Object Oriented Design and Implementation of an Inference Engine for Fuzzy Systems

José Mario García Valdez, Guillermo Licea Sandoval, Arnulfo Alanis Garza, Oscar Castillo

***Abstract*—This paper describes the design and implementation of an inference engine for the execution of Fuzzy Inference Systems (FIS), the architecture of the system is presented, and the object-oriented design of the main modules is also discussed. The engine is implemented as a component to be referenced by other applications locally or remotely as a web service. This engine is needed by our research group for the implementation of other projects, which are Internet and Web based. The distinctive characteristic of this component is the ability to define fuzzy objects and attributes.**

***Index Terms*—Fuzzy Control Systems, Fuzzy Ruled Based Systems.**

I. INTRODUCTION

Fuzzy Inference Systems have been successfully used in industrial control systems that have numerous input variables and are nonlinear. These controllers have the advantage of not needing precise information to work. FLCs are a special kind of Production Systems where the rules model a dynamic system and provide an input-output relationship. Unlike production systems in a FIS there is no need for various cycles of activation of the rules because there is only a disjoint set of input and output variables, as an advantage the inference process can be executed in parallel [5]. There are several implementations of production systems that use fuzzy logic, there is FuzzyCLIPS [11], which is an extension of CLIPS a C language production system shell, the last updated version is dated in 2004 [11], also based on Jess there is FuzzyJ [12]. Another expert system shell based fuzzy logic is FLOPS [1]. There are also many implementations of fuzzy inference engines for control systems, there is AFUZ [10], and the Fuzzy Logic Toolbox™ for use with MATLAB™ [2] and also Xfuzzy 3.0 [15] is a complete development environment for fuzzy-inference-based systems and is implemented in Java. Our proposed fuzzy inference engine is mainly used in control systems, but we plan to also use it for other applications, for example for modeling fuzzy attributes in a user model for an adaptive hypermedia system, these applications are Internet and Web based. The inference engine is implemented as a .NET framework component to be referenced by other applications locally or remotely via Web Services. The rest of this paper is organized as follows: In section 2 we present a review of production systems and their fuzzy extensions, and also we focus on fuzzy inference systems for control, in section 3 the functional requirements and the overall architecture are explained, in section 4 the object oriented design of the system is presented. A case study is presented in section 5, to conclude with a summary and future work which is presented in section 6.

Manuscript received October 30, 2006.

J. Mario Garcia Valdez is with the Tijuana Institute of Technology, Division of Graduate Studies and Research, calzada tecnologico, S/N, Tijuana, BC 22379 Mexico :664-682-72-79; e-mail: (mariog@tectijuana.mx) and a Student from the University of Baja California, Chemistry Science and Engineering School. Tijuana BC, 22390 Mexico..

G. Licea Sandova is with University of Baja California, Chemistry and Engineering School, calzada tecnologico, 14418, Tijuana BC, 22390 Mexico: 664-6822790. (e-mail: glicea@uabc.mx).

A. Alanis and Oscar Castillo are with the Tijuana Institute of Technology, Division of Graduate Studies and Research, calzada tecnologico, S/N, Tijuana, 22379 Mexico :664-682-72-79; email: (ocastillo@tectijuana.mx).

II. PRODUCTION SYSTEMS AND FUZZY MODELS

A. Traditional Production Systems

Production Systems represent knowledge in form of IF-THEN rules, which specify actions that will be executed when certain conditions are met. Also known as rule based systems many implementations consist mainly of these three components [4][5]:

1) Production Rules (PR). A set of production rules (also known as IF-THEN rules) having a two part structure; the antecedent, conformed by a set of conditions and a consequent set of actions.
2) Working Memory (WM). Represents the current knowledge or facts that are known to be true so far. These facts are tested by the antecedent conditions of the rules and the consequent part can change them.
3) I Inference Engine (IE). This interpreter matches the conditions in the production rules with the data/instantiations found in the WM, deriving new consequences.

The basic operation of these systems is described as a cycle of three steps [4]:

1) Recognize: Find which rules are satisfied by the current state of the WM. The antecedent part of the productions consists of a set of clauses connected by AND operators, when all these clauses have matching data on the WM the production has a chance of firing.
2) Conflict Resolution: Only one production can be fired at

a time, so when two or more rules can be fired concurrently a conflict occurs. Among the production rules found in the first step, choose which rules should fire.

3) A ctions: Change the working memory by performing the actions specified in the consequent part of all the rules selected in the second step. Changes occur by adding or deleting elements of the WM.

This cycle continues until no further production rules can be fired. This control strategy is data driven because whenever the antecedent part is satisfied the rule is recognized, this strategy is also named chain-forward. The other strategy is chain-backward in this case the work is done from the conclusion to the facts, to chain-backward; goals in working memory are match against consequents of the production rules.

A drawback that has been recognized in these traditional productions systems is that some times rules are not fired in the *Recognize* step because no appropriate match exists in the WM. Partial matching of rules is not possible and this can be a limitation in some systems because premature termination of the cycle is not desired. An approach to handle partial matching is using fuzzy logic [5]. In the next section we present a review of the extension of production systems with fuzzy logic.

B. Fuzzy Production Rules

Fuzzy production rules use fuzzy logic sets to characterize the variables and terms used in the propositions of the rules. Fuzzy production rules or fuzzy IF-THEN rules are expressions of the form IF antecedent THEN consequent, where the antecedent is a proposition of the form "x is A" where x is a linguistic variable and A is a linguistic term. The truth value of this proposition is based on the matching degree between x and A. Propositions are connected by AND, OR operators and also can be negated with the NOT operator. Some implementations of fuzzy rule-based systems also include other kinds of data types in their propositions, for example the FLOPS system includes fuzzy numbers, hedges, and non fuzzy data types (integers, strings and float) [1]. Depending on the form of the consequent, two main types of fuzzy production systems are distinguished [3]:

- Linguistic fuzzy model: where both the antecedent and consequent are fuzzy propositions.
- Takagi-Sugeno fuzzy model: the antecedent is a fuzzy proposition; the consequent is a crisp function.

As before, other non-fuzzy consequents can also be implemented, or also the consequent can be an action for instance the execution of a method or the addition of new data to a system state.

Linguistic Variables (LVs) are variables that can be assigned linguistic terms as values, i.e. if we define a linguistic variable SPEED we can assign it the linguistic terms SLOW, MEDIUM or FAST. The meaning of these linguistic terms is defined by their membership functions (MFs). LVs can be defined as a 5-tuple LV=<v,T,X,g,m> where v is the name of the variable, T is the set of linguistic terms of v, X is the domain (universe) of v, g is a syntactic rule to generate linguistic terms, m is a semantic rule that assigns to each term t its meaning m(t), which is a fuzzy set defined in X.

C. Fuzzy Inference Systems

Fuzzy Inference Systems (FISs) also called Fuzzy Models are fuzzy production systems used for modeling input-output relationships. From this input-output view, Babuška [3] describes that these systems as *"flexible mathematical functions which can approximate other functions or just data (measurements) with a desired accuracy"*. Fuzzy Production Rules define the relationship between input and output variables. Input variables are defined in the antecedent part of the rule and the consequent part defines the output variables.

These FIS are used mainly in control systems, and are basically composed of five modules Fig.1 [3]:

1) Rule Base. The set of fuzzy production rules.
2) Database. Where the membership functions are defined.
3) Fuzzy Inference Engine. This module executes the fuzzy inference operations.
4) Fuzzifier. This interface transforms the inputs of the systems (numerical data) into linguistic values.
5) Defuzzifier. This interface transforms the fuzzy results into numerical data.

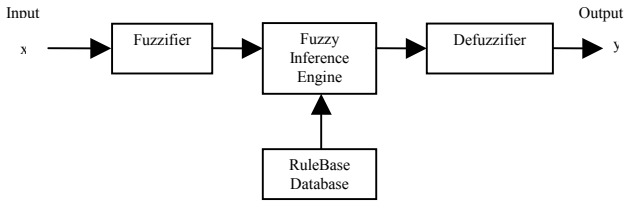

Fig.1 Main components of a Fuzzy Inference System

Usually the Rule Base and Data Base modules are collectively called the Knowledge Base module, this Knowledge Base is different to the WM, because output variables can not be input variables so the inference process is not a cycle. The steps involved in fuzzy inference in a FIS are [7]:

1) Compare the input variables with the membership functions in the antecedent, to obtain the membership values of each linguistic term. This step is frequently called fuzzification.
2) Compose through a specific T-Norm operator (mainly max-min or max-product) the membership values to obtain the degree of support of each rule.
3) Generate the qualified consequence (fuzzy or numeric) of each rule depending on the degrees of support. These outputs are then aggregated to form a unified output.
4) Then the output fuzzy set is resolved or defuzzified to a single numeric value.

Three main types of fuzzy inference systems can be described: *Tsakumoto*: The output is the average of the output weight of each rule, induced by the degree of support of each rule, the min-max or min-product with the antecedent and the

membership functions of the output. The membership functions used in this method must be non-decrease monotonic. Mamdani: The output is calculated by applying the min-max operator to the fuzzy output (each equal to the minimum support degree and the membership function of the rule). Several schemes have been proposed to choose the numeric output based on the fuzzy output; these include the centroid area method, area bisection, maximum mean and maximum criteria.

Sugeno: The output of each rule is a linear combination of the input variables plus a constant term, and the output is the average of the support degree of each rule.

This project focuses in the implementation of an engine for Fuzzy Inference Systems, but in the future can be extended to a more general Fuzzy Production Rules Inference Engine.

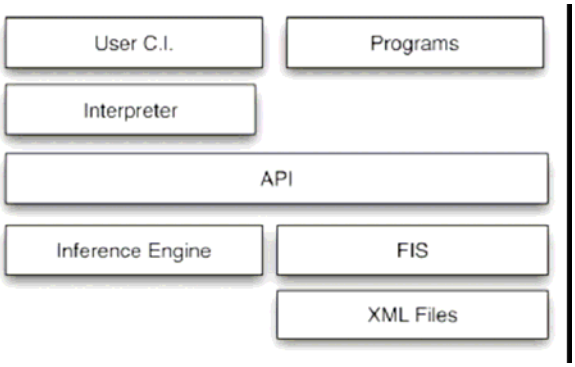

Fig.2 Application Functional Layers

III. Architecture

The functional requirements for this project are briefly described next. We need a component for the .NET framework [8] to be used in future projects which are Internet based; in addition we need control over the code to implement new extensions. The language selected for the implementation is C# [13], an object oriented language based on C++ and Java. For the first version of the engine we focused on a Fuzzy Inference Engine (FIE) that is going to be used to implement Fuzzy Controllers, Fuzzy Inference Systems need to be stored as XML [9] files, to be stored in a repository of Fuzzy Inference Systems. An Application Programming Interface (API) is implemented. At this moment a Graphical User Interface (GUI) is not yet implemented Fig. 2 shows the functional layers of the architecture. There are two core modules:

Inference Engine. C# Interfaces have been defined so different inference methods can be implemented. For instance a `FuzzyLogicOperator` Interface can be implemented with different classes; in this case a `MamdaniOperator` has been defined.

FIS Module. This module manages the knowledge base, it has I/O functions to store or read from file (XML) the definition of the FIS. The FIS component can also be referenced from a program as a component.

IV. Design

The main UML [14] Class Diagram is shown in Fig. 3, this diagram is an abstraction of the Fuzzy Inference System described in section II.C. The FIS has three collections: Fuzzy Rules, Linguistic Variables (LVs) and Composition this is the collection of qualified member functions given as the output of each rule in the inference process. The `FIS` class also has attributes to describe and identify the system, Name, Description, Owner, Status, and also methods to save and load the FIS from an XML file. The FIS data components are mainly static but some values change at runtime, this attributes have the `current` prefix and normally are not saved to file. In the next subsections FIS components are described with more detail.

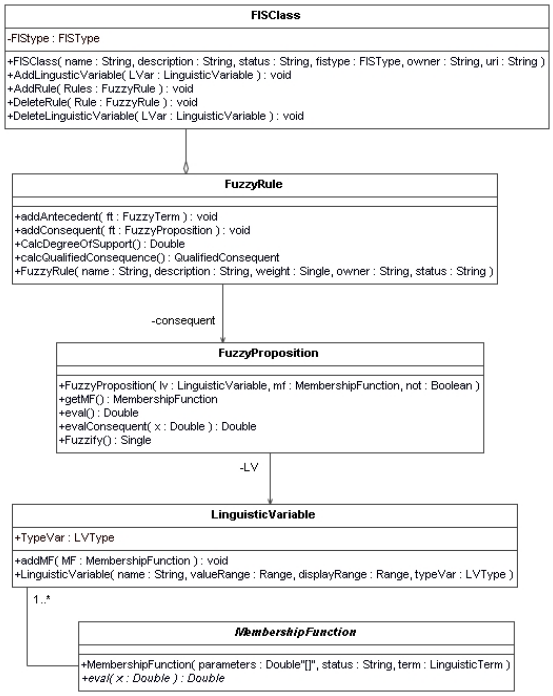

Fig. 3 Main UML class diagram.

A. Linguistic Variables

As defined in section II.C each LV can be assigned one or more Linguistic Terms each defined by a Membership Function. The `MembershipFunction` class is defined as an abstract class with an abstract method `eval()`, subclasses must implement this method appropriately. Subclasses implemented so far are: `Trapezoidal`, `Triangular`, `Gaussian` and `Bell`, other membership functions can be defined as subclases of `MembershipFunction`. The `LinguisticVariable` class has a `currentCrispValue` attribute used in the inference process. Lingustic Variables can be of input or output type.

B. Fuzzy Rules

Fuzzy rules have one `FuzzyPropositon` instance as consequent, and they can have a composite `FuzzyProposition` as antecedent, this is explained in the next section. Fuzzy Propositions where described earlier in section

II.B as having the form "x is A" where x is a linguistic variable and A is a linguistic term. Following this, each `FuzzyProposition` instance is associated with an instance of `LinguisticVariable` and an instance of `MembershipFunction` which in turn has a corresponding `LinguistcTerm`. There are two important methods of Fuzzy Rules:

CalcDegreeOfSupport() this method evaluates the `antecedent:FuzzyProposition` for the current crisp value of the Linguistic Variables returning the degree of support.

CalcQualifiedConsequence() this method also calculates the degree of support of the antecedent but also returns a `QualifiedConsequence` instance, this class has a reference to the FuzzyProposition and the Current Degree of Support. This is nedded for the final compostion of the output member functions.

C. Antecedent

The antecedent part of a Rule can include a composite Fuzzy Proposition, as seen in Fig. 4 the antecedent of a `FuzzyRule` is a `FuzzyTerm` Interface, There are two classes implementing this interface, `FuzzyProposition` and `MamdaniOperator`; The `MamdaniOperator` class relates two `FuzzyTerm` instances this means other `MamdaniOperator` or other `FuzzyProposition`. This pattern implements a collection of Fuzzy Propositions associated with their corresponding operators.

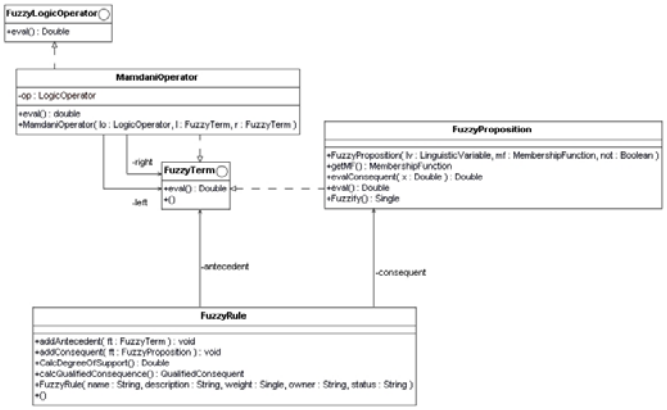

Fig. 4 FuzzyRule class diagram.

D. Inference

Next we describe the implementation of each of the fuzzy inference steps defined in section II.C. , we assume that fuzzy rules have been defined with their corresponding propositions. Also input linguistic variables have a current value defined.

Compare: Each Fuzzy Proposition has an associated Membership Function. Each Membership Function has an eval() method that takes an input linguistic variable and returns the corresponding membership value.

Compose: As described in section IV.C the antecedent part of the rule can be a composite FuzzyProposition, each FuzzyTerm implementation has a eval() method, for instance a MamdaniOperator instance uses max-min operator to compose the values of the two FuzzyTerms it includes. This is done recursively until the Degree of support for the fuzzy rule is returned.

Generate: In our implementation *Compare*, *Compose* and *Generate* are done in a single step. This process is exposed in Fig. 5 with the Sequence diagram of the method `calcQualifiedConsequence()`, it starts with an empty `Composition` collection, and adds the Qualified Consequence of each rule to the collection, for example:

```
Composition.Add(Rule[i].calcQualifiedConsequence());
```

For each rule, the `calcQualifiedConsequence()` returns a `QualifiedConsequence` object which has the `DegreeOfSupport` of each rule and the `LinguisticVariable` of the consequent. For this each rule calls the `eval()` method of his antecedent. The `antecedent` instance is an implementation of the `FuzzyTerm` interface. After this cycle, the `Composition` collection contains the aggregated output of each rule; in the next step a single value is calculated.

Deffuzzyfication. In this step the centroid area for the composition of Qualified Outputs is calculated using:

$$ {}^{x}COA = \frac{\int_x \mu A(x)\, x\, dx}{\int_x \mu A(x)\, dx} $$

When the area of the composition is calculated for each x in the domain only the maximum value is considered, this is done so overlapped membership functions don't conflict with each other. This method returns a single crisp value as a result of the inference process.

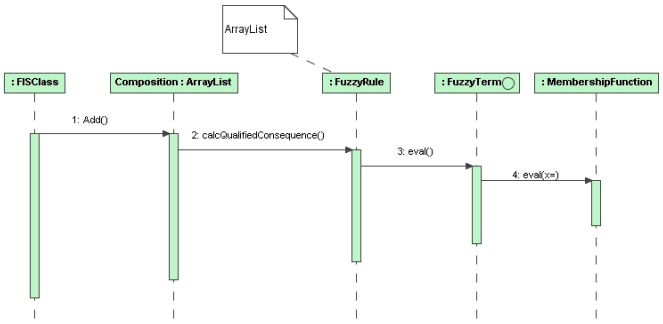

Fig. 5. Sequence diagram for `calcQualifiedConsequence()`.

V. CASE STUDY

As a case study of the FIS component we implemented the "Diner for Two" example found in [2], in this system a `tip` between 5% and 25% is calculated with two input variables `service` and `food`. This example uses the classes described in previous sections. First we have to create the `LinguisticVariables` and their corresponding membership functions, only the `service` Linguistic Variable is shown, also the name of the classes are abbreviated to limit the space needed:

```
//Service
LV service = new LV("Service", new Range(0, 10), new
Range(0, 10), LVType.In);
```

```
//Membership Functions
MF poor     = new Gaussian(new Double[] { 1.5, 0 },
"test", new LTerm("Poor"));
MF good     = new Gaussian(new Double[] { 1.5, 5 },
"test", new LTerm("Good"));
MF excelent = new Gaussian(new Double[] { 1.5, 10 },
"test", new LTerm("Excelent"));
//Asign Membership Functions to the LV
service.addMF(poor);
service.addMF(good);
service.addMF(excelent);
```

Next the rules are defined:

```
FRule r1 = new FRule("active");
r1.addAntecedent(
new MamdaniOperator( LogicOperator.OR,
  new FProposition(service,poor,true),
  new FProposition(food    , rancid, true)));
r1.addConsequent( new FProposition(tip,cheap,
true));

FRule r2 = new FRule("active");
r2.addAntecedent( new FProposition(service, good,
false));
r2.addConsequent( new FuzzyProposition(tip,average,
false));

FRule rule3 = new FuzzyRule("active");
r3.addAntecedent(
new MamdaniOperator(LogicOperator.OR,
  new FuzzyProposition(service, excelent, false),
  new FuzzyProposition(food, delicious, false)));
r3.addConsequent( new FuzzyProposition(tip,generous,
false));
```

The `rule.addAntecedent()` method takes a `FuzzyTerm` as imput in r1 this is a Mamdani OR operator, and r2 only takes a simple `FuzzyProposition`. The `FuzzyProposition` constructor takes three arguments, first a `LinguisticVariable` then a `MembershipFunction`, the third argument indicates if this proposition is negated, that is, if the `NOT` operator is assigned. This is the basic definition of the FIS, at this point we can save or load this structure. The next step is to give some values to the input Linguistic Variables;

```
service.CurrentCrispValue = 2;
food.CurrentCrispValue = 5;
```

For this example we make the Composition collection explicit, so the functionality is better exhibited;

```
ArrayList Composition = new ArrayList();

Composition.Add(rule1.calcQualifiedConsequence());
Composition.Add(rule2.calcQualifiedConsequence());
Composition.Add(rule3.calcQualifiedConsequence());
```

With the Composition collection we can deffuzify the output:

```
Double num   = 0;
Double denom = 0;
Single delta = 0.1F;

for
(Single x = tip.Rge.Min; x <= tip.Rge.Max; x +=
delta){

  Double m = 0;
  foreach (QualifiedConsequent q in Composition){
     Double mtest = q.eval(x);
       if (mtest > m)
           m = mtest;
     }
   num += x * m;
   denom += m;
   }
Double result;
if (denom == 0)
  result = (tip.Rge.Min + tip.ge.Max) / 2;
else
  result =  num / denom;
```

As explained earlier only the maximum value of the collection `QualifiedConsequent` is considered. In the variable `result` is the Deffuzyfied output.

VI. Summary and Further Work

In this work we have presented the design of an inference engine, which can be used to implement FIS systems. Our future work will focus on the implementation of different inference mechanisms. Also the definition of other types of Membership Functions is also needed. The object oriented approach of this project is well suited for the better understanding of the FIS components; the practical use of this system is for academic implementations so the students can manipulate the components more easily.

References

[1] W. Siler and J. Buckley, Fuzzy Expert Systems and Fuzzy Reasoning, John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[2] Math Works Inc., "Fuzzy Logic Toolbox User's Guide", 2006. www.mathworks.com.

[3] R. Babuŝka., "Fuzzy Systems, Modeling and Identification", Delft University of Technology, Department of Electrical Engineering Control Laboratory, Mekelweg 4, P.O. Box 5031, 2600 GA Delft, The Netherlands.

[4] R.J. Brachman and H.J. Levesque, Knowledge Representation and Reasoning. Morgan Kaufmann. 2004.

[5] A.Konar, Computational Intelligence, Springer-Verlag, Berlin Heidelberg. 2005.

[6] L. Wang, A Course in Fuzzy Systems and Control, Prentiece-Hall, NJ . 1997.

[7] D. Dubois and H. Prade. Fuzzy sets and systems: theory and applications, of Mathematics in science and engineering. Academic Press. Volume 144, 1980.

[8] Microsoft Inc. .NET information home page http://www.microsoft.com/net/.

[9] [9] Bray, T., et al (editors), "Extensible Markup Language (XML) 1.0 (Second Edition)",W3C Recommendation 6 October 2000, W3C [Online] Available: http://www.w3.org/TR/2000/ REC-xml-20001006.

[10] A.J. Abebe , "AFUZ: a tool to build fuzzy rule based systems" [Online] Available: http://www.xs4all.nl/%7Edpsol/datamachine/afuz1.htm.

[11] FuzzyCLIPS CLIPS With Fuzzy Extensions. [Online] Available: http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyClips/fuzzyCLIPSIndex2.html.

[12] FuzzyJ A Fuzzy Java ToolKit [Online] Available: http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit2.html.

[13] C# Wikipedia [Online] Available: http://en.wikipedia.org/wiki/C_Sharp_programming_language.

[14] G.Booch, I.Jacobson, J.Rumbaugh. The UML User Guide, Addison-Wesley, 1998.

[15] XFuzzyTeam and IMSE-CNM, Xfuzzy 3.0 [Online] Available: http://www.imse.cnm.es/Xfuzzy/Xfuzzy_3.0/index.html

José Mario García Valdez. Bachelor in Industrial Engineering from Tijuana Institute of Technology. Current Doctoral Student at the Autonomous University of Baja California. His research interest include: Adaptive Hypermedia, Adaptive Educational Environments and Social Systems.

Guillermo Licea Sandoval. Bachelor of Science in Computer Science from University of Baja California in Ensenada, a Master of Science in Computer Science from Tijuana's Institute of Technology, and a Ph. D. in Computer Science from Scientific Research and Higher Education Center of Ensenada (CICESE). Professor at University of Baja California in Tijuana since 1991. His research interests are: Computer Supported Cooperative Work, Software Engineering Tools, and Mobile Application Development.

Arnulfo Alanis Garza. Bachelor in Computer Engineering Systems from Technological Institute of San Luis Potosi. Candidate to the PhD Degree in Computer Science from Polytechnic University of Valencia, Full time Researcher in Tijuana Institute of Technology since 2005. Current areas of interest include: Intelligent Agents, Expert Systems, Robotics and Networks and Fault Tolerance Systems.

Oscar Castillo. Professor of Computer Science in the Graduate Division, Tijuana Institute of Technology, Tijuana, Mexico. In addition, he is serving as Research Director of Computer Science and head of the research group on fuzzy logic and genetic algorithms. Currently, he is President of HAFSA (Hispanic American Fuzzy Systems Association) and Vice-President of IFSA (International Fuzzy Systems Association) in charge of publicity. Prof. Castillo is also Vice-Chair of the Mexican Chapter of the Computational Intelligence Society (IEEE). Prof. Castillo is also General Chair of the IFSA 2007 World Congress to be held in Cancun, Mexico. He also belongs to the Technical Committee on Fuzzy Systems of IEEE and to the Task Force on "Extensions to Type-1