The consequent of R_i (Eq. 1) is now a function f_i that for first order TSK is as follows:

$$f_i(X_1, \ldots, X_n) = p_1 X_1 + \cdots + p_n X_n + c \tag{2}$$

Accordingly, TSK systems can describe non-linear systems with a small number of rules. However, the interpretation of functional rule outputs is usually harder than the interpretation of fuzzy outputs in Mamdani systems.

3) *Tsukamoto fuzzy systems* [23]. These FLSs define fuzzy rules in a similar way to Mamdani except for the definition of the consequents where the use of monotone MFs is imposed. Thus, the inferred output of each rule is a crisp value determined by the conjunction of the antecedents. Moreover, the output aggregation and defuzzification stages are usually computed as in TSK systems.
4) *Angelov-Yager fuzzy systems*. These FLSs are called AnYa systems [24]. The rule base comprises a set of IF-THEN rules where the consequent part can be the same as in Mamdani or TSK systems. However, the antecedent part changes radically. Rule antecedents are represented by "Data Clouds" which are free of logical connectives, non-parametric and based on relative data density. Moreover, AnYa clouds are described by the similarity among the associated set of data samples. The membership degree to a cloud is interpreted by local and global density measures. The main advantage of AnYa systems is that they can be seen as networks of related data clouds.

This standard provides a wide variety of MFs and fuzzy operators, among other components (see [20] for more information). However, researchers may need to use other elements to design their FLSs that are not included in the current definition of the scheme. For example, other fuzzy sets (e.g., type-2 or intuitionistic), fuzzy inference mechanisms, new MF shapes, among others. In order to ensure the viability of the specification, this standard provides several extension mechanisms that allow to utilize extended files without errors or loss of functionality. This kind of extensions should be forwarded to the IEEE 1855 committee for potential inclusion in future releases of the standard.

III. RELATED WORK

There is well-known commercial software for designing FLSs, e.g., the Fuzzy Logic Toolbox for Matlab [12] or the Fuzzy Logic add-on for Mathematica [13]. However, the relevance of free and open source software is growing in the scientific research community [30].

In this section, we give a global overview on fuzzy systems software. We take as starting point the review on free and open source software that was previously carried out by the IEEE-CIS Task Force on Fuzzy Systems Software (FSS) of the Fuzzy Systems Technical Committee (FSTC) [31]. It is

FIGURE 2. **Overview on fuzzy systems software. On the top, statistics related to all types of software. On the bottom, details on libraries.**

worthy to note that this review came along with a thematic website[2] with complementary material. It is linked from the FSTC website[3] as the reference website for FSS. This website presents a two-level taxonomy with the aim of facilitating the review of existing software. The first level deals with the software purpose. Three categories are considered (General Purpose, Specific Applications, and Fuzzy Languages). For each of them, the second level provides four lists of software paying attention to the type of software (Code, Library, Toolbox, and Suite). Moreover, links to other related websites as well as a bibliography compilation of related papers are also provided. It is note worthy this website is maintained and periodically updated by the Chair and Vice-chair of the IEEE-CIS Task Force on FSS.

We have identified and analyzed 128 different pieces of software. The interested reader is kindly referred to the FSS website for further details on each single software. Let us summarize here the main results (see Fig. 2). Most software is released as toolboxes (60%) or libraries (26%). No matter the type of software (look at the three pie charts on the top of Fig. 2), Java (35%), C++ (19%), and Matlab (17%) are the main programming languages. However, regarding only libraries (pay attention to the bottom left pie chart in Fig. 2), the three most used languages are: Java, 37% (e.g., Juzzy [14], jFuzzyLite [15] or jFuzzyLogic [32]); C++, 18% (e.g., FuzzyLite [15] or FFLL [33]); and R, 15% (e.g., FRBS [34] or SAFD [35]).

With respect to the application purpose, most software is for general purpose (52%). This percentage is larger (70%) regarding only libraries. General purpose software allows to read, design, and write FLSs that can be applied to different problems in relation to all research areas addressed by the fuzzy community (e.g., clustering, classification, or regression). In addition, there is software ready to deal with different types of fuzzy sets (type-1 FLSs but also their extensions, e.g., type-2 or intuitionistic fuzzy sets). For example, Juzzy [14] is a Java library intended not only for type-1 FLSs, but also for Interval and General type-2 FLSs. Moreover, it is especially designed to take advantage of the parallel computing capabilities of Java architectures. IT2FLS [36]–[39] is a toolbox for Interval type-2 FLSs. The main advantages of this toolbox are the capacity to develop complex systems and the flexibility that permits the user to extend the availability of functions for working with Interval type-2 FLSs. On the other hand, paying attention to software for specific purpose, it is easy to appreciate how control and education are the main specific applications.

Some software (8%) follows a standard language (FCL or PMML [25], [26] which stands for Predictive Model Markup Language) to facilitate interoperability with other software. This percentage achieves 21% in case of considering only libraries. Actually, we have found six libraries (see Table 1 where they are listed from the newest to the oldest) which are ready to compile, read, design, and write FLSs in accordance with a standard language. Most of these libraries (4/6) are aimed for control applications and therefore they adopted FCL in accordance with the IEC 61131-7 norm.

As far as we know, pyfuzzy [41] was the first general purpose library, written in Python, for designing FLSs. It provides developers with a framework to work with fuzzy sets and process them with operations of fuzzy logic. Authors

[2]http://sci2s.ugr.es/es/fss
[3]http://cis.ieee.org/fuzzy-systems-tc.html

TABLE 1. Libraries for designing FLSs in accordance with a standard.

Name	Latest release	Language	Short description	Standard	Purpose
FuzzyLite [15]	2017	C++	A library for designing and operating fuzzy logic controllers without relying on external libraries	FCL	Control
jFuzzyLite [15]	2017	Java/Android	A clone of FuzzyLite for Java and Android platforms	FCL	Control
FRBS [34]	2015	R	A library for building FLSs for classification and regression tasks with operations of fuzzy logic	PMML	General
jFuzzyQt [40]	2015	C++/Qt	A clone of jFuzzyLogic	FCL	Control
jFuzzyLogic [32]	2014	Java	FCL and Fuzzy logic Application Programming Interface	FCL	Control
pyfuzzy [41]	2014	Python	A library for working with fuzzy sets and processing them	FCL	General

were aware of the need to provide interoperability with other software and they adopted FCL which was the only standard language for fuzzy systems available at that moment, even though it was thought only for control applications.

Later, Cingolani and Alcala-Fdez developed jFuzzyLogic [32]. It is a Java library to design fuzzy logic controllers in accordance with FCL. The latest release is available in GitHub to make easier maintenance and update. This library has partial code synthesis to C++ and can be parallelized. jFuzzyQt [40] is the clone of jFuzzyLogic in C++/Qt.

In addition, Rada-Vilela developed FuzzyLite [15] for designing fuzzy logic controllers in C++. This library is fully self-content (i.e., it does not rely on any external library) and carefully designed for effective and efficient operation, even with embedded control systems. It has multiple code synthesizers (including code conforming to the FCL standard) and parallelization capabilities. Notice that jFuzzyLite is the clone of FuzzyLite for Java and Android.

In the case of FRBS [34], authors chose PMML instead of FCL because their interest was more in machine learning tasks (mainly classification and regression) than in solving control problems. PMML is an XML-based predictive model interchange format which has become the *de facto* standard for data mining and machine learning algorithms (e.g., logistic regression and feed-forward neural networks). FRBS is an R package which implements various learning algorithms based on fuzzy rule-based systems for dealing with classification and regression tasks. It also allows to construct fuzzy models defined by human experts.

Nowadays, there is not any software ready to read, design, and write FLSs in accordance with the IEEE Standard 1855-2016. As we have previously explained, this standard extends the advantages of the IEC61131-7 norm to other types of problems apart from control (e.g., classiification or regression) with the aim of allowing the exchange of FLSs across different programming systems and software, avoiding the need of recoding or developing new pieces of software to replicate functionalities already provided by other software. Based on these requirements, we have developed the library JFML. In the next section we will describe JFML in detail.

IV. JFML

This library is developed in Java. This language is one of the most used in the development of open source fuzzy software (see Section III) and enables JFML to be used on the most used platforms and operating system since Java Virtual Machine is platform independent. JFML follows a strict object-oriented approach and a modular design based on the same labeled tree structure that FML uses to represent FLSs, allowing developers to extend JFML without changing the language grammar. This software is distributed as open source software under the terms of the GNU Public License GPLv3 and it is hosted in the public hosting GitHub,[4] which offers different tools (e.g., bug tracker or mailing) in order to make use of the advantages of the open source model. Moreover, JFML has a web page associated[5] with a complete documentation and a good variety of examples. In the following, we will explain the main characteristics of JFML: main classes, binding with the FML XSD, extensibility, and interoperability with other software.

A. JFML CORE

FML-based FLSs are generated from a set of semantic tags. They allow representing the components of a classical FLS making use of a labeled tree structure, in which each XML tag, XML attribute and value of an attribute are represented by an element, attribute and text node, respectively. The relations between XML elements and attributes are represented by the connections of the tree (see Fig. 1, section II). JFML follows strictly the FML XSD. Fig. 3 shows the main class diagram of this library, which will be explained below in order to provide the reader with an overview of the possibilities of the library.

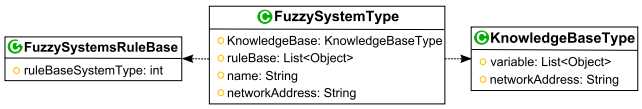


FIGURE 3. Main class diagram of JFML.

1) FuzzySystemType CLASS

FuzzySystemType is the top-level class of the class diagram and it allows representing the four FLSs enclosed in the scheme of the standard: Mamdani, Tsukamoto, TSK and AnYa. Each object of this class has a unique name and a network address to define the location of the FLS in a computer network system (this attribute is also included in other

[4] https://github.com/sotillo19/JFML
[5] http://www.uco.es/JFML

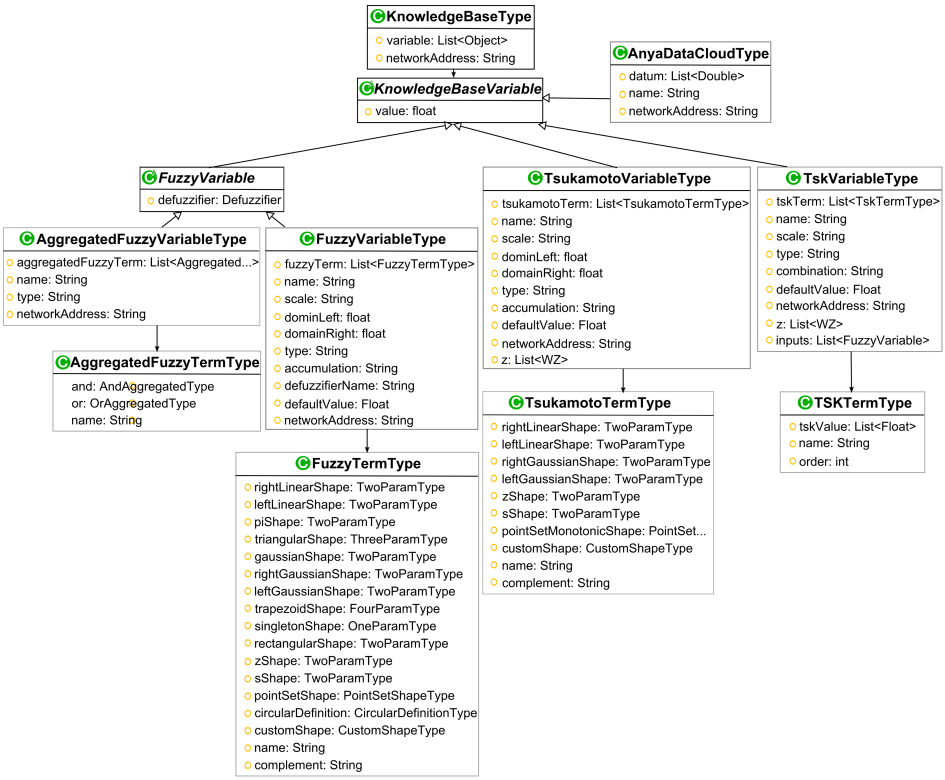


FIGURE 4. Main class diagram for the definition of a KB.

subclasses with the same aim), and consists of one element *KnowledgeBase* and of a list of elements *FuzzySystemRuleBases* that represent the KB and the RBs of the system, respectively. Notice that, a FLS can contain more than one RB and they can be of different type. By default, RBs of the system are evaluated in the order in which they are located in the list but this order can be modified. Accordingly, we can define FLSs in different sub-hierarchies.

2) KnowledgeBaseType CLASS

This class represents the KB of the FLS, containing the definition of the input and output variables of the system. Each object of this class consists of a list with one or several elements of the class *KnowledgeBaseVariable*, which represents a variable of the system in an abstract way. This class only contains the basic operations that we can apply to all the variables, while the definition of each variable and its specific operations are delegated to the subclasses (see Fig. 4). They are introduced as follows.

The *FuzzyVariableType* class represents a fuzzy variable that can be part of the antecedent or consequent of the rules in Mamdani RBs. It can also be part of the antecedent of the rules in Tsukamoto and TSK RBs, and part of the consequent of the rules in AnYa RBs. Each object of this class contains information about the domain, type (input or output), the scale used to measure the variable, the default value for this variable when no rule has been fired, the accumulation (also called combination in the standard), defuzzification methods used when this variable is involved in the consequent of the rules, and a list with the linguistic terms of the variable.

The *TsukamotoVariableType* class represents an output variable which can be part of the consequent of the rules in Tsukamoto RBs. The objects of this class contain the same information as the *FuzzyVariableType* objects, but the membership functions of their linguistic terms can only be monotone functions.

The *TskVariableType* class represents an output variable which can be part of the consequent of the rules in TSK RBs. Each object contains information about the type (input or output), the scale used to measure the variable, the default value for this variable when no rule has been fired, the accumulation method, and a list of objects of the *TskTermType*. Each *TskTermType* object can be involved in the consequent of TSK rules and it represents a constant value (zero-order TSK system) or a linear function of the inputs (one-order TSK system).

The *AnYaDataCloudType* class represents a data cloud that can be part of the antecedent of the rules in AnYa RBs. Each object contains a list of datums that represents a sub-set of previous data samples with common properties. Notice that, each datum is defined as only one double value in the standard but it can be extended to consider datum objects with more values.

The *AggregatedFuzzyVariable* class represents a new fuzzy variable in which linguistic terms are defined by means of the aggregation of linguistic terms of other variables. Each object of the class contains a list of *AggregatedFuzzyTerm* objects,

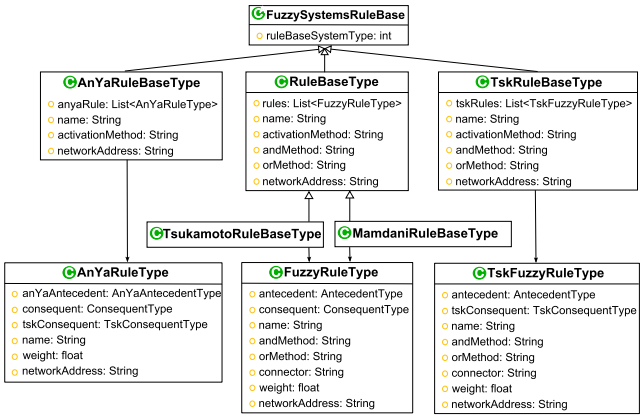


FIGURE 5. Main class diagram for the definition of RBs.

which makes use of AND (a t-norm) and OR (a t-conorm) operators to generate the new terms of the variable from two or more terms of other variables. This class allows defining variables and rules in a flexible way.

3) FuzzySystemsRuleBase CLASS

This class represents a RB of the FLS in an abstract way. It only contains the basic operations that we can apply to all the type of rules, while the definition of the rule sets and its specific operations are delegated to the subclasses (see Fig. 5). They are introduced as follows.

The *RuleBaseType* class represents a RB in which the consequent and antecedent of the rules only have fuzzy variables. Each object of this class contains information about the activation method, the AND and OR methods used by default for all rules, and a list with the fuzzy rules. This class is extended by the classes *MamdaniRuleBaseType* and *TsukamotoRuleBaseType* to generate RBs with Mamdani and Tsukamoto rules, respectively. Both of them contain a list of *FuzzyRuleType* objects. Notice that the rules of the *MamdaniRuleBaseType* object involve *FuzzyVariableType* variables in the antecedent and consequent, and the rules of the *TsukamotoRuleBaseType* objects involve *FuzzyVariableType* variables in the antecedent and *TsukamotoVariableType* variables in the consequent in order to assure the use of monotone membership functions in the consequent of the rules.

The *TskRuleBaseType* class represents a TSK RB. The objects of this class contain the same information as the *RuleBaseType* objects together with a list of *TskFuzzyRuleType* objects. Each *TskFuzzyRuleType* object involves *FuzzyVariableType* variables in the antecedent and *TskVariableType* variables in the consequent of the rules.

The *AnYaRuleBaseType* class represents AnYa RBs. Each object of this class contains the activation method and a list of *AnYaRuleType* objects. Each *AnYaRuleType* object involves *AnYaDataCloudType* variables that represent the data cloud used in the antecedent, and *FuzzyVariableType* or *TskVariableType* variables in the consequent of the rules. Notice that, this type of rules can use the same consequent as the Mamdani and TSK rules (see section II).

Each antecedent element in the rules (*AntecedentType*) contains one or more clause connected through AND/OR operators. For each rule, we can select the specific connector along with the related t-norm/t-conorm. On the other hand, the consequent elements (*ConsequentType* and *TskConsequentType*) contain one element THEN, representing the THEN-part of a rule, and an optional element ELSE, representing the ELSE-part of a rule. Both elements can also contain one or multiple clauses, thus allowing the representation of FLSs with multiple outputs. Moreover, we can indicate a weight for each rule to define the importance of the rule in the inference.

It should be noted that JFML implements all the t-norms (7), t-conorms (7), accumulation methods (7), defuzzification methods (4), membership functions (14), and activation methods (7) defined in the standard.

B. BINDING WITH THE FML XSD

The W3C XML-based nature of FML allows to convert the FML description of FLSs into source code (e.g., in C/C++, Java or Python) making use of any library for XML processing. For example, developers can use the eXtensible Stylesheet Language Translators (XSLTs)[6] to design parsers that allow to convert FML descriptions into source programs for different programming systems. To accomplish this, JFML uses the Java Architecture for XML Binding (JAXB) that allows developers to access and process W3C XML data without having to know about W3C XML or W3C XML processing. JAXB includes a binding compiler to bind the XSD for a FML document into the set of Java classes for JFML and provides a package for performing operations such as unmarshalling (reading), marshalling (writing), and validation. When an unmarshalling operation is performed, a tree of instances of the Java classes produced by the binding compiler is created from an XML document. Let us show in an illustrative example how to unmarshal a FML document using JFML:

```
// Creating a new file Java object
File xml = new File("./XMLFiles/TipperMamdani1.xml");

// Loading a FLS from an FML document
FuzzyInferenceSystem tipper = JFML.load(xml);
```

A marshalling operation is the opposite to unmarshalling. This operation creates an FML document from a tree of instances of the Java classes. Let us consider a simple example of how to marshal a developed FLS into a FML document:

```
// Creating a new file Java object
File tipper = new File("./XMLFiles/TipperMamdani1.xml");

// Writting the developed FLS into an FML document
JFML.writeFSTtoXML(tipper, tipperXMLFile);
```

C. EXTENSIBILITY

The language defined in the standard can be extended using custom methods (called according to the pattern

[6] https://www.w3.org/Style/XSL/

custom_name) to include new values for attributes (e.g., new fuzzy operators or defuzzification methods) without requiring to modify the language grammar. JFML includes custom methods for all the elements indicated in the XSD of the standard, enabling a way to extend the library conforming to this standard. In order to show a simple example, JFML includes the center of gravity singleton as a custom defuzzification method (called *custom_COGS*) for the *FuzzyVariableType* class. Moreover, the modular design of JFML based on the same labeled tree structure as FML allows to accommodate future changes in the standard design modifying only the corresponding part in the library.

D. IMPORT/EXPORT MODULES

JFML includes a module with classes to read FLSs from FCL or PMML documents, and to write a FLS designed with JFML into a FCL or PMML document. Moreover, this library also includes a module to read FLSs designed with the Matlab Fuzzy Logic Toolbox and to export FLSs designed with JFML to the Matlab toolbox. The aim of these modules is to make easier the interoperability of the library with other well-established software, such as jFuzzylogic, FuzzyLite, pyfuzzy, FRBS, among others. Notice that these classes extend the classes *Import* and *Export*, which represents import/export parsers in an abstract way. Since these classes only contain the most common and basic operations of a parser, they must be extended in case of developers requiring more complex parsers to import/export FLSs in other formats.

V. CASE STUDIES

Users and developers can find several examples for problems of classification (for the problems Iris and Tao), regression (for the problems Inverted Pendulum, Tipper and Japanese Diet Assessment) and control (for a fuzzy controller to manage a mobile robot) in the folders *src/jfml/test* and *Examples* provided with the JFML package and in the associated website. For each example, the website provides: the Java code needed to design the related FLSs in JFML; the Java code to read FLSs from FML documents; the Java code to make inference with a new input sample; and the FML documents related to the FLSs designed with JFML.

This section shows three case studies that illustrates the potential of JFML and the advantages of the exchange of FLSs across different software available. The first case is focused on the development of a FLS for the Tipper problem. The second case shows how developers can import and use a fuzzy controller for mobile robotics defined according to the standard IEC 61131-7. The last case is focused on a fuzzy rule-based classification system (FRBCS), i.e. a FLS for classification, for the Tao problem.

A. TIPPER FUZZY SYSTEM

The *Tipper* problem consists of computing the right tip in a restaurant. It considers two inputs (food and service) and one output (tip). In the rest of this section we explain how to build and evaluate a Mamdani FLS with JFML for this problem.

Firstly, we create an empty KB:

```
FuzzyInferenceSystem tipper =
      new FuzzyInferenceSystem("tipper - MAMDANI");
KnowledgeBaseType kb = new KnowledgeBaseType();
tipper.setKnowledgeBase(kb);
```

Secondly, we define the inputs and output. Food is characterized by 2 linguistic terms (rancid, delicious) in the range [0, 10]. Service comprises 3 linguistic terms (poor, good, excellent) in the same range. Tip is defined by 3 linguistic terms (cheap, average, generous) in the range [0, 20]. For example, the ''service'' variable is created and added to the KB as follows:

```
FuzzyVariableType service =
    new FuzzyVariableType("service", 0, 10);
FuzzyTermType poor = new FuzzyTermType("poor",
    FuzzyTermType.TYPE_leftGaussianShape,
    (new float[]  0f, 2f ));
service.addFuzzyTerm(poor);
FuzzyTermType good = new FuzzyTermType("good",
    FuzzyTermType.TYPE_gaussianShape,
    (new float[]  5f, 2f ));
service.addFuzzyTerm(good);
FuzzyTermType excellent = new FuzzyTermType("excellent",
    FuzzyTermType.TYPE_rightGaussianShape,
    (new float[]  10f, 2f ));
service.addFuzzyTerm(excellent);
kb.addVariable(service);
```

The definition of the output variable includes setting fuzzy operators in addition to defining linguistic terms:

```
FuzzyVariableType tip = new FuzzyVariableType("tip",0,20);
tip.setAccumulation("MAX");
tip.setDefuzzifierName("COG");
tip.setType("output");
```

Then, we can create a rule base which relates inputs and output. For example, the rule ''**IF** food is rancid AND service is very poor **THEN** tip is cheap'' with rule weight (RW) equals 1.0 is created as follows:

```
MamdaniRuleBaseType rb = new MamdaniRuleBaseType("rb1");
FuzzyRuleType rule1 =
   new FuzzyRuleType("rule1", "or", "MAX", 1.0f);
AntecedentType ant1 = new AntecedentType();
ant1.addClause(new ClauseType(food, rancid));
ant1.addClause(new ClauseType(service, poor, "very"));
rule1.setAntecedent(ant1);
ConsequentType con1 = new ConsequentType();
con1.addThenClause(tip, cheap);
rule1.setConsequent(con1);
rb.addRule(rule1);
```

Then, the ''tipper'' system can be written in an XML document according to the IEEE std 1855-2016 making use of the marshal methods provided by the JAXB API. It asserts that the document is valid according to the XSD of the standard:

```
File tipperXML=new File("./XMLFiles/TipperMamdani1.xml");
JFML.writeFSTtoXML(tipper, tipperXML);
```

Finally, we can assign values to the input variables, evaluate the system, and print results with the following code:

```
KnowledgeBaseVariable input1 = fs.getVariable("food");
KnowledgeBaseVariable input2 = fs.getVariable("service");
input1.setValue(2);
input2.setValue(1);
fs.evaluate();
KnowledgeBaseVariable output =  fs.getVariable("tip");
```

```
float value = output.getValue();

System.out.println("RESULTS");
System.out.println(" (INPUT): "
    +input1.getName() + "="+input1.getValue()
    +", "+input2.getName() + "="+input2.getValue());
System.out.println("(OUTPUT):"+output.getName()+"="+value);
System.out.println(fs.toString());
```

The related outcome shown in standard output stream is as follows:

```
RESULTS
 (INPUT): food=2.0, service=1.0
 (OUTPUT): tip=7.2561355

*food - domain[0.0, 10.0] - input
   rancid -  triangular [a: 0.0, b: 2.0, c: 5.5]
   delicious -  rightLinear [a: 5.5, b: 10.0]
*service - domain[0.0, 10.0] - input
   poor -  leftGaussian [c: 0.0, sigma: 2.0]
   good -  gaussian [c: 5.0, sigma: 4.0]
   excellent -  rightGaussian [c: 10.0, sigma: 2.0]
*tip - domain[0.0, 20.0]
     - Accumulation:MAX; Defuzzifier:COG - output
   cheap -  triangular [a: 0.0, b: 5.0, c: 10.0]
   average -  triangular [a: 5.0, b: 10.0, c: 15.0]
   generous -  triangular [a: 10.0, b: 15.0, c: 20.0]
RULEBASE:
  *mamdani - rulebase1: OR=MAX; AND=MIN; ACTIVATION=MIN
RULE 1: rule1 - (1.0)
  IF food IS rancid OR service IS very poor
     THEN tip IS cheap [weight=1.0]
...
(the rest of the rules are omitted for the sake of space)
```

Moreover, an excerpt of the XML document generated according to the IEEE std 1855-2016 is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<fuzzySystem xmlns="http://www.ieee1855.org"name="tipper-
  MAMDANI">
 <knowledgeBase>
  <fuzzyVariable name="food" scale="" domainleft="0.0"
    domainright="10.0" type="input">
    <fuzzyTerm name="rancid" complement="false">
      <triangularShape param1="0.0" param2="2.0"
        param3="5.5"/>
    </fuzzyTerm>
    <fuzzyTerm name="delicious" complement="false">
      <rightLinearShape param1="5.5" param2="10.0"/>
    </fuzzyTerm>
  </fuzzyVariable>
...
(the rest of the document is omitted for the sake of space)
```

B. ROBOT FUZZY CONTROL SYSTEM

In mobile robotics, fuzzy controllers are commonly considered for producing the so-called 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 while the robot moves as fast as possible, avoiding abrupt changes in the trajectory movements and velocity. In [32], the authors designed a wall-following fuzzy controller according to the standard IEC 61131-7. It is made up of four input variables (distances to the right (*RD*) and left walls (*DQ*); orientation regarding to the wall (*O*); and velocity (*V*)) and two output variables (linear acceleration (*LA*) and angular velocity (*AV*)). The corresponding file (robot.fcl) is available as one of the examples provided with the software jFuzzyLogic.

FIGURE 6. The robot moving in a corridor.

Figure 6 shows the robot we used in the experiments. In order to use with JFML the same fuzzy controller previously developed by jFuzzyLogic, the first step is to import the controller by means of the FCL parser provided by the *ImportFCL* class:

```
FuzzyInferenceSystem fs;
ImportFCL fcl = new ImportFCL();
fs=fcl.importFuzzySystem("./XMLFiles/robot.fcl");
```

Then, we can assign values to the input variables and evaluate the FLS, dually as it was done in the previous case study. Moreover, the imported FLS can be easily stored into an XML file as follows:

```
File xmlFile = new File("./XMLFiles/RobotMamdani.xml");
JFML.writeFSToXML(fs, xmlFile);
```

The corresponding Java code to import, to evaluate and to marshal the wall-following controller (RobotImportFCL.java) together with the generated XML file (RobotMamdani.xml) are available in the folders Test and XMLFiles of the JFML package, respectively.

Finally, let us explain how JFML can evaluate a FML-based FLS through the command line. Developers have to run the JFML.jar file with the following arguments: the XML file that contains the FLS description, the number of output variables that they want to infer, the names of the output variables, and the list of pairs (input variable name / data value). Then, JFML returns a summary of the tasks carried out: (1) loading the FML file; (2) setting the input values; and (3) running the fuzzy inference. In addition, the library prints a brief description of the FLS, regarding both the knowledge base and the rule base, including the activation degree for each rule. The reader can find below an illustrative example of how to evaluate RobotMamdani.xml file through the command line with the inputs *RD*=0.2, *DQ*=0.25, *O*=20, and *V*=0.25:

```
java -jar JFML.jar RobotMamdani.xml $2$~la av rd 0.2
dq 0.25~o 20~v 0.25

1) Loading Fuzzy System from an XML file according to
the standard IEEE 1855
2) Setting input variables: rd=0.2, dq=0.25, o=20.0, v=0.25
3) Making fuzzy inference
RESULTS
 (INPUT): rd=0.2, dq=0.25, o=20.0, v=0.25
 (OUTPUT): la=0.034830566~av=0.054393604
4) Fuzzy System Description
FUZZY SYSTEM: robotIEEE1855
KNOWLEDGEBASE:
```

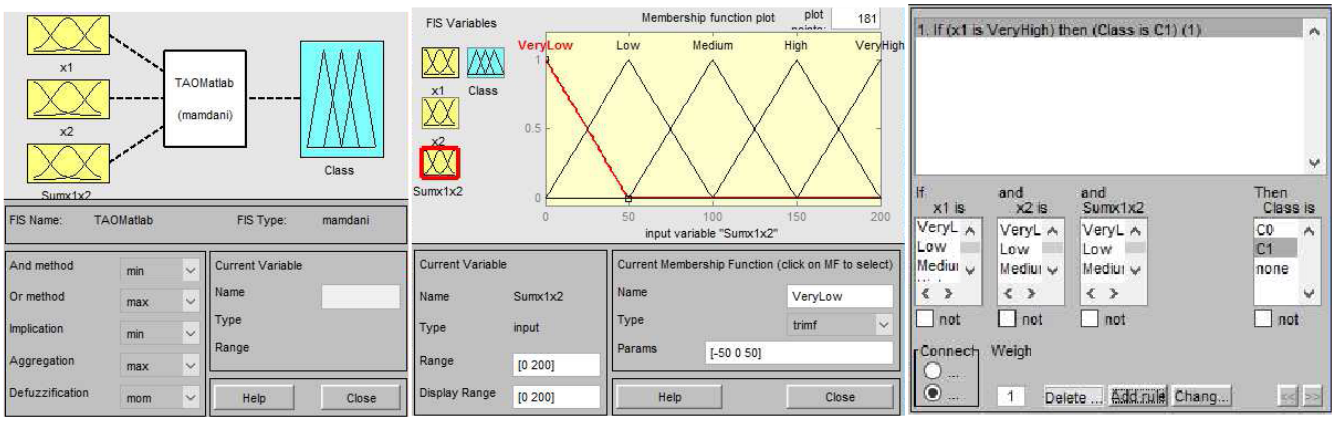


FIGURE 7. Design of the FLS for the database Tao with the Fuzzy Logic Toolbox in Matlab.

```
  *rd - domain[0.0, 3.0] - input
L -  triangular [a: 0.0, b: 0.0, c: 1.0]
M -  triangular [a: 0.0, b: 1.0, c: 2.0]
...(the rest of the variables are omitted by space)
RULEBASE:
  *mamdani - rulebase1: OR=MAX; AND=MIN; ACTIVATION=MIN
RULE 1: 01 - (0.0) IF rd ...
... (the rest of the rules are omitted by space)
```

C. TAO FUZZY SYSTEM

Here, we consider the FLS for classification which was developed in [42] for the database Tao [43]. This is a synthetic problem with two input variables (x_1 and x_2) and two classes (C_0 and C_1). Moreover, a new input variable was created from the addition of the two input variables ($Sum(x_1, x_2)$). This FLS uses homogeneous fuzzy partitions with five triangular fuzzy sets (Very Low, Low, Medium, High and Very High) for the three input variables. With respect to the fuzzy reasoning mechanism, the minimum t-norm plays the role of both the AND operator and the accumulation method, and maximum t-conorm plays the role of both the OR operator and the activation method. The RB of the FLS consists of three Disjunctive Normal Form (DNF) rules with weights, which are defined as follows:

- R_1: If x_1 is (Very High) Then Class is C_1 (Weight = 1)
- R_2: If Sum(x_1, x_2) is (High or Very High) Then Class is C_1 (Weight = 0.884)
- R_3: If x_1 is (Very Low or Low or Medium) and Sum(x_1, x_2) is (Very Low or Low) Then Class is C_0 (Weight = 0.908)

Let us consider that a user prefers to use the Fuzzy Logic Toolbox for Matlab to design this FLS. To do this, a new Mamdani Fuzzy Inference System (FIS) is created with the AND and OR operators, the accumulation method, and the activation method used in the FLS. Then, the three input variables are added to the DB, including for each one the five triangular fuzzy sets that were considered in the design. Finally, the rules are added to the RB regarding the MFs previously defined for each input variable. Notice that the rules R_2 and R_3 can not be added to the RB. This is due to the fact that Matlab toolbox does not allow us to select several labels (connected by the OR operator in this case) for a variable selected for the antecedent of the rule. Figure 7 shows the FIS design for the database Tao with the Matlab toolbox.

In order to add the rules R_2 and R_3 to the RB, the FLS generated with the Matlab toolbox is exported to a ".fis" file to complete the design of the RB with JFML. To do this, the first step is to create a new Java program in which the library JFML is imported at the beginning. The second step is to load in the program the designed FLS from the ".fis" file making use of the import options provided by JFML. The third step is to create the missing terms related to the variables x_1 and $Sumx_1x_2$ that are required for rules R_2 and R_3. Notice that the standard IEEE Std 1855-2016, and therefore also the library JFML, provides an option called *Circular Definition* that allows to define new MFs from those ones previously defined for the terms of the variable. Then, the new terms are created and added to the list of terms of the variables. Finally, the fourth step is to generate the rules R_2 and R_3 and add them to the RB. Figure 8 shows the Java code used to implement the whole procedure described above and how to evaluate new input values and write the FLS description to a FML document.

This FLS can be used in the future by creating a new Java program that reads the system from the FML document with the unmarshalling option of JFML, assigns values to the input variables and evaluates the system. Moreover, JFML allows us to evaluate the FLS written in the FML document through the command line. Developers have to run the JFML.jar file with the following arguments: the FML document, the number of output variables, the name of the output variable, and the list of pairs (input variable name / data value).

JFML will return a summary of the tasks carried out. For instance, the following command can be used to evaluate the FLS represented in the *tao.xml* file with the inputs $x1 = 90$, $x2 = 60$, and $Sumx1x2 = 150$:

```
java -jar JFML.jar tao.xml 1 Class x1 90~x2 60~Sumx1x2 150
```

```
// SECOND STEP: Loading a FLS from a Matlab .fis file
FuzzyInferenceSystem fs = (new ImportMatlab()).importFuzzySystem("./XMLFiles/taoMatlab.fis");

// THIRD STEP: Creating new linguistic terms
KnowledgeBaseVariable x1, x2, Sumx1x2, C;  CircularDefinitionType mf1, mf2, mf3;
FuzzyTermType t1, t2, t3; x1 = fs.getVariable("x1"); x2 = fs.getVariable("x2");
Sumx1x2 = fs.getVariable("Sumx1x2"); C = fs.getVariable("Class");

mf1 = new CircularDefinitionType (
new OrLogicalType ("MAX", "VeryLow", new OrLogicalType("MAX", "Low", "Medium")), x1);
t1 = new FuzzyTermType("VLLM", mf1); ((FuzzyVariableType)x1).addFuzzyTerm(t1);

mf2 = new CircularDefinitionType (
new OrLogicalType ("MAX", "High", "VeryHigh"), Sumx1x2);
t2 = new FuzzyTermType("HVH", mf2); ((FuzzyVariableType)Sumx1x2).addFuzzyTerm(t2);

mf3 = new CircularDefinitionType (
new OrLogicalType ("MAX", "VeryLow", "Low"), Sumx1x2);
t3 = new FuzzyTermType("VLL", mf3); ((FuzzyVariableType)Sumx1x2).addFuzzyTerm(t3);

// FOURTH STEP: Adding rules Rule 2 and Rule 3 to the RB
MamdaniRuleBaseType rb = (MamdaniRuleBaseType) fs.getRuleBase(0);
// Rule 2
FuzzyRuleType R2 = new FuzzyRuleType("r2", "and", "min", 0.884f);
AntecedentType ant2 = new AntecedentType();  ConsequentType cons2 = new ConsequentType();
ant2.addClause(Sumx1x2, "HVH");  cons2.addThenClause(C, "C1");
R2.setAntecedent(ant2);  R2.setConsequent(cons2);   rb.addRule(R2);
// Rule 3
FuzzyRuleType R3 = new FuzzyRuleType("r3", "and", "min", 0.908f);
AntecedentType ant3 = new AntecedentType();  ConsequentType cons3 = new ConsequentType();
ant3.addClause(x1, "VLLM");  ant3.addClause(Sumx1x2, "VLL");  cons3.addThenClause(C, "C0");
R3.setAntecedent(ant3);  R3.setConsequent(cons3);  rb.addRule(R3);

// EVALUATING new input values
x1.setValue(90);  x2.setValue(60);  Sumx1x2.setValue(150);
fs.evaluate();
System.out.println("(OUTPUT): "+C.getName()+"="+C.getValue());

// WRITTING the FLS to a FML document
JFML.writeFSTtoXML(fs, new File("./tao.xml"));
```

FIGURE 8. Java code for the implementation of the FLS developed in [42] for the database Tao [43] in JFML.

VI. CONCLUSIONS

In this paper we have presented JFML, a new open source Java library with license GPLv3 ready to design and to use FLSs according to the IEEE Std 1855-2016. It offers a complete implementation of the four fuzzy inference systems enclosed in the W3C XML Schema definition (XSD) of the standard: Mamdani, TSK, Tsukamoto, and AnYa. JFML has been designed following the hierarchical structure based on the concept of labeled tree used in the definition of FML and it includes the extension methods considered in the standard, facilitating the integration of changes in future releases of the standard. Additionally JFML includes a module to import/export FLSs in accordance with FCL documents, the Predictive Model Markup Language (PMML) and the proprietary format understood by the Matlab Fuzzy Logic Toolbox.

JFML uses the Java architecture JAXB to provide a fast and convenient way for reading and writing FLSs according to the standard. Moreover, JFML has a website associated with a complete documentation and a good variety of examples. In addition, it is hosted in the public hosting GitHub which offers tools such as bug tracker or mailing. Notice that GitHub looks with favor on exploiting all the advantages of the open source model. JFML provides the research community with a well-defined tool for designing and sharing fuzzy systems without any additional, hardware and/or software, porting task.

Three detailed case studies have been described in order to illustrate the potential of JFML and the advantages of JFML to exchange FLSs across different software. We have first designed a FLS for the well-known tipper regression problem. Then, we have designed a FLS for controlling the wall-following behavior of a robot. Finally, we designed a preliminary FLS for classification with the Matlab Fuzzy Logic Toolbox and then we used JFML to enhance the design and evaluate unknown input values. Accordingly, any available software that allows reading an FML document can also read this system to perform other tasks. This mechanism for exchanging information increases the usability of the available software and facilitates getting FLSs into widespread real-world usage.

JFML is continuously updated and improved. We are developing a graphic user interface as a new module in order to facilitate the design and analysis of FLSs using JFML. Finally, on the basis of the distributed computing capabilities proposed in FML (i.e., a FLS can be distributed in computer network environments), we will face Internet of Things (IoT) challenges and we will extend opportunely the JFML library

to enable fuzzy programmers to approach these novel scenarios in a simple and direct way.

ACKNOWLEDGMENT

The authors want thank you to the community around JFML, whose feedback and support have been of great importance towards improving the library. The support of all the members of the IEEE-CIS Task Force on Fuzzy Systems Software is especially acknowledged. They actively support all activities carried out by the Task Force, being JFML one of the most recent outcomes.

REFERENCES

[1] D. J. Dubois, *Fuzzy Sets and Systems: Theory and Applications* (Mathematics in Science and Engineering), vol. 144. New York, NY, USA: Academic, 1980.

[2] D. Dubois and H. Prade, *Fuzzy Sets and Systems: Theory and Applications*. New York, NY, USA: Academic, 1980.

[3] X. Gu, F.-L. Chung, H. Ishibuchi, and S. Wang, "Imbalanced TSK fuzzy classifier by cross-class Bayesian fuzzy clustering and imbalance learning," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 8, pp. 2005–2020, Aug. 2017.

[4] Y. Li, S. Sui, and S. Tong, "Adaptive fuzzy control design for stochastic nonlinear switched systems with arbitrary switchings and unmodeled dynamics," *IEEE Trans. Cybern.*, vol. 47, no. 2, pp. 403–414, Feb. 2017.

[5] M. J. Gacto, M. Galende, and R. Alcalá, and F. Herrera, "METSK-HD^e: A multiobjective evolutionary algorithm to learn accurate TSK-fuzzy systems in high-dimensional and large-scale regression problems," *Inf. Sci.*, vol. 276, pp. 63–79, Aug. 2014.

[6] R. Alcalá, M. Gacto, and J. Alcalá-Fdez, "Evolutionary data mining and applications: A revision on the most cited papers from the last 10 years (2007–2017)," *Wires Data Mining Knowl. Discovery*, vol. 8, no. 2, p. e1239, 2018.

[7] X. Xiang, C. Yu, L. Lapierre, J. Zhang, and Q. Zhang, "Survey on fuzzy-logic-based guidance and control of marine surface vehicles and underwater vehicles," *Int. J. Fuzzy Syst.*, vol. 20, no. 2, pp. 572–586, 2018.

[8] S. El-Sappagh, J. M. Alonso, F. Ali, A. Ali, J.-H. Jang, and K.-S. Kwak, "An ontology-based interpretable fuzzy decision support system for diabetes mellitus diagnosis," *IEEE Access*, vol. 6, pp. 37371–37394, 2018.

[9] E. Aranda-Escolástico, M. Guinaldo, M. Santos, and S. Dormido, "Control of a chain pendulum: A fuzzy logic approach," *Int. J. Comput. Intell. Syst.*, vol. 9, no. 2, pp. 281–295, 2016.

[10] H. Zermane and H. Mouss, "Internet and fuzzy based control system for rotary kiln in cement manufacturing plant," *Int. J. Comput. Intell. Syst.*, vol. 10, no. 1, pp. 835–850, 2017.

[11] H. Thimbleby, "Explaining code for publication," *Softw., Pract. Exper.*, vol. 33, no. 10, pp. 975–1001, 2003.

[12] MathWorks. (2017). *Fuzzy Logic Toolbox—R2017b*. [Online]. Available: https://www.mathworks.com/products/fuzzy-logic.html

[13] Wolfram. (2005). *Fuzzy Logic*. [Online]. Available: https://reference.wolfram.com/legacy/v4/AddOns/Applications_FuzzyLogic.html

[14] C. Wagner, "Juzzy—A Java based toolkit for type-2 fuzzy logic," in *Proc. IEEE Symp. Adv. Type-2 Fuzzy Log. Syst. (T2FUZZ)*, Apr. 2013, pp. 45–52.

[15] J. Rada-Vilela. (2017). *Fuzzylite: A Fuzzy Logic Control Library*. [Online]. Available: http://www.fuzzylite.com

[16] S. Guillaume and B. Charnomordic, "Learning interpretable fuzzy inference systems with FisPro," *Inf. Sci.*, vol. 181, no. 20, pp. 4409–4427, 2011. [Online]. Available: http://www.inra.fr/mia/M/fispro/

[17] (1998). *Open Source Initative*. [Online]. Available: http://www.opensource.org/docs/osd

[18] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Man-Mach. Studies*, vol. 7, pp. 1–13, Jan. 1975.

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

[20] *IEEE Standard for Fuzzy Markup Language*, Standard 1855-2016, 2016, pp. 1–89. [Online]. Available: https://standards.ieee.org/findstds/standard/1855-2016.html.

[21] G. Acampora, "Fuzzy Markup Language: A XML based language for enabling full interoperability in fuzzy systems design," in *On the Power of Fuzzy Markup Language*, G. Acampora, V. Loia, C.-S. Lee, and M.-H. Wang, Eds. Berlin, Germany: Springer, 2013, pp. 17–31.

[22] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. SMC-15, no. 1, pp. 116–132, Feb. 1985.

[23] Y. Tsukamoto, "An approach to fuzzy reasoning method," in *Advances in Fuzzy Set Theory and Applications*, M. M. Gupta, R. K. Ragade, and R. R. Yager, Eds. Amsterdam, The Netherlands: North Holland, 1979, pp. 137–149.

[24] P. Angelov and R. Yager, "Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density," in *Proc. IEEE Workshop Evolving Adapt. Intell. Syst. (EAIS)*, Apr. 2011, pp. 62–69.

[25] A. Guazzelli, M. Zeller, W.-C. Lin, and G. Williams, "PMML: An open standard for sharing models," *R J.*, vol. 1, no. 1, pp. 60–79, 2009.

[26] A. Guazzelli, W.-C. Lin, T. Jena, and J. Taylor, *PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics*, 2nd ed. New York, NY, USA: Space Independent Publishing Platform, 2012.

[27] G. Acampora, B. Di Stefano, and A. Vitiello, "IEEE 1855: The first ieee standard sponsored by ieee computational intelligence society," *IEEE Comput. Intell. Mag.*, vol. 11, no. 4, pp. 4–7, Nov. 2016.

[28] E. Trillas and L. Eciolaza, *Fuzzy Logic: An Introductory Course for Engineering Students*. New York, NY, USA: Springer, 2015.

[29] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-3, no. 1, pp. 28–44, Jan. 1973.

[30] S. Sonnenburg *et al.*, "The need for open source software in machine learning," *J. Mach. Learn. Res.*, vol. 8, pp. 2443–2466, Oct. 2007.

[31] J. Alcalá-Fdez and J. M. Alonso, "A survey of fuzzy systems software: Taxonomy, current research trends, and prospects," *IEEE Trans. Fuzzy Syst.*, vol. 24, no. 1, pp. 40–56, Feb. 2016. [Online]. Available: http://sci2s.ugr.es/fss/

[32] P. Cingolani and J. Alcalá-Fdez, "jFuzzyLogic: A robust and flexible Fuzzy-Logic inference system language implementation," *Int. J. Comput. Intell. Syst.*, vol. 6, no. 1, pp. 61–75, 2013. [Online]. Available: http://jfuzzylogic.sourceforge.net/html/index.html

[33] M. Zarozinski, "An open source fuzzy logic library," in *AI Game Programming Wisdom*. Newton Center, MA, USA: Charles River Media, 2002, pp. 90–103. [Online]. Available: http://ffll.sourceforge.net/

[34] L. S. Riza, C. Bergmeir, F. Herrera, and J. M. Benítez, "FRBS: Fuzzy rule-based systems for classification and regression in R," *J. Stat. Softw.*, vol. 65, no. 6, pp. 1–30, 2015. [Online]. Available: http://www.jstatsoft.org/v65/i06/

[35] W. Trutschnig and A. Lubiano. (2015). *SAFD: Statistical Analysis of Fuzzy Data, R Package Version 1.0-1*. [Online]. Available: https://CRAN.R-project.org/package=SAFD

[36] J. R. Castro, O. Castillo, and P. Melin, "An interval type-2 fuzzy logic toolbox for control applications," in *Proc. IEEE Int. Fuzzy Syst. Conf.*, London, U.K., Jul. 2007, pp. 1–6.

[37] J. R. Castro, O. Castillo, P. Melin, L. G. Martínez, S. Escobar, and I. Camacho, "Building fuzzy inference systems with the interval type-2 fuzzy logic toolbox," in *Analysis and Design of Intelligent Systems Using Soft Computing Techniques*, P. Melin, O. Castillo, E. Ramírez, J. Kacprzyk, and W. Pedrycz, Eds. Berlin, Germany: Springer, 2007, pp. 53–62.

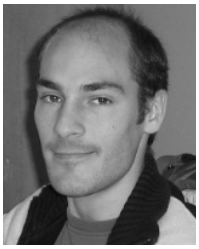
[38] J. Castro, O. Castillo, P. Melin, and A. Rodríguez-Díaz, "Building fuzzy inference systems with a new interval type-2 fuzzy logic toolbox," in *Transactions on Computational Science I*, M. Gavrilova and C. Tan, Eds. Berlin, Germany: Springer, 2008, pp. 104–114.

[39] O. Castillo, P. Melin, and J. Castro, "Computational intelligence software for interval type-2 fuzzy logic," *Comput. Appl. Eng. Educ.*, vol. 21, no. 4, pp. 737–747, 2013.

[40] (2015). *jFuzzyQt—C++ Fuzzy Logic Library*. [Online]. Available: http://jfuzzyqt.sourceforge.net/

[41] (2014). *PyFuzzy—Python Fuzzy Package*. [Online]. Available: http://pyfuzzy.sourceforge.net/
[42] D. García, A. González, and R. Pérez, "A two-step approach of feature construction for a genetic learning algorithm," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Taipei, Taiwan, Jun. 2011, pp. 1255–1262.
[43] E. Bernadó, X. Llorà, J. M. Garrell, "XCS and GALE: A comparative study of two learning classifier systems on data mining," in *Advances in Learning Classifier Systems*, P. Lanzi, W. Stolzmann, and S. Wilson, Eds. Berlin, Germany: Springer, 2002, pp. 115–132.

J. M. SOTO-HIDALGO received the M.S. and Ph.D. degrees in computer science from the University of Granada, Spain, in 2004 and 2014, respectively. Since 2007, he has been a member with the Department of Computer Architecture, Electronics and Electronic Technology, University of Córdoba, Spain, where he is currently an Associate Professor. He has authored over 50 papers in international journals and conferences. His research interests include softcomputing applied to image processing and data sensors management.



JOSE M. ALONSO received the M.S. and Ph.D. degrees in telecommunication engineering from the Technical University of Madrid, Spain, in 2003 and 2007, respectively. He is currently a Post-Doctoral Researcher with the Research Centre in Information Technologies, University of Santiago de Compostela. He has published over 85 papers in international journals, book chapters, and peer-review conferences. His research interests include the development of free software tools, interpretable fuzzy modeling, natural language generation, knowledge extraction and representation, integration of expert and induced knowledge, data science, and big data. He is the Secretary of the European Society for Fuzzy Logic and Technology. He has been the Chair of the IEEE-CIS Task Force on Fuzzy Systems Software since 2018, and an Associate Editor of the *IEEE Computational Intelligence Magazine*.

GIOVANNI ACAMPORA received the master's and Ph.D. degrees in computer science with the University of Salerno in 2003 and 2007, respectively. He is currently an Associate Professor in artificial intelligence and quantum computing with the University of Naples Federico II and the Working Group Chair of the IEEE 1855 Working Group, IEEE Standards Association. He was the Chair of Standards Committee at the IEEE Computational Intelligence Society, the Hoofddocent Tenure Track in Process Intelligence at the Eindhoven University of Technology, the Secretary and Treasurer at the IEEE CIS Italian Chapter, a Research Fellow at the University of Salerno, the Chair of the Task Force on Terminology and Taxonomy IEEE CIS Standards Committee, IEEE Computational Intelligence Society, and a Professor at UniPegaso. His papers include using FML and fuzzy technology in adaptive ambient intelligence environments, diet assessment based on type-2 fuzzy ontology and fuzzy markup language, A Survey on Ambient Intelligence in Health Care, Fuzzy control interoperability and scalability for adaptive domotic framework, Interoperable and adaptive fuzzy services for ambient intelligence applications, and A hybrid evolutionary approach for solving the ontology alignment problem.

J. ALCALÁ-FDEZ received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, Spain, in 2002 and 2006, respectively. From 2005 to 2007, he was with the Department of Computer Science, University of Jaèn. He is currently an Associate Professor with the Department of Computer Science and Artificial Intelligence, University of Granada, where he is also a member of the Soft Computing and Intelligent Information Systems Research Group. He has been the Chair of the Software Fuzzy Systems Task Force and the Fuzzy Systems Technical Committee, IEEE Computational Intelligence Society, since 2011. He has published over 60 papers in international journals, book chapters, and conferences.