USING XFUSZY ENVIRONMENT FOR THE WHOLE DESIGN OF FUZZY SYSTEMS

I. Baturone¹, F. J. Moreno-Velo², S. Sánchez-Solano¹, A. Barriga¹, P. Brox¹, A. A. Gersnoviez¹, M. Brox¹

¹ Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica - CSIC
Avda. Reina Mercedes s/n, (Edif. CICA), E-41012, Sevilla, Spain.

² DIESIA - Escuela Politécnica Superior - Universidad de Huelva
Ctra. De Palos de la Frontera s/n, E21071, La Rábida-Huelva, Spain


© 2007 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author’s copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.
Using Xfuzzy Environment for the Whole Design of Fuzzy Systems

Iluminada Baturone, Francisco J. Moreno-Velo, Santiago Sánchez-Solano, Ángel Barriga, Piedad Brox, Andrés A. Gersnoviez, and María Brox

Abstract—From 1992, Xfuzzy environment has been improving to ease the design of fuzzy systems. The current version, Xfuzzy 3, which is entirely programmed in Java, includes a wide set of new featured tools that allow automating the whole design process of a fuzzy logic based system: from its description (in the XFL3 language) to its synthesis—in C, C++, or Java (to be included in software projects) or in VHDL (for hardware projects). The new features of the current version have been exploited in different application areas such as autonomous robot navigation and image processing.

I. INTRODUCTION

The research in fuzzy systems is so active that theoretical and practical advances are numerous. The first fuzzy systems were controllers with one or two rule bases with simple ‘if-then’ rules obtained from heuristic knowledge. Nowadays, fuzzy logic-based systems may contain fuzzy decision-making modules, fuzzy classifiers, and/or fuzzy controllers, which can be combined with non-fuzzy modules and interchange fuzzy or on-fuzzy values among them. The rules employed may be weighted by different values and may use different membership functions and operators (including linguistic hedges) to relate antecedents and consequents and to obtain the global conclusion. Besides, rule bases are usually obtained not only from heuristic knowledge but also from numerical data (the last method is particularly addressing a lot of attention for its relevance in the area of knowledge discovering), and the application of simplification as well as tuning methods to the obtained rules are also becoming usual practices.

This increase in complexity has motivated the evolution of the Xfuzzy environment. While Xfuzzy 1 focused on describing and simulating simple fuzzy controllers, the current Xfuzzy 3 uses a formal specification language, named XFL3, which facilitates the translation of complex rules expressed linguistically by allowing the use of rule weights, any kind of connective function to relate the antecedents, and linguistic hedges that may be applied to single or connected antecedents. In addition, this language allows the inclusion of new operators defined by the user as well as defining hierarchical modular systems [1].

The different versions of Xfuzzy have been distributed freely under the GNU General Public License. The last stable version, Xfuzzy 3.0, can be downloaded from its website: http://www.imse.cnm.es/Xfuzzy. This release contains a set of CAD tools which share the XFL3 language and offer Graphical User Interfaces to ease the design flow at the stages of description, tuning, verification, and synthesis. They are the following:

(a) xfdit, which eases describing the logical structure of a fuzzy system, that is, its inputs, outputs, groups of membership functions for each variable, sets of operators for each rule base, rule bases, and the system architecture (how rule bases are interconnected).

(b) xfpkg, which eases defining the function packages, that is, the code blocks describing the parameters, mathematical expressions, and other features of membership functions, defuzzification methods, and unity and binary functions (related, respectively, to linguistic hedges and fuzzy connectives).

(c) xfdplot and xfd3plot to visualize graphically one of the outputs of the system against 2 or 3 of its inputs.

(d) xfmt, to monitor how the output values are obtained by inferring from the input ones.

(e) xfsim, to simulate how the fuzzy system behaves within the application domain.

(f) xfsi, which allows applying a wide set of supervised learning algorithms (gradient-descent, second-order, Gauss-Newton, and statistical algorithms) to any system defined by XFL3.

(g) xfc, xfc, and xfi, which, respectively, translate the description of the system in XFL3 to C, C++, and Java code.

The reader is referred to [2]-[3] to find a wider description of these tools.

This paper focuses on describing the new release, Xfuzzy 3.1, which is currently being tested and, hence, is not available yet at the website. The new features added to XFL3 and some of the tools such as xfdit and xfsiplot are described in Section II. Section III summarizes the new tools incorporated to automate the process of extracting fuzzy rule bases from numerical data, to simplify the description of a fuzzy system, and to synthesize a VHDL-based description. All these capabilities are currently being exploited by our research group to design fuzzy controllers for autonomous robots and fuzzy processors for still image and video signal
II. NEW FEATURES OF EXISTING TOOLS

A. Defining families of membership functions

A linguistic variable is defined in Xfuzzy 3 by using a type object. This definition includes the name of the type, the description of the universe of discourse (its limits and discretization), the list of associated linguistic labels, and their related membership functions. Until now, membership functions had to be “free” functions selected from a function package, that is, they were defined independently and could not be explicitly related among them. For example, the variable ‘x’ shown in Figure 1 had to be defined by XFL3 as follows:

```plaintext
type Tx [0,100] {
  very_small  xfl.triangle(-25,0,25);
  small  xfl.triangle(0,25,50);
  medium  xfl.triangle(25,50,75);
  large  xfl.triangle(50,75,100);
  very_large  xfl.triangle(75,100,125);
}
```

A new feature added to XFL3 is that membership functions can be defined now as members of a family, that is, as functions explicitly related among them because share certain parameters. For example, the variable ‘x’ in Figure 1 could be also defined now as follows:

```plaintext
type Tx [0,100] {
  family[]  xfl.triangular(25,50,75);
  very_small family[0];
  small  family[1];
  medium family[2];
  large family[3];
  very_large family[4];
}
```

Several reasons have motivated the inclusion of this new feature. Firstly, the number of parameters to define a family of membership functions is smaller, which facilitates the tuning of the fuzzy system and permits the use of some automatic learning algorithms (such as simulated annealing) that are not appropriate with a large number of parameters. Secondy, it is easier to guarantee the linguistic meaning of the membership functions after applying an automatic modification process because, by construction, they cannot evolve to a state with highly overlapped or disordered functions. Finally, the use of certain families simplifies very much the hardware synthesis. As a disadvantage, systems that use families of membership functions cannot reach the optimization degree obtained with free functions, due to the imposed constraints. In general, free membership functions are more appropriate to describe output variables, while families of membership functions are specially indicated to describe input variables. A good practice is to use free membership functions for variables for which no much information is available, then perform an automatic tuning or identification process to acquire some knowledge about it, and, finally, use this information to employ a suitable family if possible.

The tools xedit and xfpkg (in particular) have been reprogrammed to admit this new feature. As example, Figure 2 shows the way how a family of membership functions based on second-order B-splines is shown by xedit.

B. Defining crisp modules

The structure of a system is defined in Xfuzzy 3 by using a system object. This definition includes the name of the global inputs and outputs and the list of interconnected modules. Until now, all the modules had to be rule bases containing ‘if-then’ rules. A new feature added to XFL3 is that now modules can be crisp, that is, they can implement any function on its inputs that is described mathematically by the Java code of its corresponding package. This allows designing complex systems in which crisp functions such as arithmetic operations or (de)multiplexers should be performed as intermediate steps between fuzzy inferences. As example, Figure 3 shows the main window of xedit illustrating a system with fuzzy and crisp modules. The definition of this structure in XFL3 is as follows:
rulebase position (TX x : TInter inter) using wheel_opset { ... }
rulebase smoothing (TSua diff : TWheel wheel) using wheel_opset { ... }
crisp { substraction xfl.diff2(); }
system (TX x, TCQ angle : TWheel wheel){
    position(x : inter);
    substraction(inter, angle : i0);
    smoothing(i0 : wheel);}
be employed which employ a fixed user-defined partition for the input variables (equal or different): Wang&Mendel, Nauck, and Senhadji. All of them evaluate which possible combination of input fuzzy sets is the most activated by each data (if there is any). In the Wang&Mendel algorithm no rule selection is implemented, so that the extracted rule base can be very large due to the curse of the dimensionality. Nauck and Senhadji algorithms avoid this problem since they allow selecting the maximum number of rules generated according to an efficiency measure [4]-[5].

The other grid-based algorithm that can be employed with \textit{xfdm} is named Incremental Grid, which is based on the proposal in [6]. The tool allows obtaining the rules' consequents by applying or not learning. This method usually finds a better covering of the input variables than the other grid-based algorithms.

Cluster-based techniques generate simultaneously the rules and the membership functions of the variables from the clusters found. They usually generate simpler systems than those obtained by grid-based techniques but with less linguistic meaning. The tool \textit{xfdm} includes four algorithms that employ a fixed user-defined number of clusters and are based on the \textit{Hard C-means}, \textit{Fuzzy C-means} [7], \textit{Gustafson-Kessel} [8], and \textit{Gath-Geva} [9] algorithms. They finish when reaching a maximum number of iterations or a minimum variation in the obtained clusters. The tool also includes an algorithm based on the proposal in [10], named Incremental Clustering, which finds the adequate number of clusters iteratively. This algorithm is configured by specifying the radius of influence of the obtained clusters and the maximum number of clusters to obtain.

\subsection*{B. Simplifying fuzzy rule bases}

Rule bases obtained from heuristic knowledge and/or numerical data and possibly adjusted by supervised learning algorithms can often be translated into simpler systems by applying simplification algorithms. The new tool \textit{xfsp} of \textit{Xfuzzy 3.1} allows applying simplification algorithms to either the variable membership functions or the rule bases of a fuzzy system.

Figure 8 shows the main window of \textit{xfsp} when membership functions (‘types’) are selected to be simplified. It shows the three simplification processes which can be applied to them: purge mechanism, clustering and similarity-based merging method.

The purge mechanism looks for those membership functions which are not used in any rule base and eliminates them. This kind of membership functions may appear as a consequence of previous simplification processes or a non careful process of heuristic knowledge translation.

The clustering method looks for a reduced number of clusters (membership function prototypes) within the original functions. It applies the Hard \textit{C-Means} algorithm, (the clusters found are crisp) on the space formed by the parameters that define the membership functions. The optimal number of membership function prototypes can be found automatically by applying validity indexes (Dunn Separation Index, Davies-Bouldin Index, and Generalized Dunn Indexes) [11] or can be fixed by the user after the visual inspection of the original functions. Figure 9a shows how the membership functions in Figure 8 are reduced from 11 to 5 (the user has selected 5 clusters).

The similarity-based merging process is an iterative process which looks for the pair of most similar functions and replaces them by a unique function if the similarity degree is over a threshold defined by the user. It finishes when no more functions can be merged. The similarity measure employed is the one defined by Dubois and Prade in [12]. Figure 9b shows how the functions in Figure 8 are reduced from 11 to 7 by applying a threshold of 0.7.

The results of applying clustering and similarity-based simplifications are similar. Advantages of using similarity-based method are that functions of different types (a triangle with a Gaussian, for instance) can be merged (which is not possible with clustering) and that the use of a threshold value can be more intuitive for the user. As a drawback, its computational cost is higher, although this cost is significant only for very complex systems.

The tool \textit{xfsp} offers four methods to process the rules of a module: pruning, compression and expansion methods, and tabular simplification.

The compression method simply merges all the rules sharing the same consequent by connecting their antecedents disjunctively. In the other side, the expansion method implements the complementary process. They do not really perform simplification but only help the user to better understand or visualize the rule base. Simplification can be
are established by the user. If the activation degree of the rules and can eliminate: (a) the n worst rules, or (b) all the rules except for the n best rules, or (c) all the rules whose activation degree is below a threshold, where the parameter n or the threshold are established by the user.

The best rule simplification method available at xfsp is a tabular algorithm developed by some of the authors, which is based on an extension of the Quine-McCluskey algorithm of Boolean design. Tabular simplification is applied to each set of rules with the same consequent (although, for the case of r consequents, r-1 simplifications could be done by using the condition else for the r-th consequent). It selects the best ‘prime implicants’ to cover a consequent, which is equivalent to find the simplest rule associated with the considered consequent [13]. The linguistic hedges available at XFL3 are exploited to better express the resulting rule. In particular, the linguistic hedges ‘not equal to’ (≠), ‘greater or equal to’ (≥), and ‘smaller or equal to’ (≤) are used.

For example, the rule base expressed by XFL3 as follows:

```
if(i0 == S & i1 == VS) -> out = low;
if(i0 == S & i1 == S) -> out = low;
if(i0 == S & i1 == M) -> out = low;
if(i0 == M & i1 == VS) -> out = low;
if(i0 == M & i1 == S) -> out = low;
if(i0 == M & i1 == M) -> out = high;
if(i0 == M & i1 == VB) -> out = high;
if(i0 == M & i1 == B) -> out = high;
if(i0 == M & i1 == M) -> out = high;
if(i0 == M & i1 == B) -> out = high;
if(i0 == M & i1 == M) -> out = high;
if(i0 == M & i1 == B) -> out = high;
```

is simplified by xfsp to the following rule base:

```
if(i0 == S & i1 == VB) -> out = high;
if(i0 == S & i1 == VB) -> out = low;
```

C. Generating VHDL code

One of the advantages of Xfuzzy 2.0 was its capability of generating the VHDL description of a fuzzy system described by XFL. The new release of Xfuzzy 3, Xfuzzy 3.1, has already the tool xfhdl to automate the FPGA (Field Programmable Gate Array) implementation of a fuzzy system described by XFL3. The implementation follows a configurable active-rule driven architecture whose modules are defined and included into a library of parametric cells which meet the constraints of the employed synthesis tools (from Xilinx or Synopsys). The tool, whose main window is shown in Figure 10, allows the user to choose the bit sizes of the variables, and to select between: (a) memory-based or arithmetic circuits to implement the membership functions of the input variables, (b) knowledge base fixed (in a ROM) or programmable (in a RAM), (c) ROM implemented as combinational logic or distributed memory and RAM implemented as distributed or block memory.

IV. APPLICATION EXAMPLES

Xfuzzy environment is currently being used by our research group in the application areas of image processing and autonomous robotics [14][15].

Figure 11 shows several captions of the Xfuzzy tools (xfedit and xfsm) for designing a fuzzy system that de-interlaces video sequences. Description, learning, and verification tools have been specially employed to design this hierarchical system, which contains two fuzzy modules connected in cascade.

Figure 12 shows several captions of the Xfuzzy tools (xfedit and xfsm) for designing a fuzzy system that controls the traction and direction motors of a car-like robot so as to navigate towards a goal configuration with quasi-optimum paths and avoiding obstacles. Description, identification, simplification, and synthesis tools have been specially employed in this system that contains ten fuzzy modules and three crisp ones.

V. CONCLUSIONS

The new release of the Xfuzzy environment incorporates new useful features to define fuzzy systems (such as the capability of using families of membership functions and crisp modules) and to verify them (with the improvements to the tool xfplot). In addition, it includes new tools to generate fuzzy rule bases from numerical data (the tool xfim), to simplify them (the tool xfsp), and to generate VHDL code (the tool xfhdl). All these capabilities have allowed our research group to design efficient fuzzy systems for image processing and robotic applications.

REFERENCES

Figure 11. Image processing applications: (a) Description of the system. (b) Simulation. (c) Experimental results with a FPGA-based implementation.


Figure 12. Robotic applications: (a) Description of the system. (b) Simulation. (c) Experimental results with a PC-based implementation.


