An specification language for fuzzy systems

F.J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone, D.R. López
Instituto de Microelectrónica de Sevilla (IMSE-CNM)
Avda. Reina Mercedes, s/n Edif. CICA. E-41012 Sevilla SPAIN
e-mail: velo@imse.cnm.es

Abstract

This work presents the main features of XFL3, a language for fuzzy system specification, which has been defined as the common description languaje for the tools forming the Xfuzzy 3.0 development environment. Its main advantages are its capability to admit user-defined membership functions, parametric operators, and linguistic hedges. A brief summary of the tools included in Xfuzzy 3.0 and an example illustrating the use of XFL3 are also included.

1 Introduction

The definition of formal languages for fuzzy system specification is usual for its several advantages [1][2][3]. However, two objectives may conflict. A generally high expressive language, able to apply all the fuzzy logic-based formalisms, is desired, but, at the same time, the final system implementation constraints have to be considered. In this sense, some languages focus on expressiveness [4][5], while others are focused on software or hardware implementations [6].

One of our main objectives when we begun to design a fuzzy system environment was to develop an open environment that was not constrained by the implementation details, but offered the user a wide set of tools allowing different implementations from a general system description. This led us to the definition of the formal language XFL [7]. The main features of XFL were the separation of the system structure definition from the definition of the functions assigned to the fuzzy operators, and the capabilities for defining complex systems. XFL is the base for several hardware- and software-oriented development tools that constitute the Xfuzzy 2.0 design environment [8].

As a starting point for the 3.0 version of Xfuzzy, a new language, XFL3, which extends the advantages of XFL, has been defined. XFL3 allows the user to define new membership functions and parametric operators, and admits the use of linguistic hedges that permit to describe more complex relationships among variables [9][10]. In order to incorporate these improvements, some modifications have been made in the XFL syntax. In addition, the new language XFL3, together with the

tools based on it, employ Java as programming language. This means the use of an advantageous object-oriented methodology and the flexibility of executing the new version of Xfuzzy in any platform with JRE (Java Runtime Environment) installed.

2 The XFL3 language

XFL3 is a fuzzy system specification language that provides the user with a great flexibility to define the functions associated with the fuzzy operators and linguistic variables and that allows to express complex rule bases.

An XFL3 specification consists of several objects defining operator sets, variable types, and rule bases. The definition format of these elements is described in the following.

2.1 Operator sets

An operator set in XFL3 is an object containing the mathematical functions that are assigned to each fuzzy operator. Fuzzy operators can be binary (like the T-norms and S-norms employed to represent linguistic variable connections, implication, or rule aggregations), unary (like the C-norms or the operators related with linguistic hedges), or can be associated with defuzzification methods [11].

XFL3 defines the operator sets with the following format (Figure 1):

operatorset identifier {
  operator assigned_function(parameter_list);
  operator assigned_function(parameter_list);
  .... }

It is not required to specify all the operators. When one of them is not defined, its default function is assumed. Table 1 shows the operators (and their default functions) currently used in XFL3.

Table 1: Operators currently defined in XFL3

Operator	Type	Default function
and	binary	min(a,b)
or	binary	max(a,b)
implication, imp	binary	min(a,b)
also	binary	max(a,b)
not	unary	$(1-a)$
very, strongly	unary	$(a)^2$
moreorless	unary	$(a)^{1/2}$
slightly	unary	$4*a*(1-a)$
defuzzification, defuz	defuzzification	center of area

The assigned functions are defined in external files which we name as packages. The format to identify a function is "*package.function*". The package name (*xfl* in

Figure 1) can be removed if the package has been imported previously (using the command "*import package;*").

```
operatorset systemop {
   and xfl.min();
   or xfl.max();
   imp xfl.min();
   strongly xfl.pow(3);
   moreorless xfl.pow(0.4);
  }
```

Figure 1: Example of an operator set definition

2.2 Types of linguistic variables

An XFL3 type is an object that describes a type of linguistic variable. This means to define its universe of discourse, to name the linguistic labels covering that universe, and to specify the membership function associated to each label. The definition format of a type is as follows (Figure 2):

type identifier [min, max; card] {
  label membership_function(parameter_list);
  label membership_function(parameter_list);
  .... }

where min and max are the limits of the universe of discourse and card (cardinality) is the number of its discrete elements. If cardinality is not specified, its default value (currently, 256) is assumed. When limits are not explicitly defined, the universe of discourse is taken from 0 to 1.

```
type input1 [0,100] {
   short xfl.triangle(0,25,50);
   medium xfl.triangle(25,50,75);
   tall xfl.triangle(50,75,100);
  }

type input2 extends input1 {
   very_short xfl.triangle(-10,0,25);
   very_tall xfl.triangle(75,100,110);
  }
```

Figure 2: Example of a variable type definition

The format of the linguistic label identifier is similar to the operator identifier, that is, "*package.function*" or simply "*function*" if the package where the user has defined the membership functions has been already imported.

XFL3 supports inheritance mechanisms in the type definitions (like its precursor, XFL). To express inheritance, the heading of the definition is as follows (Figure 2):

type identifier **extends** identifier {

The types so defined inherit automatically the universe of discourse and the labels of their parents. The labels defined in the body of the type are either added to the parent labels or overwrite them if they have the same name.

2.3 Rule bases

A rule base in XFL3 is an object containing the rules that define the logic relationships among the linguistic variables. Its definition format is as follows (Figure 3):

rulebase identifier (input_list : output_list) **using** operatorset {
[factor] **if** (antecedent) -> consequent_list;
[factor] **if** (antecedent) -> consequent_list;
.... }

The definition format of the input and output variables is "*type identifier*", where type refers to one of the linguistic variable types previously defined. The operator set selection (*systemop* in Figure 3) is optional, so that when it is not explicitly defined, the default operators are employed. It is also shown in Figure 3 how confidence weights (with default values of 1) can be applied to the rules.

```
rulebase base1(input1 x, input2 y :  output z) using systemop {
   if( x == medium & y == medium) -> z = tall;
   [0.8] if( x<=short | y != very_tall ) -> z = short;
   if( +(x>tall) & (y ~= medium) ) -> z = tall;
   ............. }
```

Figure 3: Example of a rule base definition

A rule antecedent describes the relationships among the input variables. XFL3 allows to express complex antecedents by combining basic propositions with connectives or linguistic hedges (Table 2 and Figure 4). On the other side, each rule consequent describes the assignation of a linguistic variable to an output variable as "*variable = label*" (Figure 3).

2.4 System global behavior

The description of the system global behavior means to define the global input and output variables of the system as well as the rule base hierarchy. This description in XFL3 is as follows (Figure 5):

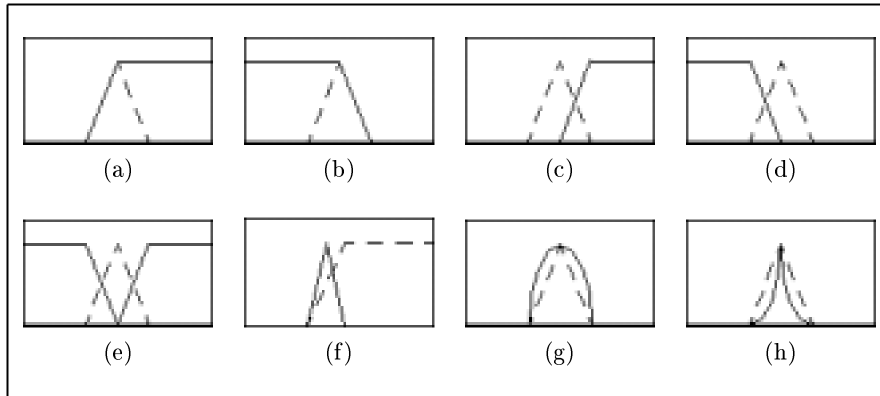


Figure 4: Illustrating linguistic hedges

Table 2: Kinds of fuzzy propositions

Basic propositions	**Description**
variable == label	equal to
variable >= label	equal or greater than (Fig. 4a)
variable <= label	equal or smaller than (Fig. 4b)
variable > label	greater than (Fig. 4c)
variable < label	smaller than (Fig. 4d)
variable != label	not equal to (Fig. 4e)
variable % = label	slightly equal to (Fig. 4f)
variable ˜ = label	moreorless equal to (Fig. 4g)
variable += label	strongly equal to (Fig. 4h)
Complex propositions	**Description**
proposition & proposition	and operator
proposition — proposition	or operator
!proposition	not operator
% proposition	slightly operator
˜ proposition	moreorless operator
+proposition	strongly operator

system (input_list : output_list) {
rule_base_identifier(inputs : outputs);
rule_base_identifier(inputs : outputs);
.... }

```
system(input1 x, input2 y :  output z) {
   rulebase1(x, y :  inner1);
   rulebase2(x, y :  inner2);
   rulebase3(inner1, inner2 :  z);
  }
```

Figure 5: Example of a system behavior definition

The definition format of the global input and output variables is the same format employed in the definition of the rule bases. The inner variables that may appear establish serial or parallel interconnections among the rule bases. Inner variables must firstly appear as output variables of a rule base before being employed as input variables of other rule bases (Figure 5).

3 Function packages

A great advantage of XFL3 is that functions asigned to fuzzy operators can be defined freely by the user in external files (named as packages), which gives a huge flexibility to the environment. A function definition include its name (and possible *alias*), the parameters that specify its behavior as well as the constraints on these parameters, the description of its behavior in the different languages to which it could be compiled (*java*, *ansi_c* and *cplusplus*, for instance), and even the description of its differential function (if it is employed in gradient-based learning mechanisms). This information is the basis to generate automatically a Java class that incorporates all the function capabilities and can be employed by any XFL3 specification.

Four types of functions can be defined in XFL3: binary functions (like T-norms, S-norms, and implication functions), unary functions (like C-norms and functions related with linguistic hedges), membership functions, and functions associated with defuzzification methods. The definition format is as follows:

binary identifier { blocks }
unary identifier { blocks }
mf identifier { blocks }
defuz identifier { blocks }

Common blocks are *alias*, *parameter*, *requires*, *java*, *ansi_c*, *cplusplus*, *derivative* and *source*. The block *alias* is used to define alternative names to the function identification. Its format is:

alias identifier, identifier, ... ;

The block *parameter* allows the definition of the parameters on which the function depends. The format is:

parameter identifier, identifier, ... ;

The block *requires* expresses the constraints in the parameter values by means of a java boolean expression. Its structure is:

requires { java_expression }

The blocks *java*, *ansi_c* and *cplusplus* define the behavior of the operator by means of their descriptions in these programming languages. The input variable is called '*a*' in unary functions, while in binary functions, the names of the input variables are '*a*' and '*b*'. Membership functions use the input variable '*x*'. Defuzzification methods use the object '*mf*' that includes the description of the aggregated membership function to be defuzzified. The format of these blocks is:

java { function_code }
ansi_c { function_code }
cplusplus { function_code }

The block *derivative* describes the differential function that is used by some gradient-descent learning algorithms whose objective is to fit a desired system behavior by changing the value of the membership functions parameters. The block contains a Java expression assigning a value to the variable '*deriv*'. When using unary functions, '*deriv*' describes the derivative with respect to the variable '*a*'. In binary functions, '*deriv*[0]' and '*deriv*[1]' contains the derivative with respect to '*a*' and '*b*', respectively. When considering membership functions, '*deriv*[i]' must be assigned to the derivative with respect to the i-th parameter. Currently, derivatives of defuzzification methods cannot be introduced. The structure of the block is:

derivative { function_code }

The block *source* is used to describe Java code to be directly included in the body of the Java class generated for the function definition. This block allows the definition of local functions that can be used in other blocks. Its format is as follows:

source { java_code }

The description of a membership function is a bit more complex than those of binary or unary functions, since it may include not only the description of the function behavior, but also another information like its center, its basis and its behavior under the linguistic hedges 'greater or equal than' and 'smaller or equal than'. In order to include this information, the blocks *java*, *ansi_c*, *cplusplus* and

derivative are divided into the following subblocks: *equal*, *greatereq*, *smallereq*, *center* and *basis*. Only the first one is mandatory. By default, the rest are computed by sweeping the universe of discourse.

Some defuzzification methods are limited to the use of certain kinds of membership functions (like triangles or trapezoids). The block *definedfor* has been introduced to express this constraints. Its format is:

definedfor identifier, identifier, ... ;

where identifiers refer to the name of the membership functions accepted by the method.

Figure 6 shows some examples of function definitions. The use of packages allows the designer to define any desired function. The standard package currently used in XFL3 (and named *xfl*) contains the most usual functions, as shown in Table 3.

Table 3: The standard package *xfl*

Function type	**Possible assigned functions**
Binary	min, prod, bounded_prod, drastic_prod, max, sum, bounded_sum, drastic_sum, dienes_resher, mizumoto, lukasiewicz, dubois_prade, zadeh, goguen, godel, sharp.
Unary	not, sugeno, yager, pow, parabola.
Membership functions	trapezoid, triangle, isosceles, slope, bell, sigma, rectangle, singleton.
Defuzzification methods	CenterOfArea, FirstOfMaxima, LastOfMaxima, MeanOfMaxima, FuzzyMean, WeightedFuzzyMean, Quality, GammaQuality, MaxLabel.

4 Example of an XFL3 specification

One of the main features of XFL3 is the inclusion of linguistic hedges and hierarchical structures on the definition of fuzzy systems. The modular division of the system description allows the designer to confront the development of complex systems. In this sense, linguistic hedges can be used to decrease the number of linguistic labels employed and to express logic rules more compactly [12].

As an example of complex system modelling, we have considered the problem of parking a car between two other cars (Figure 7). The approximation we have followed is to directly emulate how we will act as drivers. For us this is a three step problem: the first one is to approximate the car to the parking place, the second step is to drive backwards to introduce the car into the place, and the last one is to straighten and center the car. Since the proper turn of the wheels depends on the step the car is performing, our expert knowledge is represented by a hierarchical system. In particular, six rule bases are employed, as shown at the bottom of

```
binary min {
 java { return (a<b? a : b); }
 ansi_c { return (a<b? a : b); }
 cplusplus { return (a<b? a : b); }
 derivative {
  deriv[0] = (a<b? 1: (a==b? 0.5 : 0));
  deriv[1] = (a>b? 1: (a==b? 0.5 : 0));
 }
}

mf bell {
 parameter a, b;
 requires { a>=min && a<=max && b>0 }
 java {
  equal { return Math.exp( -(a-x)*(a-x)/(b*b) ); }
  greatereq { if(x>a) return 1; return Math.exp( -(a-x)*(a-x)/(b*b) ); }
  smallereq { if(x<a) return 1; return Math.exp( -(a-x)*(a-x)/(b*b) ); }
  center { return a; }
  basis { return b; }
 }
}

defuz CenterOfArea {
 alias CenterOfGravity, Centroid;
 java {
  double num=0, denom=0;
  for(double x=min; x<=max; x+=step) {
   num += x*mf.compute(x);
   denom += mf.compute(x);
  }
  if(denom==0) return (min+max)/2;
  return num/denom;
 }
}
```

Figure 6: Examples of function definitions

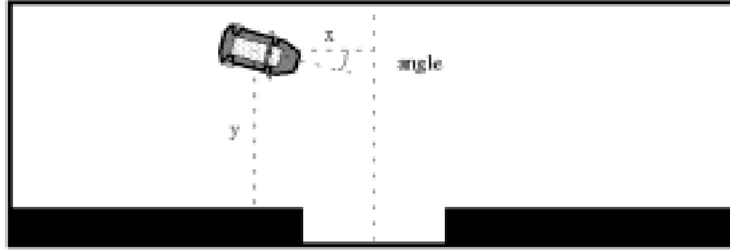
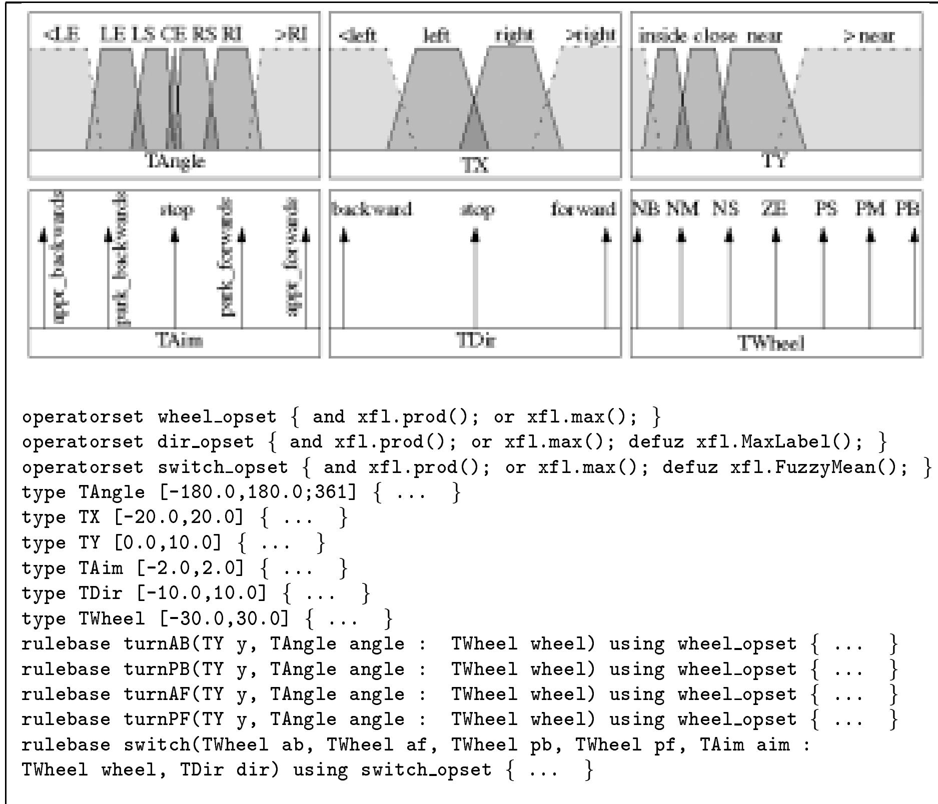
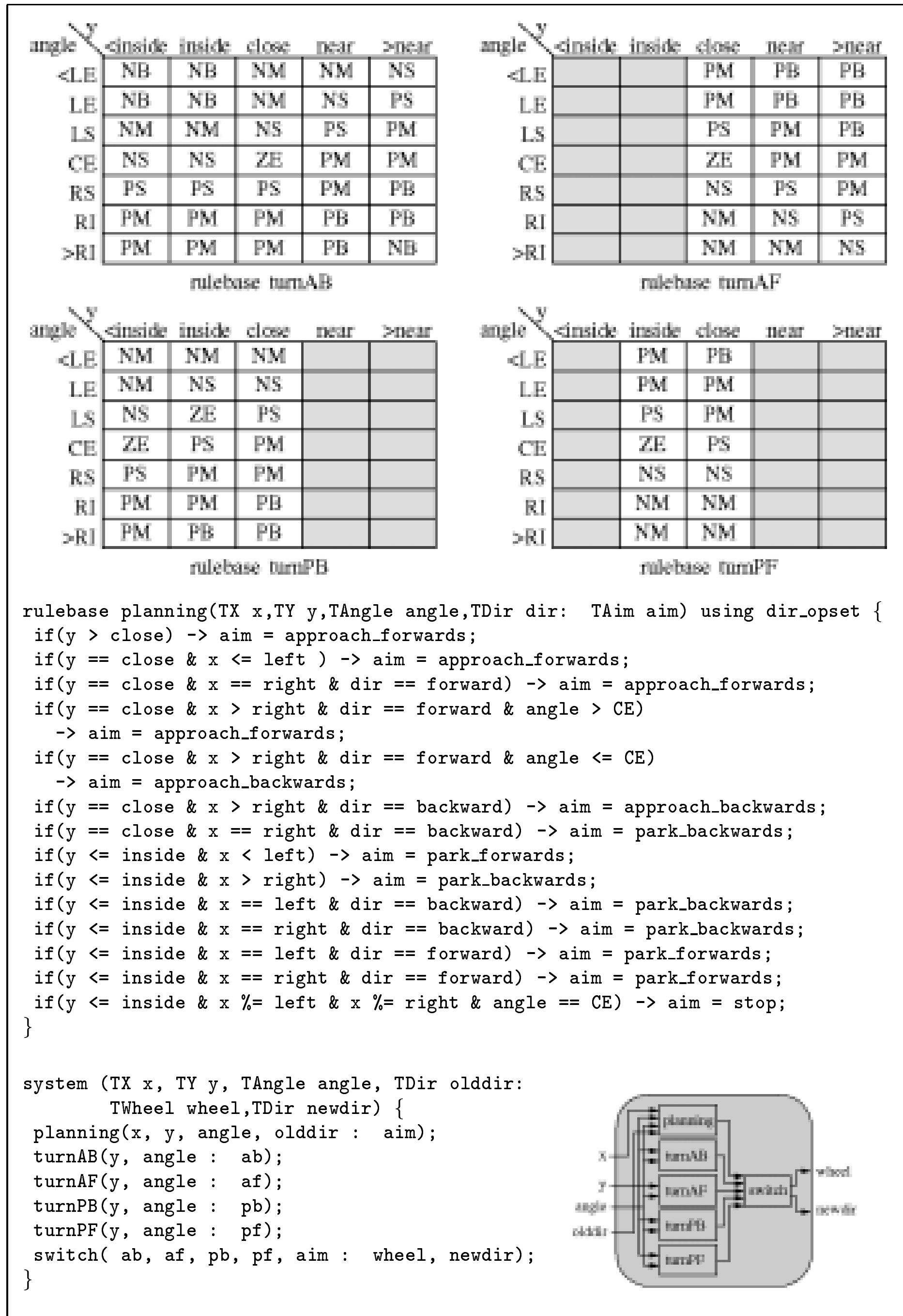


Figure 7: Diagram of the parking problem

Figure 8. The rule base *planning* decides on the kind of movement that the car will carry out: approching forwards or backwards, and parking backwards or forwards. The two objectives of the rule base *planning* is to straighten the car ($angle = 0$) and to finish in the center of the parking place ($x = 0$) ... On the other hand, the rule bases *turnAF*, *turnAB*, *turnPB* and *turnPF* decides on the wheel angle for each kind of movement. Finally, the rule base *switch* chooses the proper turn as a function of the car movement.




```
operatorset wheel_opset { and xfl.prod(); or xfl.max(); }
operatorset dir_opset { and xfl.prod(); or xfl.max(); defuz xfl.MaxLabel(); }
operatorset switch_opset { and xfl.prod(); or xfl.max(); defuz xfl.FuzzyMean(); }
type TAngle [-180.0,180.0;361] { ... }
type TX [-20.0,20.0] { ... }
type TY [0.0,10.0] { ... }
type TAim [-2.0,2.0] { ... }
type TDir [-10.0,10.0] { ... }
type TWheel [-30.0,30.0] { ... }
rulebase turnAB(TY y, TAngle angle : TWheel wheel) using wheel_opset { ... }
rulebase turnPB(TY y, TAngle angle : TWheel wheel) using wheel_opset { ... }
rulebase turnAF(TY y, TAngle angle : TWheel wheel) using wheel_opset { ... }
rulebase turnPF(TY y, TAngle angle : TWheel wheel) using wheel_opset { ... }
rulebase switch(TWheel ab, TWheel af, TWheel pb, TWheel pf, TAim aim :
TWheel wheel, TDir dir) using switch_opset { ... }
```



angle \ y	<inside	inside	close	near	>near
<LE	NB	NB	NM	NM	NS
LE	NB	NB	NM	NS	PS
LS	NM	NM	NS	PS	PM
CE	NS	NS	ZE	PM	PM
RS	PS	PS	PS	PM	PB
RI	PM	PM	PM	PB	PB
>RI	PM	PM	PM	PB	NB

rulebase turnAB

angle \ y	<inside	inside	close	near	>near
<LE			PM	PB	PB
LE			PM	PB	PB
LS			PS	PM	PB
CE			ZE	PM	PM
RS			NS	PS	PM
RI			NM	NS	PS
>RI			NM	NM	NS

rulebase turnAF

angle \ y	<inside	inside	close	near	>near
<LE	NM	NM	NM		
LE	NM	NS	NS		
LS	NS	ZE	PS		
CE	ZE	PS	PM		
RS	PS	PM	PM		
RI	PM	PM	PB		
>RI	PM	PB	PB		

rulebase turnPB

angle \ y	<inside	inside	close	near	>near
<LE		PM	PB		
LE		PM	PM		
LS		PS	PM		
CE		ZE	PS		
RS		NS	NS		
RI		NM	NM		
>RI		NM	NM		

rulebase turnPF

```
rulebase planning(TX x,TY y,TAngle angle,TDir dir:  TAim aim) using dir_opset {
 if(y > close) -> aim = approach_forwards;
 if(y == close & x <= left ) -> aim = approach_forwards;
 if(y == close & x == right & dir == forward) -> aim = approach_forwards;
 if(y == close & x > right & dir == forward & angle > CE)
   -> aim = approach_forwards;
 if(y == close & x > right & dir == forward & angle <= CE)
   -> aim = approach_backwards;
 if(y == close & x > right & dir == backward) -> aim = approach_backwards;
 if(y == close & x == right & dir == backward) -> aim = park_backwards;
 if(y <= inside & x < left) -> aim = park_forwards;
 if(y <= inside & x > right) -> aim = park_backwards;
 if(y <= inside & x == left & dir == backward) -> aim = park_backwards;
 if(y <= inside & x == right & dir == backward) -> aim = park_backwards;
 if(y <= inside & x == left & dir == forward) -> aim = park_forwards;
 if(y <= inside & x == right & dir == forward) -> aim = park_forwards;
 if(y <= inside & x %= left & x %= right & angle == CE) -> aim = stop;
}

system (TX x, TY y, TAngle angle, TDir olddir:
        TWheel wheel,TDir newdir) {
 planning(x, y, angle, olddir :  aim);
 turnAB(y, angle :  ab);
 turnAF(y, angle :  af);
 turnPB(y, angle :  pb);
 turnPF(y, angle :  pf);
 switch( ab, af, pb, pf, aim :  wheel, newdir);
}
```


Figure 8: Summary of the XFL3 specification of a fuzzy control system for the parking problem

This example does not attempt to illustrate an optimum way of solving a parking problem but the efficiency of XFL3 for representing expert linguistic knowledge. In this sense, the definitions of the variable types have been reduced by using the greater and smaller linguistic hedges, and the rule base definitions have been compacted thanks to the combination of connectives and hedges, as we use linguistically.

```
angle(t) = angle(t-1)+newdir*wheel/25
x(t) = x(t-1)+newdir*cos(angle*π/180)/50
y(t) = y(t-1)+newdir*sin(angle*π/180)/50
olddir = newdir
```

Figure 9: Model of the car behavior in the parking problem

Figure 9 shows the equations that model the car behavior. These equations have been used in a feedback process to generate a simulated track for the car. The results of two simulations are shown in Figure 10. In the first one, the car starts several meters before the parking place, so the system decides to drive forwards approching the car to the place, then to move backwards introducing the car in the place and, finally, to drive forwards straightening the car. The second simulation begins with the car in a more advanced position, so that approching forwards leads the car several meters away form the parking place. The system, therefore, drives the car backwards to approach the parking place, then turns the car to introduce it into the place and, finally, straightens the car. As the car does not reach a centered position in the straighten forwards step, the planning rule base decides to carry out a backwardsstep to finish the parking proccess.

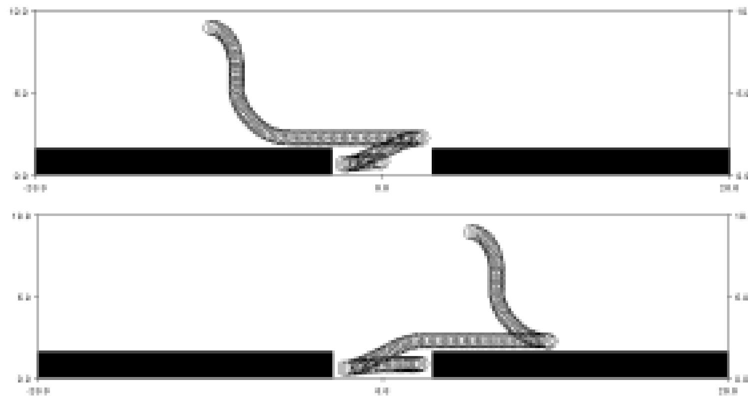


Figure 10: Simulation results on the parking problem

5 Summary of the XFL3-based tools

The core of any application developed with XFL3 is based on the use of Java classes that contain the whole structure and functionality of the specifications to be worked with. Using these classes and the Java graphic libraries, several tools have been built that allow exploiting the XFL3 features through the different development stages of a fuzzy system design [13].

Currently, a fuzzy system edition tool, named *xfedit*, is available. It allows defining the operator sets, variable types, rule bases, and system structure through a graphical user interface, as shown in Figure 11.

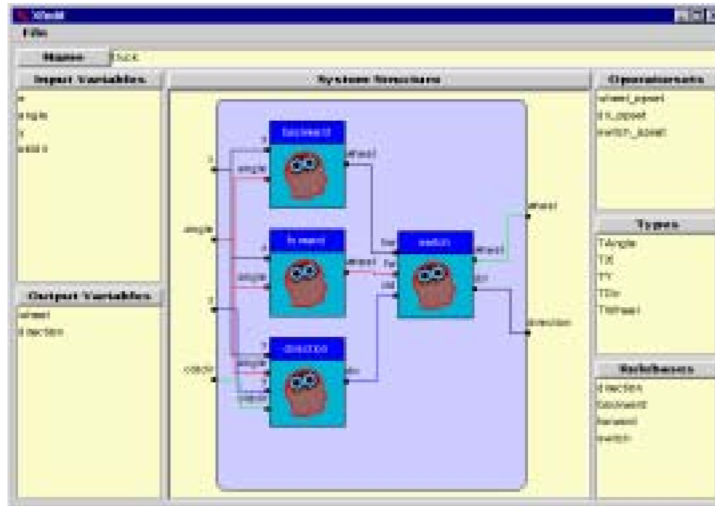


Figure 11: Main window of the system edition tool

Two graphic representation tools, named *xf2dplot* and *xf3dplot*, have been also developed (Figure 12). They allow illustrating the fuzzy system behavior by its output/input surface in a two or three dimensional space.

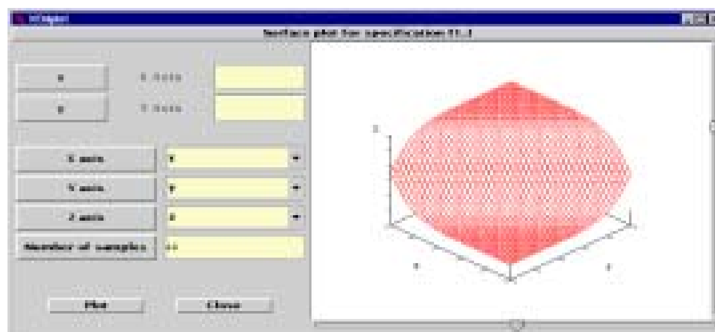


Figure 12: Main window of the graphical representation tool

A tool for automatically adjusting fuzzy systems described with XFL3 is also available. This tool, named *xfsl*, provides the user with several learning algorithms and allows selecting which system parameters are going to be tuned and which not (Figure 13). Besides, this tool includes two methods of pre- and post-processing for eliminating non significant rules and labels and for clustering the output variables, thus simplifying their associated types.

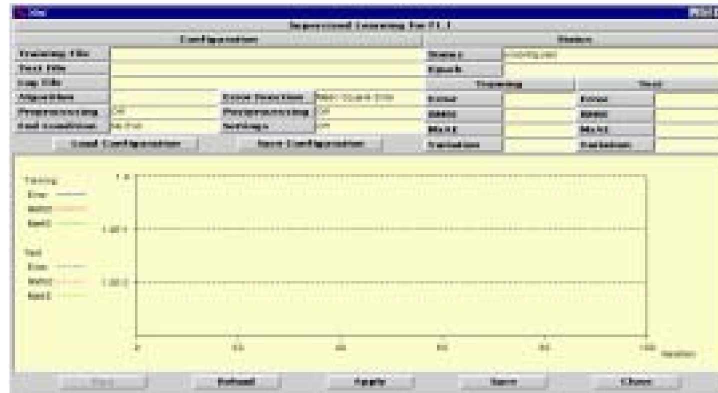


Figure 13: Main window of the supervised learning tool

The final step of any design process is the synthesis step which can lead to a software or hardware system implementation. To ease the software synthesis, three tools have been developed, *xfj*, *xfc* and *xfcpp*, which generate, respectively, the system description in Java, C, and C++ languages.

These and other tools under development attempt to cover all the different stages of a fuzzy system design from its linguistic description to its final implementation (either software or hardware) and will constitute the 3.0 version of the Xfuzzy environment.

6 Conclusions

This paper has introduced the XFL3 language, which has been developed after accumulating experience with the design of the Xfuzzy 2.0 environment. XFL3 eases the description and manipulation of complex fuzzy systems thanks to the use of quite user-defined membership functions, fuzzy operators (including linguistic hedges), and rule bases (admitting hierarchical structures). An illustrative example has been included to show the efficiency of XFL3 to rapidly translate linguistic knowledge. Based on this language, several tools are being developed to constitute the new version of Xfuzzy, which could be executed on any platform containing the Java Runtime Environment.

References

[1] M. Bonner, S. Mayer, A. Raggl, W. Slany, "FLIP++: A fuzzy logic inference library", Fuzzy Logic in Artificial Intelligence: Towards Intelligent Systems, Martin, T.P., Ralescu, A.L., Eds., Springer-Verlag, 1997.

[2] Z. A. Sosnowski, "FLISP - a language for processing fuzzy data", Fuzzy Sets and Systems, No 37, pp. 23-32, 1990.

[3] International Electrotechnical Commission, "IEC 1131 - Programmable Controllers. Part 7 - Fuzzy Control Programming", Committee Draft CD 1.0, 1997.

[4] O.G.Duarte, G. Pérez. "UNFUZZY: Fuzzy Logic System analysis, design, simulation and implementation software", Proc. 1999 Eusflat-Estylf Joint Conf., pp. 251-254 Mallorca, 1999.

[5] R. Hartwig, C. Labinsky, S. Nordhoff, B. Landorff, P. Jensch, J. Schwanke. "Free Fuzzy Logic System Design Tool: FOOL", Proc. 4th European Congress on Intelligent Techniques and Soft Computing (EUFIT'96), pp. 2274-2277, Aachen, Sep. 1996.

[6] J. Yen, R. Langari, L.A. Zadeh, Eds., Industrial Applications of Fuzzy Logic and Intelligent Systems, IEEE Press, 1995.

[7] D. López, F. J. Moreno, A. Barriga, S. Sánchez-Solano. "XFL: A Language for the Definition of Fuzzy Systems", Proc. 6th IEEE Int. Conf. on Fuzzy Systems (FUZZIEEE'97), pp. 1585-1591, Barcelona, Jul., 1997.

[8] D. R. López, C. J. Jiménez, I. Baturone, A. Barriga, S. Sánchez-Solano. "Xfuzzy: A Design Environment for Fuzzy Systems", Proc. 7th IEEE Int. Conf. on Fuzzy Systems (FUZZIEEE'98), pp. 1060-1065, Anchorage, May 1998.

[9] L.A. Zadeh, "A fuzzy-set-theoretic interpretation of linguistic hedges", J. Cybern, vol. 2, pp. 4-34, 1972.

[10] J. Gutierrez, L. de Salvador, "The use of modifiers in fuzzy systems and their hardware implementation", Proc. 6th IEEE International Conference on Fuzzy Systems, pp. 881-887, Barcelona, Jul. 1997.

[11] E.H. Ruspini, P.P. Bonissone, W. Pedrycz, Eds., Handbook of Fuzzy Computation, Institute of Physics Pub., 1998.

[12] M. Sugeno, T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling", IEEE Trans. Fuzzy Systems, vol. 1, no. 1, pp. 7-31, 1993.

[13] F.J. Moreno-Velo, I. Baturone, S. Sánchez-Solano, A. Barriga, "Xfuzzy 3.0: A Development Environment for Fuzzy Systems", Proc. 2nd International Conference in Fuzzy Logic and Technology (EUSFLAT'2001), pp. 93-96, Leicester, 2001.