Rapid development of fuzzy logic control using jFuzzyLogic

Pablo Cingolani [1], Jesús Alcalá-Fdez [2]

[1] *School of Computer Science, McGill University,*
McConnell Engineering Bldg, Room 318,
Montreal, Quebec, H3A-1A4, Canada
E-mail: pablo.cingolani@mail.mcgill.ca

[2] *Department of Computer Science and Artificial Intelligence, University of Granada,*
Research Center on Information and Communications Technology,
C/ Periodista Daniel Saucedo Aranda s/n,
Granada, 18071, Spain
E-mail: jalcala@decsai.ugr.es

Abstract

This work introduces jFuzzyLogic, a software framework that allows for rapid development of fuzzy systems. JFuzzyLogic's goal to facilitate and accelerate development of fuzzy systems is achieved by i) using standard programming language that reduces learning curves; ii) providing a fully functional and complete implementation of fuzzy inference system; iii) creating an API that developers can use or extend; iv) implementing an Eclipse plugin to easily write and test FCL code; v) making the software platform independent; and vi) distributing the software as open source. The use of jFuzzyLogic is illustrated through the analysis of one case study.

Keywords: List of four to six keywords which characterize the article.

1. Introduction

Fuzzy rule based systems (FRBSs) are one of the most important areas for the application of the Fuzzy Set Theory[1]. Classical rule based systems deal with IF-THEN rules. FRBSs constitute an extension to classical systems, having antecedents and consequents composed of fuzzy logic statements.

A Fuzzy Logic Controller (FLC) [2,3,4,5] is a FRBS composed of: i-) a Knowledge Base that comprises the information used by the expert operator in the form of linguistic control rules; ii-) a Fuzzification Interface, that transforms the crisp values of the input variables into fuzzy sets; iii-) an Inference System, that uses the fuzzy values from the Fuzzification Interface and the information from the Knowledge Base to perform the reasoning process and iv-) the Defuzzification Interface, which takes the fuzzy action from the Inference System and translates it into crisp values for the control variables.

FLCs are suitable for engineering applications in which classical control strategies do not achieve good results or when it is too difficult to obtain a mathematical model. FLCs usually have two characteristics: the need for human operator experience, and a strong non linearity. Many real-world applications use FLCs [6] such as mobile robot navigation [7,8], air conditioning controllers [9,10], domotic

control[11,12], and industrial applications[13,14].

FLCs are powerful for solving a wide range of problems, but their implementation requires a certain programming expertise. In the last few years, many fuzzy logic software tools have been developed to reduce this task. Some are commercially distributed, for example MATLAB Fuzzy logic toolbox(www.mathworks.com), while a few are available as open source software (see section 2).

In this work, we introduce an open source Java library named jFuzzyLogic. This fuzzy systems library allows FLCs design and implementation, following the standard for Fuzzy Control Language (FCL) published by the International Electrotechnical Commission (IEC 61131-7)[15].

The main goal of jFuzzyLogic is to bring the benefits of open source software and standardization to the fuzzy systems community. Our library offers several advantages:

- Standardization, which reduces programming work and learning curve. This library contains the basic programming elements for the Standard IEC 61131-7, alleviating developers from boiler plate programming tasks.
- Extensibility, the object model and API allows to create a wide range of applications. This is of special interest for the research community.
- Platform independence, allows to develop and run on any hardware and operating system configuration that supports Java.

This work is arranged as follows. The next section presents a comparison on non-commercial fuzzy software and the main benefits that the jFuzzyLogic offers with respect to other libraries. Section 3 introduces the concepts of IEC standard (IEC-61131) control programming languages. Section 4 describes jFuzzyLogic's main features. Section 6, illustrates how jFuzzyLogic can be used in a control application. Conclusions are presented in Section 7.

2. Comparison of fuzzy logic software

In this section we present a comparison on non-commercial fuzzy software (Table 1). We center our interest on free software distributions because of its important role in the scientific research community [16]. Moreover, we do not want to establish a comparison among all software tools or to emphasize the advantages of one over another. Our objective is to detect the major differences in the software and then to categorize jFuzzyLogic as an alternative to these suites when other research requirements are needed.

We analyze twenty five packages (including jFuzzyLogic), mostly from SourceForge or GoogleCode, which are considered to be some of the most respectable software repositories. The packages are analyzed in the following categories:

- *FCL support.* Only four packages ($\sim 17\%$) claim to support IEC 61131-7 specification. Notably two of them are based on jFuzzyLogic. Only two packages that support FCL are not based on our software. Unfortunately neither of them seem to be maintained by their developers any more. Furthermore, one of them has some code from jFuzzyLogic.
- *Programming language.* This is an indicator of code portability. There languages of choice were mainly Java and C++/C (column *Lang.*). Java being platform independent has the advantage of portability. C++ has an advantage in speed and also allows easier integration in industrial controllers.
- *Functionality.* Seven packages ($\sim 29\%$) were made for specific purposes, marked as 'specific' (column *Notes*, Table 1). Specific code usually has limited functionality, but it is simpler and has a faster learning curve for the user.
- *Membership functions*. This is an indicator of how comprehensive and flexible the package is. Specific packages include only one type of membership function (typically trapezoid) and/or one defuzzification method (data not shown). In some cases, arbitrary combinations of membership functions are possible. These packages are marked with asterisk. For example, '$M + N^*$' means that the software supports M membership functions plus another N which can be arbitrarily combined.

- *Latest release.* In eight cases ($\sim$ 33%) there were no released files for the last three years or more (see *Rel.* column in the Table 1). This may indicate that the package is no longer maintained, and in some cases the web site explicitly mentions this.
- *Code availability and usability.* Five of the packages ($\sim$ 21%) had no files available, either because the project was no longer maintained or because the project never released any files at all. Whenever the original sites were down, we tried to retrieve the projects from alternative mirrors. In three cases ($\sim$ 13%) the packages did not compile. We performed minimal testing by just following the instructions, if available, and make no effort to correct any compilation problems.

In summary, only eight of the software packages ($\sim$ 33%) seemed to be maintained, compiled correctly, and had extensive functionality. Only two of them (FuzzyPLC and jFuzzyQt) are capable of parsing FCL (IEC-61131-7) files and both are based on jFuzzyLogic.

3. IEC-61131 Languages

The IEC-61131 norm is well known for defining the Programmable Controller Languages (PLC), commonly used in industrial applications. In the part 7, this standard offers a well defined common understanding of the basic means to integrate fuzzy control applications in control systems. It also defines a common language to exchange portable fuzzy control programs among different platforms.

The specification defines six programming languages: Instruction list (IL), Structured text (ST), Fuzzy Control Language (FCL), Ladder diagram (LD), Function block diagram (FBD), and Sequential function chart (SFC). While IL, ST, and FCL are text based languages, LD, FBD and SFC are graphic based languages.

IL is similar to assembly language: one instruction per line, low level and low expression commands. ST, as the name suggests, intends to be more structured and it is very easy to learn and understand for anyone with a modest experience in programming. The focus of this work is FCL, which has a syntax is similar to ST and this is oriented to fuzzy logic based control systems.

3.1. IEC Language concepts

All IEC-61131 languages are modular. The basic module is called Programmable Organization Unit (POU) and includes Programs, Functions or Function Blocks. A system is usually composed of many POUs, and each of these POUs can be programmed in a different language. For instance, in a system consisting of two functions and one function block (three POUs), one function may be programed in LD, another function in IL and the function block may be programmed in ST. The norm defines all common data types (e.g. BOOL, REAL, INT, ARRAY, STRUCT, etc.) as well as ways to interconnect POUs, assign process execution priorities, process timers, CPU resource assignment, etc.

The concepts of a Program and Functions are quite intuitive. Programs are simple set of statements and variables. Functions are calculations that can return only one value and are not supposed to have state variables.

A Function Block resembles a very primitive object. It can have multiple input and multiple output variables, can be enabled by an external signal, and can have local variables. Unlike an object, a function block only has one execution block (i.e. there are no methods). The underlying idea for these limitations is that you should be able to implement programs using either text-based or graphic-based languages. Having only one execution block, allows to easily control execution when using graphic-based language to interconnect POUs.

3.2. FCL Language concepts

Fuzzy Control Language is an industry standard specification released by the International Electrotechnical Commission (IEC) as part of the Programmable Controller Languages (PLC) defined in the IEC-61131 specification.

At first glance FCL is similar to ST language described in the previous sections. However, there are some very important differences. FCL uses exclusively a new POU type: Fuzzy Inference Sys-

Table 1: Comparisson on open fuzzy logic software packages. Columns describe: Project name (Name), IEC 61131-7 language support (IEC), latest release year (Rel.), main programming language (Lang.), short description form website (Description), number of membership functions supported (MF) and Functionality (notes). Name* : package is maintained, compiles correctly, and has extensive functionality.

Name	IEC	Rel.	Lang.	Description	MF	Notes
Akira	No	2007	C++	Framework for complex AI agents.	4	
AwiFuzz	Yes	2008	C++	Fuzzy logic expert system	2	Does not compile
DotFuzzy	No	2009	C#	.NET library for fuzzy logic	1	Specific
FFLL	Yes	2003	C++	Optimized for speed critical applications.	4	Does not compile
Fispro*	No	2010	C++/Java	Fuzzy inference design and optimization	6	
FLUtE	No	2004	C#	A generic Fuzzy Logic Engine	1	Beta version
FOOL	No	2002	C	Fuzzy engine	5	Does not compile
FRBS	No	2011	C++	Fuzzy Rule-Based Systems	1	Specific
funzy	No	2007	Java	Fuzzy Logic reasoning	2*	Specific
Fuzzy Logic Tools*	No	2011	C++	Framework fuzzy control systems,	12	
FuzzyBlackBox	No	-	-	Implementing fuzzy logic	-	No files released
FuzzyClips	No	2004	C/Lisp	Fuzzy logic extension of CLIPS	3 + 2*	No longer maintained
FuzzyJ ToolKit	No	2006	Java	Fuzzy logic extension of JESS	15	No longer maintained
FuzzyPLC*	Yes	2011	Java	Fuzzy controller for PLC Siemens s226	11 + 14*	Uses jFuzzyLogic
GUAJE*	No	2011	Java	Development environment		Uses FisPro
javafuzzylogicctrltool	No	-	Java	Framework for fuzzy rules	-	No files released
JFCM	No	2011	Java	Fuzzy Cognitive Maps (FCM)	-	Specific
JFuzzinator	No	2010	Java	Type-1 Fuzzy logic engine	2	Specific
jFuzzyLogic*	Yes	2011	Java	FCL and Fuzzy logic API	11 + 14*	This paper
jFuzzyQt*	Yes	2011	C++	jFuzzyLogic clone	8	
libai	No	2010	Java	AI library, implements some fuzzy logic	3	Specific
libFuzzyEngine	No	2010	C++	Fuzzy Engine for Java	1	Specific
nxtfuzzylogic	No	2010	Java	For Lego Mindstorms NXT	1	Specific
Octave FLT*	No	2011	Octave	Fuzzy logic for Toolkit	11	
XFuzzy3*	No	2003	Java	Development environment	6	Implements XFL3 specification language

tem (FIS) which is a special case of a Function Block. All fuzzy language definitions should be within a FIS. Since a fuzzy system is inherently parallel, there is no concept of execution order, therefore there are no statements. For instance, there is no way to create the typical "Hello world" example since there is no *print* statement. A simple example of a FIS using FCL is shown bellow, which calculates the tip in a restaurant (this trivial example is the equivalent to a "Hello world" program for fuzzy systems). Figure 1 shows the membership functions.

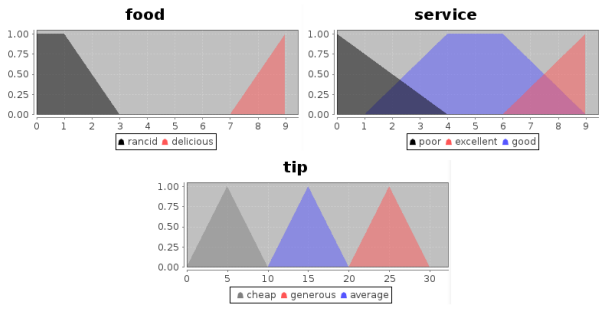

Fig. 1. Membership functions for tipper example.

```
FUNCTION_BLOCK tipper
VAR_INPUT
  service, food : REAL;
END_VAR

VAR_OUTPUT
  tip : REAL;
END_VAR

FUZZIFY service
  TERM poor := (0, 1) (4, 0) ;
  TERM good := (1, 0) (4,1) (6,1) (9,0);
  TERM excellent := (6, 0) (9, 1);
END_FUZZIFY

FUZZIFY food
  TERM rancid := (0, 1) (1, 1) (3,0);
  TERM delicious := (7,0) (9,1);
END_FUZZIFY

DEFUZZIFY tip
  METHOD : COG; // Center of Gravity
  TERM cheap := (0,0) (5,1) (10,0);
  TERM average := (10,0) (15,1) (20,0);
  TERM generous := (20,0) (25,1) (30,0);
END_DEFUZZIFY

RULEBLOCK tipRules
  Rule1: IF service IS poor OR food IS rancid
           THEN tip IS cheap;
  Rule2: IF service IS good THEN tip IS average;
  Rule3: IF service IS excellent AND food IS delicious
           THEN tip IS generous;
END_RULEBLOCK
END_FUNCTION_BLOCK
```

A FIS inference system is usually composed of one or more Function Blocks (FB). Every

FUNCTION_BLOCK has the following sections: i) input and output variables are define in VAR_INPUT and VAR_OUTPUT sections respectively; ii) fuzzification and defuzzification membership functions defined in FUZZIFY and DEFUZZIFY sections respectively; iii) fuzzy rules are written in the RULEBLOCK section.

Variable definition sections are straightforward, the variable name, type and possibly a default value are specified.

Membership functions either in FUZZIFY or DEFUZZIFY are defined for each linguistic term using the TERM statement followed by a function definition. In the previously shown example, functions are defined as piece-wise linear functions using series of points $(x_0,y_0)(x_1,y_1)...(x_n,y_n)$. For instance `TERM average := (10,0) (15,1) (20,0)` defines a triangular membership function as shown (in blue) at the bottom of Figure 1. Only two membership functions are defined in the IEC standard: singleton and piece-wise linear. As we shown in Section 4, jFuzzyLogic significantly extends these concepts.

A FIS can contain one or more RULEBLOCK, where fuzzy rule sets are defined. Since rules are intrinsically parallel, no execution order is implied or warranted by the specified order in the program. Each rule is defined using standard "`IF condition THEN conclusion [WITH weight]`" clauses. The optional `WITH weight` statement allows weighting factors for each rule. Conditions tested in each IF clause are of the form "variable IS [NOT] linguistic_term". This test membership of *variable* to a *linguistic_term* using the membership function defined in the corresponding FUZZIFY block. An optional NOT operand negates the membership function (i.e. $\overline{m}(x) = 1 - m(x)$). Obviously, several conditions can be combined using AND and OR connectors.

4. JFuzzyLogic

JFuzzyLogic's main goal is to facilitate and accelerate development of fuzzy systems. We achieve this goal by: i) using standard programming language (FCL) that reduces learning curves; ii) providing a fully functional and complete implementation of fuzzy inference system (FIS); iii) creating a programming interface (API) that developers can use or extend; iv) implementing an Eclipse plugin to easily write and test FCL code; v) making the software platform independent. and vi) distributing the software as open source. This allows to significantly accelerate development and testing of fuzzy systems in both industrial and academic environments.

In these sections we show how these design and implementation goals were achieved. This should be particularly useful for developers and researchers looking to extend the functionality or use the available API.

4.1. jFuzzyLogic Implementation

jFuzzyLogic is fully implemented in Java, thus the package is platform independent. ANTLR[17] was used to generate Java code for a lexer and parser based on our FCL grammar definition. This generated parser uses a left to right leftmost derivation recursive strategy, formally know as "LL(*)".

Using the lexer and parser created by ANTLR we are able to parse FCL files by creating an Abstract Syntax Tree (AST), a well known structure in compiler design. The AST is converted into an Interpreter Syntax Tree (IST), which is capable of performing the required computations. This means that the IST can represent the grammar, like and AST, but it also capable of performing calculations. The parsed FIS can be evaluated by recursively transversing the IST.

4.2. Membership functions

Only two membership functions are defined in the IEC standard: singleton and piece-wise linear. jFuzzyLogic also implements other commonly used membership functions:

- Cosine : $f(x|\alpha,\beta) = cos[\frac{\pi}{\alpha}(x-\beta)], \forall x \in [-\alpha,\alpha]$
- Difference of sigmoidals: $f(x|\alpha_1,\beta_1,\alpha_2,\beta_2) = s(x,\alpha_1,\beta_1) - s(x,\alpha_2,\beta_2)$, where $s(x,\alpha,\beta) = 1/[1+e^{-\beta(x-\alpha)}]$
- Gaussian : $f(x|\mu,\sigma) = e^{(x-\mu)^2/2\sigma^2}$

- Gaussian double : $f(x|\mu_1,\sigma_1,\mu_2,\sigma_2) =$

$$\begin{cases} e^{(x-\mu_1)^2/2\sigma_1^2} & x < \mu_1 \\ 1 & \mu_1 \leqslant x \leqslant \mu_2 \\ e^{(x-\mu_2)^2/2\sigma_2^2} & x > \mu_2 \end{cases}$$

- Generalized bell : $f(x|\mu_1,a,b) = \frac{1}{1+|(x-\mu)/a|^{2b}}$
- Sigmoidal : $f(x|\beta,t_0) = \frac{1}{1+e^{\beta(x-t_0)}}$
- Trapezoidal : $f(x|m_{in},l_{ow},h_{igh},m_{ax}) =$

$$\begin{cases} 0 & x < m_{in} \\ \frac{x-m_{in}}{l_{ow}-m_{in}} & m_{in} \leqslant x < l_{ow} \\ 1 & l_{ow} \leqslant x \leqslant h_{igh} \\ \frac{x-h_{igh}}{m_{ax}-h_{igh}} & h_{igh} < x \leqslant m_{ax} \\ 0 & x > m_{ax} \end{cases}$$

- Triangular: $f(x|m_{in},m_{id},m_{ax}) =$

$$\begin{cases} 0 & x < m_{in} \\ \frac{x-m_{id}}{l_{ow}-m_{id}} & m_{in} \leqslant x \leqslant m_{id} \\ \frac{x-m_{id}}{m_{ax}-m_{id}} & m_{id} < x \leqslant m_{ax} \\ 0 & x > m_{ax} \end{cases}$$

- Piece-wise linear : Defined as the union of all points by affine functions.

Furthermore, jFuzzyLogic allows to build arbitrary membership functions by combining mathematical expressions. This is implemented by parsing an Interpreter Syntax Tree (IST) of mathematical expressions. IST is evaluated at running time, thus allows including variables into the expressions. Current implementation allows the use of the following functions:Abs, Cos, Exp, Ln, Log, Modulus, Nop, Pow, Sin, Tan, as well as addition, subtraction, multiplication and division.

4.3. Aggregation, Activation & Accumulation

As mentioned in section 3.2, rules are defined inside the `RULEBLOCK` statement in a FIS. Each rule block also specifies Aggregation, Activation and Accumulation methods. All methods defined in the norm are implemented in jFuzzyLogic. It should be noted that we adhere to the definitions of Aggregation, Activation and Accumulation as defined by IEC-61131-7, which may differ from the naming conventions from other references.

Aggregation methods (sometimes be called "combination" or "rule connection methods") define the t-norms and t-conorms playing the role of AND & OR operators, which can be:

Name	x AND y	x OR y
Min/Max	$min(x,y)$	$max(x,y)$
Bdiff/Bsum	$max(0,x+y-1)$	$min(1,x+y)$
Prod/PobOr	$x\,y$	$x+y-x\,y$
Drastic	if$(x==1)\rightarrow y$ if$(y==1)\rightarrow x$ otherwise $\rightarrow 0$	if$(x==0)\rightarrow y$ if$(y==0)\rightarrow x$ otherwise $\rightarrow 1$
Nil potent	if$(x+y>1)\rightarrow min(x,y)$ otherwise $\rightarrow 0$	if$(x+y<1)\rightarrow max(x,y)$ otherwise $\rightarrow 1$

Needless to say, each set of operators must satisfy De Morgans laws.

Activation method define how rule antecedents modify rule consequents, i.e. once the IF part has been evaluated, how this result is applied to the THEN part of the rule. The most common activation operators are Minimum and Product (see Figure 2). Both methods are implemented in jFuzzyLogic.

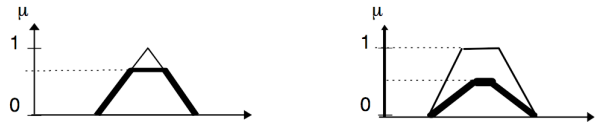

Fig. 2. Activation methods: Min (left) and Prod (right).

Finally, accumulation method defines how the consequents from multiple rules are combined within a Rule Block (see Figure 4). Accumulation methods implemented by jFuzzyLogic defined in the norm include:

- Maximum : $\alpha_{cc} = max(\alpha_{cc},\delta)$
- Bounded sum: $\alpha_{cc} = min(1,\alpha_{cc}+\delta)$
- Normalized sum: $\alpha_{cc} = \frac{\alpha_{cc}+\delta}{max(1,\alpha_{cc}+\delta)}$
- Probabilistic OR : $\alpha_{cc} = \alpha_{cc}+\delta-\alpha_{cc}\,\delta$

where α_{cc} is the accumulated value (at point x) and $\delta = m(x)$ is the membership function for de-

fuzzification (also at x).

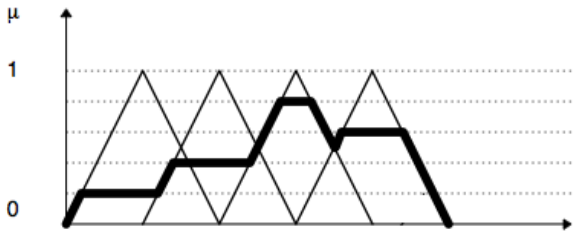

Fig. 3. Accumulation method: Combining consequents from multiple rules using Max accumulation method.

4.4. Defuzzification

In case of simple membership functions, such as trapezoidal and piece-wise linear, defuzzicication can be computed easily by applying know mathematical equations. Although it can be done very efficiently, unfortunately it cannot be generalized to arbitrary expressions.

Due to the flexibility in defining membership functions, we use a more general method. We discretize membership functions at a number of points and use a , more computational intensive, numerical integration method. The number of points used for discretization, by default one thousand, can be adjusted according to the precision-speed trade-off required for a particular application. Inference is performed by evaluating membership functions at these discretization points. In order to perform a discretization, the "universe" for each variable, has to be estimated. The universe is defined as the range where the variable has non-neglectable value. For each variable, each membership function and each term is taken into account when calculating a universe. Once all rules have been analyzed, the accumulation for each variable is complete.

The last step when evaluating a FIS is defuzzification. The value for each variable is calculated using the selected defuzzification method, which can be:

- Center of gravity : $\frac{\int x\mu(x)dx}{\int \mu(x)dx}$
- Center of gravity singleton : $\frac{\sum_i x_i \mu_i}{\sum_i \mu_i}$
- Center of area : $u \mid \int_{-\infty}^{u} \mu(x)dx = \int_{u}^{\infty} \mu(x)dx$
- Rightmost Max : $arg\max_x [\mu(x) = max(\mu(x))]$
- Leftmost Max : $arg\min_x [\mu(x) = max(\mu(x))]$
- Mean max : $mean(x) \mid \mu(x) = max(\mu(x))$

4.5. API extensions

Some of the extensions and benefits provided by jFuzzyLogic are described in this section.

Modularity. Modular design allows to extend the language and the API easily. It is possible to add custom aggregation, activation or accumulation methods, defuzzifiers, or membership functions by extending the provided object tree (see Figure 4).

Dynamic changes. Our API supports dynamic changes made onto a fuzzy inference system: i) variables can be used as membership function parameters; ii) rules can be added or deleted from rule blocks, iii) rule weights can be modified; iv) membership functions can use combinations of predefined functions.

Data Types. Due to the nature of fuzzy systems and in order to reduce complexity, jFuzzyLogic considers each variable as *REAL* variable which is mapped to a *double* Java type.

Excecution order. By default it is assumed that a FIS is composed of only one Function Block, so evaluating the FIS means evaluating the default FB. If a FIS has more than one FB, they are evaluated in alphabetical order by FB name. Other execution orders can be implemented by the user, which allows

us to easily define hierarchical controllers.

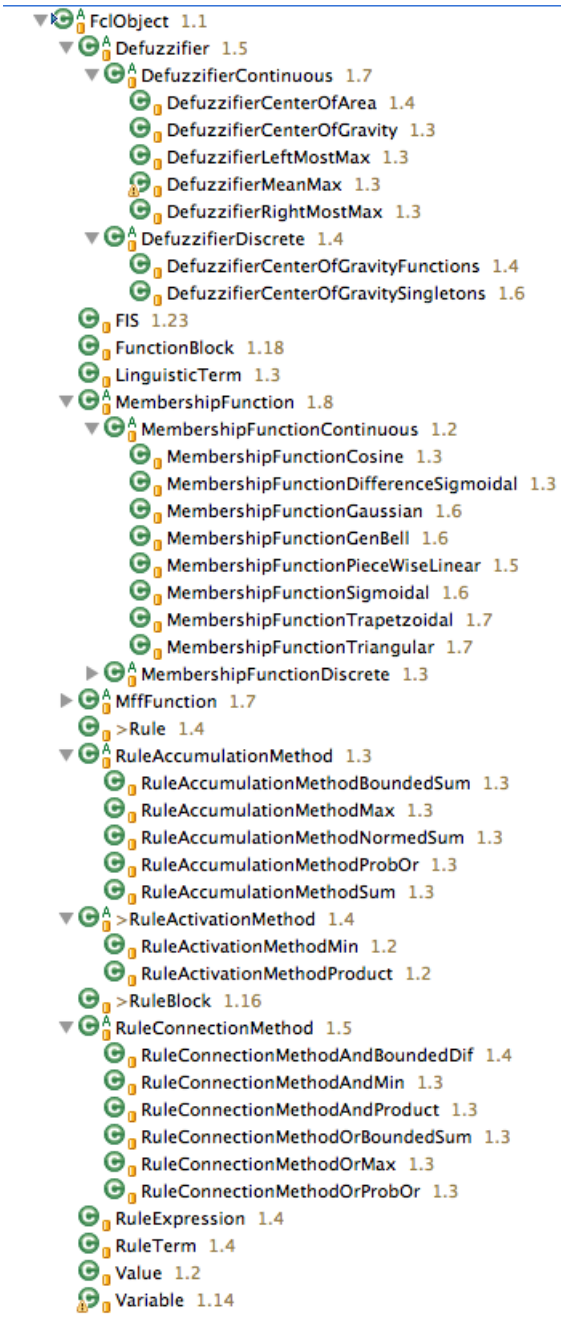

Fig. 4. jFuzzyLogic object tree provides many extension points.

4.6. Optimization API

An optimization API was developed in order to automatically fine tune FIS parameters. Our goal was to define a very lightweight and easy to learn API, which was flexible enough to be extended for general purpose usage.

The most common parameters to be optimized are membership functions and rule weights. For instance if a variable has a fuzzifier term "`TERM rancid := TRIAN 0 1 3`", there are three parameters that can be optimized in this membership function (whose initial values are 0, 1 and 3 respectively). Using the API, we can choose to optimize any of subset of them. Similarly, in the rule "`IF service IS good THEN tip IS average`" we can optimize the weight of this rule (implicit "WITH 1.0" statement).

The API, is composed of the following objects:

- *ErrorFunction* : An object that evaluates a Rule Block and calculates the error. Extending ErrorFunction is the bare minimum required to implement an optimization using one of the available optimization methods.
- *OptimizationMethod* : An optimization method object is an abstraction of an algorithm. It changes *Parameter* based on the performance measured using an ErrorFunction.
- *Parameter* : This class represents a parameter to be optimized. Any change on a parameter, will perform the corresponding change on the FIS, thus changing the outcome. There are two basic parameters: ParameterMembershipFunction and ParameterRuleWeight which allows for changes in membership functions and rule weights respectively. Other parameters could be created, for instance, in order to completely rewrite rules. We plan to extend them in future releases. Most users will not need to extend *Parameter* objects.

For most optimization applications, extending only one or two objects is enough (i.e. *ErrorFunction*, and sometimes *OptimizationMethod*). We provide template and demo objects to show how this can be done, all of them are included in our freely available source code.

A few optimization algorithms are implemented, such as gradient descent, partial derivative, and delta algorithm. As we mentioned, other algorithms can be easily implemented based on these templates or by directly extending them. In the provided examples it is assumed that error functions can be evaluated anywhere in the input space. Obviously, such evaluation is usually not available in real world applications, otherwise we would probably not need to implement a fuzzy system. This is not a limitation in the API, since we can always develop an optimization algorithm and the corresponding error function that evaluates the FIS on a learning set.

5. Eclipse plugin

Eclipse is one of the most commonly used software development platforms. It allows to specific language development tool by using the Eclipse-plugin framework. We developed a jFuzzyLogic plugin that allows developers to easily and rapidly write FCL code, and test it. Our plugin was developed using Xtext, a well known framework for domain specific languages based on ANTLR.

The plugin supports several features, such as syntax coloring, content assist, validation, program outlines and hyperlinks for variables and linguistic terms, etc. Figure 5 shows an example of the plugin being used to edit FCL code, the left panel shows an editor providing syntax coloring while adding content assist at cursor position, the right panel shows the corresponding code outline.

Running an FCL program (Figure 6) shows membership functions for all input and output variables while output console shows the FCL code parsed by jFuzzyLogic.

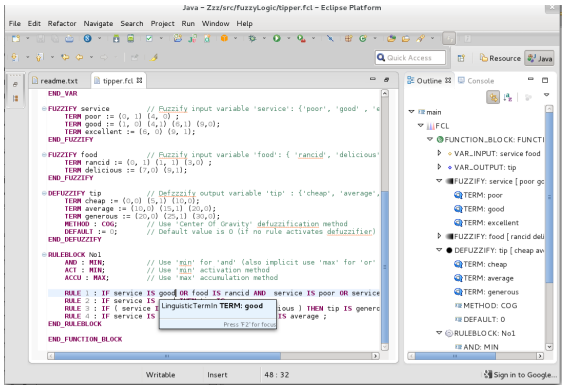

Fig. 5. jFuzzyLogic plugin for Eclipse. The editor (left window) provides syntax coloring and content assist. The right window shows code outline.

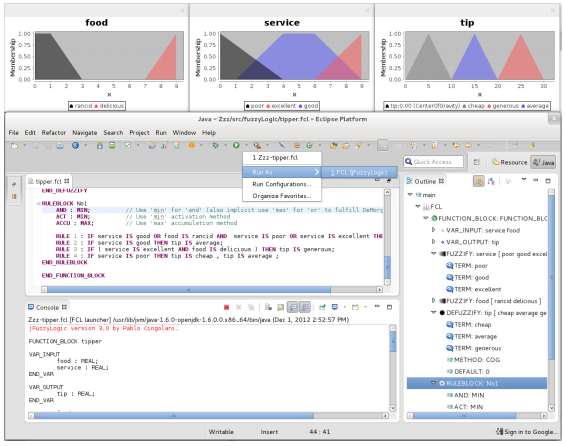

Fig. 6. jFuzzyLogic plugin for Eclipse. Running an FCL program (Figure 6) shows membership functions for all input and output variables.

6. A case study

We present an example of creating an FLC controller with jFuzzyLogic. This case study is focused on the development of the wall following robot as explained in [18]. Wall following behavior is well known in mobile robotics. It is frequently used for the exploration of unknown indoor environments and for the navigation between two points in a map.

The main requirement of a good wall-following controller is to maintain a suitable distance from

the wall that is being followed. The robot should also move as fast as possible, while avoiding sharp movements, making smooth and progressive turns and changes in velocity.

In our fuzzy control system, the input variables are: i) normalized distances from the robot to the right (*RD*) and left walls (*DQ*); ii) orientation with respect to the wall (*O*); and iii) linear velocity (*V*).

The output variables in this controller are the normalized linear acceleration (*LA*) and the angular velocity (*AV*). The linguistic partitions are shown in Figure 7 which are comprised by linguistic terms with uniformly distributed triangular membership functions giving meaning to them.

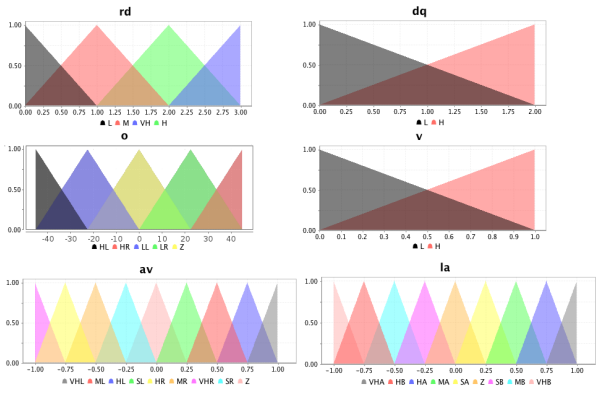

Fig. 7. Membership functions for wall-following robot.

In order to implement the controller, the first step is to declare the input and output variables and to define the fuzzy sets. Variables are defined in *VAR_INPUT* and *VAR_OUTPUT* sections. Fuzzy sets are defined in *FUZZIFY* blocks for input variables and *DEFUZZIFY* blocks for output variables.

One *FUZZIFY* block is used for each input variable. Each *TERM* line within a *FUZZIFY* block defines a linguistic term and its corresponding membership function. In this example all membership functions are triangular, so they are defined using the *'trian'* keyword, followed by three parameters defining left, center and right points (e.g. *'TRIAN 1 2 3'*).

Output variables define their membership functions within *DEFUZZIFY* blocks. Linguistic terms and membership functions are defined using the *TERM* keyword as previously described for input variables. In this case we also add parameters to select the defuzzyfication method. The statement *'METHOD : COG'* indicates that we are using 'Center of gravity'. The corresponding FCL code generated for the first step is as follows:

```
VAR_INPUT
  rd : REAL; // Right distance
  dq : REAL; // Distance quotient
  o  : REAL; // Orientation
  v  : REAL; // Velocity
END_VAR

VAR_OUTPUT
  la : REAL; // Linear acceleration
  av : REAL; // Angular velocity
END_VAR

FUZZIFY rd
  TERM L  := TRIAN 0 0 1;
  TERM M  := TRIAN 0 1 2;
  TERM H  := TRIAN 1 2 3;
  TERM VH := TRIAN 2 3 3;
END_FUZZIFY

FUZZIFY dq
  TERM L := TRIAN 0 0 2;
  TERM H := TRIAN 0 2 2;
END_FUZZIFY

FUZZIFY o
  TERM HL := TRIAN -45 -45 -22.5;
  TERM LL := TRIAN -45 -22.5 0;
  TERM Z  := TRIAN -22.5 0 22.5;
  TERM LR := TRIAN 0 22.5 45;
  TERM HR := TRIAN 22.5 45 45;
END_FUZZIFY

FUZZIFY v
  TERM L := TRIAN 0 0 1;
  TERM H := TRIAN 0 1 1;
END_FUZZIFY

DEFUZZIFY la
  TERM VHB := TRIAN -1 -1 -0.75;
  TERM HB  := TRIAN -1 -0.75 -0.5;
  TERM MB  := TRIAN -0.75 -0.5 -0.25;
  TERM SB  := TRIAN -0.5 -0.25 0;
  TERM Z   := TRIAN -0.25 0 0.25;
  TERM SA  := TRIAN 0 0.25 0.5;
  TERM MA  := TRIAN 0.25 0.5 0.75;
  TERM HA  := TRIAN 0.5 0.75 1;
  TERM VHA := TRIAN 0.75 1 1;
  METHOD : COG; // Center of Gravity
  DEFAULT := 0;
END_DEFUZZIFY

DEFUZZIFY av
  TERM VHR := TRIAN -1 -1 -0.75;
  TERM HR  := TRIAN -1 -0.75 -0.5;
```

```
  TERM MR  := TRIAN -0.75 -0.5 -0.25;
  TERM SR  := TRIAN -0.5 -0.25 0;
  TERM Z   := TRIAN -0.25 0 0.25;
  TERM SL  := TRIAN 0 0.25 0.5;
  TERM ML  := TRIAN 0.25 0.5 0.75;
  TERM HL  := TRIAN 0.5 0.75 1;
  TERM VHL := TRIAN 0.75 1 1;
  METHOD : COG;
  DEFAULT := 0;
END_DEFUZZIFY
```

These membership functions can be plotted by running jFuzzyLogic with the FCL file generated as argument (e.g. `java -jar jFuzzyLogic.jar robot.fcl`). The corresponding FCL file for this case study is available for download as one of the examples provided in jFuzzyLogic package (`jfuzzylogic.sourceforge.net`).

The second step is to define the rules used for inference. They are defined in *RULEBLOCK* statements. For the wall-following robot controller, we used 'minimum' connection method (*AND : MIN*), minimum activation method (*ACT : MIN*), and maximum accumulation method (*ACCU : MAX*). We implemented the rule base generated in [18] by the WCOR method [19]. Each entry in the rule base was converted to a single FCL rule. Within each rule, the antecedent (i.e. the *IF* part) is composed of the input variables connected by *'AND'* operators. Since there are more than one output variable, we can specify multiple consequents (i.e. *THEN* part) separated by semicolons. Finally, we add the desired weight using the *'with'* keyword followed by the weight. This completes the implementation of a controller for a wall-following robot using FCL and jFuzzyLogic. The Java code generated for the second step is as follows:

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

RULE 01: IF rd is  L and dq is L and o is LL
    and v is L THEN la is VHB , av is VHR with 0.4610;
RULE 02: IF rd is  L and dq is L and o is LL
    and v is H THEN la is VHB , av is VHR with 0.4896;
RULE 03: IF rd is  L and dq is L and o is  Z
    and v is L THEN la is   Z , av is  MR with 0.6664;
RULE 04: IF rd is  L and dq is L and o is  Z
    and v is H THEN la is  HB , av is  SR with 0.5435;
RULE 05: IF rd is  L and dq is H and o is LL
    and v is L THEN la is  MA , av is  HR with 0.7276;
RULE 06: IF rd is  L and dq is H and o is  Z
    and v is L THEN la is  MA , av is  HL with 0.4845;
RULE 07: IF rd is  L and dq is H and o is  Z
    and v is H THEN la is  HB , av is  ML with 0.5023;
RULE 08: IF rd is  L and dq is H and o is LR
    and v is H THEN la is VHB , av is VHL with 0.7363;
RULE 09: IF rd is  L and dq is H and o is HR
    and v is L THEN la is VHB , av is VHL with 0.9441;
RULE 10: IF rd is  M and dq is L and o is  Z
    and v is H THEN la is  SA , av is  HR with 0.3402;
RULE 11: IF rd is  M and dq is L and o is LR
    and v is H THEN la is   Z , av is VHL with 0.4244;
RULE 12: IF rd is  M and dq is L and o is HR
    and v is L THEN la is  SA , av is  HL with 0.5472;
RULE 13: IF rd is  M and dq is L and o is HR
    and v is H THEN la is  MB , av is VHL with 0.4369;
RULE 14: IF rd is  M and dq is H and o is HL
    and v is L THEN la is   Z , av is VHR with 0.1770;
RULE 15: IF rd is  M and dq is H and o is HL
    and v is H THEN la is VHB , av is VHR with 0.4526;
RULE 16: IF rd is  M and dq is H and o is LL
    and v is H THEN la is  SA , av is VHR with 0.2548;
RULE 17: IF rd is  M and dq is H and o is  Z
    and v is L THEN la is  HA , av is   Z with 0.2084;
RULE 18: IF rd is  M and dq is H and o is LR
    and v is L THEN la is  HA , av is VHL with 0.6242;
RULE 19: IF rd is  M and dq is H and o is LR
    and v is H THEN la is  SA , av is VHL with 0.3779;
RULE 20: IF rd is  M and dq is H and o is HR
    and v is L THEN la is   Z , av is VHL with 0.6931;
RULE 21: IF rd is  M and dq is H and o is HR
    and v is H THEN la is VHB , av is VHL with 0.7580;
RULE 22: IF rd is  H and dq is L and o is  Z
    and v is L THEN la is  HA , av is VHR with 0.5758;
RULE 23: IF rd is  H and dq is L and o is LR
    and v is H THEN la is  SA , av is  MR with 0.2513;
RULE 24: IF rd is  H and dq is L and o is HR
    and v is L THEN la is  HA , av is VHL with 0.5471;
RULE 25: IF rd is  H and dq is L and o is HR
    and v is H THEN la is  SA , av is  HL with 0.5595;
RULE 26: IF rd is  H and dq is H and o is HL
    and v is L THEN la is VHB , av is VHR with 0.9999;
RULE 27: IF rd is  H and dq is H and o is HL
    and v is H THEN la is VHB , av is VHR with 0.9563;
RULE 28: IF rd is  H and dq is H and o is LL
    and v is L THEN la is  HA , av is VHR with 0.9506;
RULE 29: IF rd is  H and dq is H and o is  Z
    and v is L THEN la is  HA , av is VHR with 0.4529;
RULE 30: IF rd is  H and dq is H and o is  Z
    and v is H THEN la is  SA , av is VHR with 0.2210;
RULE 31: IF rd is  H and dq is H and o is LR
    and v is L THEN la is  HA , av is  MR with 0.3612;
RULE 32: IF rd is  H and dq is H and o is LR
    and v is H THEN la is  SA , av is  MR with 0.2122;
RULE 33: IF rd is  H and dq is H and o is HR
    and v is L THEN la is  HA , av is  HL with 0.7878;
RULE 34: IF rd is  H and dq is H and o is HR
    and v is H THEN la is  SA , av is VHL with 0.3859;
RULE 35: IF rd is VH and dq is L and o is LR
    and v is L THEN la is  HA , av is VHR with 0.5530;
RULE 36: IF rd is VH and dq is L and o is HR
    and v is L THEN la is  HA , av is  HR with 0.4223;
```

```
RULE 37: IF rd is VH and dq is L and o is HR
    and v is H THEN la is  SA , av is  HR with 0.3854;
RULE 38: IF rd is VH and dq is H and o is LL
    and v is L THEN la is  HA , av is VHR with 0.0936;
RULE 39: IF rd is VH and dq is H and o is LR
    and v is L THEN la is  HA , av is VHR with 0.7325;
RULE 40: IF rd is VH and dq is H and o is LR
    and v is H THEN la is  SA , av is VHR with 0.5631;
RULE 41: IF rd is VH and dq is H and o is HR
    and v is L THEN la is  HA , av is  HR with 0.5146;
END_RULEBLOCK
```

The corresponding Java code that use jFuzzyLogic for running the FCL generated will be:

```
public class TestRobot {
  public static void main(String[] args)
  throws Exception {
    FIS fis = FIS.load("fcl/robot.fcl", true);
    FunctionBlock fb = fis.getFunctionBlock(null);
    // Set inputs
    fb.setVariable("rd", 0.3);
    fb.setVariable("dq", 1.25);
    fb.setVariable("o", 2.5);
    fb.setVariable("v", 0.6);
    // Evaluate
    fb.evaluate();
    // Get output
    double la = fb.getVariable("la").getValue());
    double av = fb.getVariable("av").getValue());
  }
}
```

This can also be done using the command line option "-e", which assigns values in the command line to input variables alphabetically (in this case: "dp", "o", "rd" and "v") and then evaluates the FIS. This utility produces plots of memnbership functions for all variables, as well as deffuzification areas for output all variables, in this example, "av" and "la" shown in light grey in Figure 8). Here we show the command, as well as part of the output:

```
$ java -jar jFuzzyLogic.jar -e robot.fcl \
      1.25 2.5 0.3 0.6

FUNCITON_BLOCK robot
  VAR_INPUT          dq = 1.250000
  VAR_INPUT           o = 2.500000
  VAR_INPUT          rd = 0.300000
  VAR_INPUT           v = 0.600000
  VAR_OUTPUT         av = 0.061952
  VAR_OUTPUT         la = -0.108399
   ...(rule activations omitted)
```

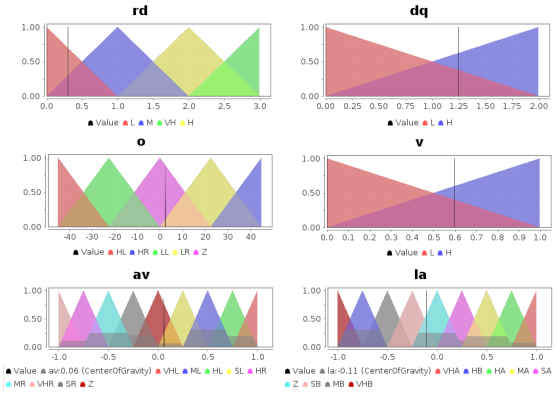

Fig. 8. Memnbership functions and deffuzification areas (light grey) for `robots.fcl` example.

7. Conclusions

In this paper, we have described jFuzzyLogic, an open source Java library for fuzzy systems which allow us to design FLCs following the standard IEC 61131. It allows us to reduce programming work and extend the range of possible users applying fuzzy systems and FLCs.

We have shown a case study to illustrate the use of jFuzzyLogic. In this case, we developed an FLC controller for wall-following behavior in a robot. The example shows how FCL can be used to easily implement fuzzy logic systems.

The jFuzzyLogic software package is continuously being updated and improved. At the moment, we are developing an implementation of a C++ compiler for fuzzy inference systems. This will allow easy implementation with embedded control systems using different processors.

Acknowledgments

jFuzzyLogic was designed and developed by P. Cingolani. He is supported in part by McGill Uninversity, Genome Quebec. J. Alcala-Fdez is supported by the Spanish Ministry of Education and Science under Grant TIN2011-28488 and the Andalusian Government under Grant P10-TIC-6858. We would

like to thank M. Blanchette and R. Sladek for their comments.

References

1. L.A. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.
2. C.C. Lee. Fuzzy logic in control systems: Fuzzy logic controller parts i and ii. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:404–435, 1990.
3. H. Hellendoorn D. Driankov and M. Reinfrank. *An Introduction to Fuzzy Control*. Springer-Verlag, 1993.
4. RR Yager and DP Filev. *Essentials of fuzzy modeling and control*. Wiley, New York, 1994.
5. P.P. Bonissone. Fuzzy logic controllers: An industrial reality. In *Computational Intelligence: Imitating Life*, pages 316–327. IEEE Press, 1994.
6. R Palm, D Driankov, and H Hellendoorn. *Model based fuzzy control*. Springer, Berlin, 1997.
7. M. Mucientes, J. Alcalá-Fdez, R. Alcalá, and J. Casillas. A case study for learning behaviors in mobile robotics by evolutionary fuzzy systems. *Expert Systems With Applications*, 37(2):1471–1493, 2010.
8. Ch.-F. Juang and Y.-Ch. Chang. Evolutionary-group-based particle-swarm-optimized fuzzy controller with application to mobile-robot navigation in unknown environments. *IEEE Transactions on Fuzzy Systems*, 19(2):379–392, 2011.
9. M.J. Gacto, R. Alcalá, and F. Herrera. A multi-objective evolutionary algorithm for an effective tuning of fuzzy logic controllers in heating, ventilating and air conditioning systems. *Applied Intelligence*, 36(2):330–347, 2012.
10. E. Cho, M. Ha, S. Chang, and Y. Hwang. Variable fuzzy control for heat pump operation. *Journal of Mechanical Science and Technology*, 25(1):201–208, 2011.
11. F. Chávez, F. Fernández, R. Alcalá, J. Alcalá-Fdez, G. Olague, and F. Herrera. Hybrid laser pointer detection algorithm based on template matching and fuzzy rule-based systems for domotic control in real home enviroments. *Applied Intelligence*, 36(2):407–423, 2012.
12. G. Acampora and V. Loia. Fuzzy control interoperability and scalability for adaptive domotic framework. *IEEE Transactions on Industrial Informatics*, 1(2):97 – 111, 2005.
13. Y. Zhao and H. Gao. Fuzzy-model-based control of an overhead crane with input delay and actuator saturation. *IEEE Transactions on Fuzzy Systems*, 20(1):181 –186, 2012.
14. O. Demir, I. Keskin, and S. Cetin. Modeling and control of a nonlinear half-vehicle suspension system: A hybrid fuzzy logic approach. *Nonlinear Dynamics*, 67(3):2139–2151, 2012.
15. *International Electrotechnical Commission technical committee industrial process measurement and control. IEC 61131 - Programmable Controllers*. IEC, 2000.
16. S. Sonnenburg, M.L. Braun, Ch.S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. Muller, F. Pereira, C.E. Rasmussen, G. Ratsch, B. Scholkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.
17. T. Parr. *The definitive ANTLR reference: building domain-specific languages*. 2007.
18. M. Mucientes, R. Alcalá, J. Alcalá-Fdez, and J. Casillas. Learning weighted linguistic rules to control an autonomous robot. *International Journal of Intelligent Systems*, 24(3):226–251, 2009.
19. R. Alcalá, J. Alcalá-Fdez, J. Casillas, O. Cordón, and F. Herrera. Hybrid learning models to get the interpretability-accuracy trade-off in fuzzy modelling. *Soft Computing*, 10(9):717–734, 2006.