XFSML: UN LENGUAJE DE MODELADO DE SISTEMAS DIFUSOS BASADO EN XML

Ramón Santano Ruiz [1], Francisco José Moreno Velo [2]

[1] *Universidad de Huelva*, ramón.santano@alu.uhu.es

[2] *Dpto. Tecnologías de la Información, Universidad de Huelva*, francisco.moreno@dti.uhu.es

Resumen

Este trabajo presenta un nuevo lenguaje de modelado de sistemas difusos llamado XFSML. Se trata de un lenguaje basado en XML que se propone como punto de partida para desarrollar un lenguaje de modelado estándar en la comunidad fuzzy. La principal característica del lenguaje es su gran capacidad expresiva, así como su independencia respecto a plataformas, herramientas o lenguajes de programación concretos.

Palabras Clave: XML, Fuzzy Modeling, Fuzzy Software.

1 INTRODUCCIÓN

En el ámbito científico los sistemas difusos han sido utilizados con éxito en múltiples campos como la ingeniería de control, el tratamiento de imágenes, el reconocimiento de patrones, la minería de datos o los sistemas de toma de decisiones, entre otros. Esto ha generado una floreciente comunidad científica dedicada al desarrollo teórico y práctico de los sistemas difusos, con un nivel de producción de cientos de artículos científicos al año.

La situación es bastante diferente en cuanto a la implantación de los sistemas difusos en el ámbito industrial. La mayor parte de las aplicaciones de sistemas difusos en la industria están dedicadas a resolver problemas de automática y control. En el resto de campos de aplicación, los sistemas difusos tienen, hoy por hoy, una presencia en la industria casi residual. Esto contrasta con la situación de otros paradigmas de la Inteligencia Artificial, como las redes neuronales o los sistemas basados en reglas clásicas, que sí se han incorporado con éxito a aplicaciones del ámbito empresarial (*business intelligence*), como los sistemas de extracción de conocimiento (KDD), los sistemas expertos o los sistemas de ayuda a la toma de decisiones.

Algunos autores, como D. Nauck [8], consideran que los sistemas difusos no han alcanzado el mismo grado de implantación que otros paradigmas de la Inteligencia Artificial debido entre otras razones a la falta de una herramienta de diseño de sistemas difusos que sea aceptada como estándar dentro de esta área. Dicha herramienta estándar, de existir, contribuiría a difundir el uso de sistemas difusos en sectores más amplios de la industria.

En nuestra opinión, es complicado que la comunidad fuzzy se decante abiertamente por el uso de una herramienta en concreto, debido fundamentalmente a la variedad de aplicaciones existentes y a sus diferentes campos de aplicación. Por ejemplo, entre las aplicaciones de ámbito comercial podemos citar a las herramientas TILShell, de Togai Infralogic, FIDE, de Aptronix, o FuzzyTech, de Inform. También de ámbito comercial son los módulos de lógica fuzzy (*toolboxes*) de entornos como MatLab o Mathematica. En el ámbito académico se han desarrollado librerías de funciones difusas, como FFLL[12] o FuzzyJ [11], y numerosas herramientas dedicadas a campos como la minería de datos (p.e. KEEL [4]), la generación de datos imperfectos (p.e. Nip1.5 [3]), la generación de clasificadores difusos (p.e. NEFCLASS [9]), controladores difusos (p.e. NEFCON [10]), o el diseño de sistemas difusos (p.e. FisPro[5], GUAJE [1] o Xfuzzy [2]).

Consideramos que es muy difícil que una nueva herramienta consiga sustituir a las aplicaciones existentes, dada la diversidad de enfoques, campos de aplicación, lenguajes de programación y plataformas de ejecución que utilizan estas aplicaciones. Nuestro punto de vista es que resultaría mucho más adecuado avanzar hacia un lenguaje de representación de sistemas difusos que se aceptara como estándar dentro de la comunidad fuzzy. De esta forma, los desarrolladores de software sólo tendrían que adaptar sus aplicaciones al uso de dicho lenguaje de representación y se mantendría la diversidad de enfoques y plataformas existentes en la actualidad.

Siguiendo esta filosofía, este trabajo presenta un lenguaje de modelado de sistemas difusos que pretende ser lo suficientemente expresivo como para poder ser utilizado

como lenguaje de representación por la mayoría de herramientas existentes en el ámbito fuzzy.

2 EL LENGUAJE XFSML

XFSML es un lenguaje de modelado de sistemas difusos basado en XML. El nombre es el acrónimo de compleX/eXtensible Fuzzy System Modeling/Marckup Language. El punto de partida en el diseño del lenguaje ha sido la experiencia acumulada en el diseño del lenguaje XFL3 [6], que se definió como lenguaje de representación del entorno Xfuzzy 3.

El lenguaje está centrado en la descripción de la estructura de los sistemas difusos y no en su descripción funcional. Esto quiere decir que el lenguaje no incluye la definición de las funciones asociadas a los operadores difusos (como funciones de pertenencia, t-normas, t-conormas, o métodos de *defuzzificación*), sino la forma en que estos operadores se combinan para describir un determinado sistema. Estas funciones son referenciadas por su nombre y se asume que la descripción funcional debe encontrarse en alguna librería externa. Esto permite ampliar libremente las funciones disponibles para cada operador difuso (por ejemplo, definiendo nuevos tipos de funciones de pertenencia).

La característica fundamental del lenguaje es su alta capacidad expresiva. Esto quiere decir que se han incluido numerosas formas de representar sistemas difusos, de manera que el lenguaje permita modelar la mayor parte de los sistemas difusos utilizados por la comunidad científica y técnica.

El uso de XML permite además que XFSML sea independiente de herramientas, plataformas o lenguajes de programación concretos. De esta forma se permite que los desarrolladores adapten sus aplicaciones a este nuevo lenguaje de representación, sea cual sea el lenguaje de programación utilizado o el sistema operativo del equipo en el que se pretenda ejecutar.

2.1. ESTRUCTURA GLOBAL

La definición de un sistema difuso en XFSML consiste en un fichero XML formado por una única etiqueta (*fuzzysystem*). Esta etiqueta tiene el atributo "*name*" con el nombre del sistema difuso y contiene otras cuatro etiquetas que desarrollan las descripción del sistema:

- ***domains*** : describe los dominios (universos de discurso) de las diferentes variables que formen parte del sistema.
- ***partitions*** : describe las particiones difusas (es decir, los conceptos lingüísticos) asociados a cada dominio.
- ***relations*** : describe las relaciones difusas entre dominios. Esto permite expresar relaciones como X > Y en términos difusos.
- ***modules*** : describe los módulos (bases de conocimiento) que forman el sistema. La descripción global del sistema corresponde al módulo llamado "*system*".

A continuación se muestra un esquema del contenido de un sistema difuso descrito en XFSML.

```
<?xml version="1.0"?>
<fuzzysystem name="Example" xmlns:xsi= … >
        <domains>
        ...
        </domains>
        <partitions>
        ...
        </partitions>
        <relations>
        ...
        </relations>
        <modules>
        ...
        </modules>
</fuzzysystem>
```

Figura 1: Esquema general de un especificación XFSML.

2.2. DOMINIOS

Los dominios describen los universos de discurso de las variables que aparecen en el sistema. En el caso de los controladores difusos las variables suelen ser de tipo real. Sin embargo, en otro tipo de aplicaciones, como la minería de datos, es frecuente trabajar con variables de tipo entero o con valores enumerados. Por tanto, XFSML se ha diseñado para admitir tres tipos de dominio: reales, enteros y enumerados. Para definir estos dominios se utilizan tres etiquetas: *real*, *integer* y *enum*. Los dominios reales y enteros se describen por medio de los atributos *min* y *max*. Por su parte, los dominios enumerados se describen por medio de etiquetas *elem*.

```
<domains>
 <real name="dom1" min="-5.0" max="5.0" />
 <integer name="dom2" min="1" max="10" />
 <enum name="dom3">
   <elem value="Iris-setosa" />
   <elem value="Iris-virginica" />
   <elem value="Iris-versicolor" />
 </enum>
</domains>
```

Figura 2: Ejemplo de definiciones de dominios.

```
<partition name="part1" domain="dom1" >
  <freelabel name="SMALL" library="std"
             function="trapezoid">
    <param value="-10.0" />
    <param value="0.0" />
    <param value="40.0" />
    <param value="60.0" />
  </freelabel>
  <freelabel name="BIG" library="std"
             function="trapezoid">
    <param value="40.0" />
    <param value="60.0" />
    <param value="100.0" />
    <param value="110.0" />
  </freelabel>
</partition>
```

```
<partition name="part2" domain="dom2" >
  <family name="fam1" library="std"
          function="FAMtriangular">
    <param value="-10.0" />
    <param value="0.0" />
    <param value="10.0" />
  </family>

  <familylabel name="NEG" family="fam1"
               index="0" />
  <familylabel name="ZE" family="fam1"
               index="1" />
  <familylabel name="POS" family="fam1"
               index="2" />
</partition>
```

Figura 3: Ejemplo de definición de particiones usando funciones de pertenencia libres y familias de funciones

2.3. PARTICIONES

Las particiones describen los conceptos lingüísticos que se pueden asociar a los distintos dominios. Corresponden a la definición de las funciones de pertenencia asociadas a las variables en los sistemas difusos.

Al igual que el lenguaje XFL3 [7], las funciones de pertenencia definidas en XFSML pueden ser de dos tipos: funciones de pertenencia libres (que no tienen ningún tipo de restricción con respecto a otras funciones de pertenencia) y funciones de pertenencia de una familia. Las familias permiten definir conjuntos de funciones de pertenencia que presentan restricciones entre sí. Por ejemplo, funciones de pertenencia que compartan parámetros, funciones de pertenencia que deban ser simétricas, funciones de pertenencia que deben mantener un orden, etc.

La descripción funcional de estas funciones de pertenencia no está incluida en el lenguaje XFSML. Esto hace que el lenguaje no tenga restricciones en cuanto a la forma de las funciones de pertenencia que se pueden utilizar. Toda referencia a una función de pertenencia (o a una familia) se hace utilizando el nombre de la función y el nombre de la biblioteca en la que está descrita. Por otra parte, los parámetros de definición de estas funciones se describen como cadenas (*strings*), de manera que sea la función la responsable de verificar el valor de los parámetros utilizados.

La Figura 3 muestra un ejemplo de una partición formada por dos funciones libres de tipo trapezoide y de una partición formada por una familia de tres funciones triangulares que comparten los parámetros.

2.4. RELACIONES

Las relaciones permiten describir conceptos lingüísticos que afectan a dos dominios. Esto permite expresar enunciados del tipo "*el valor de X es aproximadamente igual al de Y*" o "*el valor de X es mucho mayor que el de Y*". Matemáticamente, las relaciones se definen como funciones de pertenencia sobre un universo de dos dimensiones.

Hasta el momento, las relaciones difusas no han sido demasiado utilizadas en el diseño de sistemas difusos, probablemente porque ninguna herramienta software ha dado soporte a este tipo de concepto. La inclusión de relaciones en XFSML abre la posibilidad de utilizar estos conceptos a la hora de representar el conocimiento.

La descripción de las relaciones es bastante similar a las particiones, aunque en este caso se definen sobre dos universos de discursos y no existen las familias. La Figura 4 muestra un ejemplo de descripción de relaciones.

```
<relation domain1="dom1" domain2="dom1">
  <label name="IgualQue" library="std"
         function="RELEqual">
     <param value="0.5" />
  </label>
  <label name="MuchoMayorQue"
       library="std" function="RELMuchGT">
     <param value="8.0" />
  </label>
</relation>
```

Figura 4: Ejemplo de definición de una relación.

```
<tree name="modulo1">
 <inputs>
   <input name="X1" partition="part2" /> <input name="X2" partition="part2" />
 </inputs>
 <outputs>  <output name="Y" partition="part1"/>  <outputs>
 <and library="std" function="product" />
 <defuz library="std" function="MaxLabel" />
 <root>
  <switch var="X1">
   <case label="NEG”> <node> <decision var="Y" label="SMALL" /> </node> </case>
   <case label="ZE">
    <node><switch var="X2">
      <case label="NEG”> <node> <decision var="Y" label="SMALL" /></node> </case>
      <case label="ZE"> <node> <decision var="Y" label="BIG" /> </node> </case>
      <case label="POS"> <node> <decision var="Y" label="SMALL" /> </node> </case>
    </switch></node>
   </case>
   <case label="POS"> <node> <decision var="Y" label="BIG" /> </node> </case>
  </switch>
 </root>
</tree>
```

Figura 5: Ejemplo de definición de un árbol de decisión difuso.

2.5. MÓDULOS

Los módulos permiten representar las bases de conocimiento que forman el sistema difuso. El lenguaje XFSML ofrece diversas formas de representación, como los conjuntos de reglas difusas, los árboles de decisión o los módulos jerárquicos. En general, un sistema difuso puede estar formado por uno o varios módulos. La descripción global del sistema es aquella asociada al módulo llamado "*system*".

El lenguaje XFSML incluye una amplia variedad de representaciones de conocimiento. Esto permite optar al diseñador por el uso de estructuras de diferente grado de complejidad. De cara a los desarrolladores de software, los diferentes tipos de módulos permiten desarrollar técnicas específicas (de identificación, de optimización, de simplificación, de generación de código, etc.) para cada tipo de estructura. Por ejemplo, el código de implementación de un árbol de decisión es diferente del de un conjunto de reglas.

Los siguientes sub-apartados describen los diferentes tipos de módulos incluidos en XFSML. La Figura 5 muestra un ejemplo de uno de estos tipos de módulos.

2.5.1. Árboles de decisión difusos

Los árboles de decisión son módulos de N entradas y una salida. El conocimiento se expresa en forma de árbol donde los nodos hoja son decisiones del tipo “Y es LB” y los nodos internos reflejan preguntas sobre el valor de una variable de entrada.

La definición de un árbol de decisión difuso está contenida en una etiqueta “*tree*”. El contenido de la etiqueta está formado por otras cinco etiquetas en el siguiente orden:

- ***inputs***: Descripción de las variables de entrada. Cada variable se describe mediante una etiqueta “*input*” que incluye el nombre de la variable y su partición.
- ***outputs***: Descripción de las variables de salida. Cada variable se describe mediante una etiqueta “*output*” que incluye el nombre de la variable y su partición.
- ***and***: Descripción del operador AND utilizado en el proceso de inferencia.
- ***defuz***: Descripción del método de *defuzzificación* utilizado en el proceso de inferencia.
- ***root***: Descripción del contenido del árbol a partir del nodo raíz. Los nodos terminales se representan con una etiqueta “*decisión*”. Los nodos internos se representan mediante una etiqueta “*switch*” con un lista de hijos formada por etiquetas “*case*” que a su vez pueden tener como hijo a un nodo terminal o a un nodo interno.

2.5.2. Matrices de reglas difusas

Las matrices permiten describir de manera compacta las bases de conocimiento formadas por dos variables de entrada, una variable de salida y reglas del tipo “IF X1 es LB1 AND X2 es LB2 THEN Y es LB3”. La matriz está

formada por todas las combinaciones posibles entre los valores de las etiquetas de la primera variable y los de la segunda.

La definición de una matriz de reglas difusas está contenida en una etiqueta "*array*", que a su vez contiene otras cinco etiquetas: *inputs*, *outputs*, *and*, *defuz* y *cells*. Las cuatro primeras son idénticas a las utilizadas en los árboles de decisión. La etiqueta "*cells"* describe el contenido de las celdas de la matriz. Cada fila corresponde a un valor (una función de pertenencia de la partición asociada) de la primera variable de entrada y está descrita mediante la etiqueta "*row*". El contenido de cada fila es un conjunto de celdas, correspondientes a cada valor (función de pertenencia de la partición asociada) de la segunda variable de entrada. Las celdas se describen con etiquetas "*cell*" con el atributo "*label*" que expresa el valor de la variable de salida para esa celda de la matriz.

2.5.3. Tablas de reglas difusas conjuntivas

Las tablas de reglas difusas definen módulos de N entradas y M salidas. Las filas corresponden a reglas del tipo "IF X1 es LB1 AND X2 es LB2 AND X3 es LB3 THEN Y1 es LB7 AND Y2 es LB8". Las tablas se definen mediante la etiqueta "*table*", que está formada por las etiquetas *inputs*, *outputs*, *and*, *defuz* y *rows*. Esta última etiqueta contiene la descripción del contenido de las reglas de la tabla. Cada fila se describe por una etiqueta "row", formada a su vez por etiquetas "*cell*" que describen las celdas de la tabla.

2.5.4. Conjuntos de reglas difusas conjuntivas

Los conjuntos de reglas difusas conjuntivas describen reglas similares a las expresadas en forma de tabla, con la diferencia de que los antecedentes de las reglas pueden no incluir valores de algunas variables de entrada. Se describen por medio de la etiqueta "*ruleset*" que contiene las etiquetas *inputs*, *outputs*, *and*, *defuz* y *rules*. La etiqueta *rules* está compuesta por una lista de etiquetas "*rule*", formada por un antecedente (etiqueta "*antecedent*") y un consecuente (etiqueta "*consequent*"). Tanto el antecedente como el consecuente está formado por un conjunto de términos (etiquetas "*term*") que expresan información de tipo "X es LB". Por ejemplo <term var="X1" label="Positive"/>.

2.5.5. Listas ordenadas de reglas difusas conjuntivas

Las listas ordenadas de reglas difusas conjuntivas definen módulos de N entradas y M salidas donde las reglas son del tipo "IF ... ELSEIF ... ELSEIF ... ELSE...". La definición de una lista ordenada de reglas difusas conjuntivas está contenida en una etiqueta "*rulelist"*. El contenido del módulo está formado por las etiquetas *inputs*, *outputs*, *and*, *defuz* y *rules*. La etiqueta *rules* está compuesta por una lista de etiquetas "*rule*", formada por un antecedente (etiqueta "*antecedent*") y un consecuente (etiqueta "*consequent*"). Tanto el antecedente como el consecuente está formado por un conjunto de términos (etiquetas "*term*"). Las reglas deben entenderse en el orden en el que están escritas (IF ... ELSEIF ... ELSEIF). La etiqueta *rules* puede contener opcionalmente una última etiqueta "*else*" que está formada sólo por el consecuente (no tiene la etiqueta "*antecedent*").

2.5.6. Conjuntos de reglas difusas complejas

Los conjuntos de reglas complejas se diferencian de los conjuntos de reglas conjuntivas en la forma de los antecedentes de las reglas. El antecedente de las reglas conjuntivas se interpreta como la operación AND sobre la lista de términos que forma el antecedente. Por su parte, las reglas complejas tienen antecedentes formados por combinaciones de operadores lógicos (AND, OR y NOT), por modificadores lingüísticos (STRONGLY, MOREORLESS), términos complejos (por ejemplo, "X > LB" o "X <LB") y por relaciones entre variables (por ejemplo, "MayorQue(X,Y)".

Los conjuntos de reglas complejas se definen por medio de la etiqueta "*complexruleset*", que a su vez contiene las etiquetas *inputs*, *outputs*, *operators*, *defuz* y *rules*. La etiqueta *operators* contiene la definición de las funciones asociadas a cada modificador lingüístico utilizado en las reglas. La etiqueta *rules* contiene la lista de reglas (etiquetas *rule*), cuyo antecedente está formado por una expresión compleja.

2.5.7. Listas ordenadas de reglas difusas complejas

Las listas ordenadas de reglas complejas permiten definir reglas del tipo "IF ... ELSEIF ... ELSEIF ... ELSE...", donde el antecedente de las reglas puede ser una expresión compleja como las explicadas en el apartado anterior.

Para definir una lista ordenada de reglas difusas complejas se utiliza la etiqueta "*complexrulelist*", que está formada por las etiquetas *inputs*, *outputs*, *operators*, *defuz* y *rules*. La etiqueta *rules* puede contener opcionalmente una última etiqueta "*else*" que está formada sólo por el consecuente.

2.5.8. Sistemas difusos jerárquicos

Los módulos jerárquicos son módulos formados por llamadas a otros módulos Estos módulos se definen por medio de la etiqueta "*hierarchy*". El contenido de la etiqueta está formado por otras tres etiquetas en el siguiente orden: *inputs*, *outputs* y *calls*. Las etiquetas *inputs* y *outputs* se diferencian de las de otros módulos en que las variables se definen con su nombre y su dominio en vez de su nombre y partición. La etiqueta *calls* está formada por una lista de etiquetas "*call"* que describen las llamadas que forman la estructura jerárquica. Cada etiqueta *call* tiene un atributo "*module*" (que referencia al módulo al que se llama), una etiqueta *inputs* (que indica las variables

de entrada del módulo) y una etiqueta *outputs* (que indica cuales son las variables de salida del módulo).

2.5.9. Módulos no difusos

Los módulos no difusos son módulos dedicados a operaciones aritméticas entre datos no difusos (suma, multiplicación,...). Se suelen utilizar como módulos dentro de una estructura jerárquica. Para definirlos se utiliza la etiqueta "*crisp*", que incluye a su vez tres etiquetas: *inputs*, *outputs* y *description*. Las etiquetas *inputs* y *outputs* definen las variables del módulo con su nombre y dominio. La etiqueta *description* contiene la referencia a la librería, función y parámetros que definen el módulo no difuso.

3 CONCLUSIONES

Los sistemas difusos tienen un gran desarrollo a nivel académico pero su uso en otros ámbitos es aún escaso. En nuestra opinión, la adopción por parte de la comunidad *fuzzy* de un lenguaje estándar de modelado de sistemas difusos contribuiría a difundir el uso de estos sistemas y facilitaría su implantación fuera del ámbito académico.

La definición de este lenguaje estándar debería ser promovida y aprobada por el Comité Técnico de Sistemas Difusos (*Fuzzy Systems TC*) de la IEEE Computational Inteligence Society, como órgano más influyente dentro de la comunidad *fuzzy*. Dicho Comité Técnico cuenta actualmente con un equipo de trabajo (*Task Force on Fuzzy Systems Software*) que podría constituir un marco adecuado para la definición de dicho estándar.

El lenguaje XFSML, presentado en este trabajo, puede ser un buen punto de partida para construir dicho estándar. Este lenguaje está basado en XML y se apoya en la experiencia acumulada en el desarrollo del entorno Xfuzzy 3 y de su lenguaje de especificación XFL3. Además, en este momento es fácil introducir modificaciones ya que se encuentra en los primeros pasos de su definición.

Agradecimientos

Este trabajo ha sido financiado parcialmente por los Proyectos de Investigación TEC2011-24319, TEC2008-04920 y TIN2008-06681-C06-06 del Ministerio de Ciencia y Tecnología y el Proyecto de Excelencia P08-TIC-03674 de la Junta de Andalucía.

Referencias

[1] J. M. Alonso, L. Magdalena: GUAJE - A Java environment for generating understandable and accurate models. En: *Actas del XV Congreso Español Sobre Tecnologías y Lógica Fuzzy (ESTYLF-2010)*, Huelva, pp. 399-404, 2010.

[2] I. Baturone, F. J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, P. Brox, A. Gersnoviez, M. Brox: Using Xfuzzy Environment for the Whole Design of Fuzzy Systems. En: *Proc. IEEE International Conference on Fuzzy Systems*, Londres, 2007.

[3] J. M. Cadenas, J.V. Carrillo, M. C. Garrido, E. Muñoz: Nip1.5 : una herramienta software para la generación de conjuntos de datos con imperfección para minería de datos. En: *Actas del XV Congreso Español Sobre Tecnologías y Lógica Fuzzy (ESTYLF-2010)*, Huelva, pp. 411-416, 2010.

[4] J. Derrac, A. Fernández, J. Luengo, S. García, L. Sánchez, J. Alcalá, F. Herrera: KEEL: una herramienta software para el análisis de sistemas difusos evolutivos. En: *Actas del XV Congreso Español Sobre Tecnologías y Lógica Fuzzy (ESTYLF-2010)*, Huelva, pp. 417-422, 2010.

[5] S. Guillaume, B. Charnomordic: Learning interpretable Fuzzy Inference Systems with FisPro. En: *Information Sciences* 181(20), pp. 4409-4427, 2011.

[6] F. J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone, D.R. López: XFL3: a new fuzzy system specification language, En: *Advanced in Scientific Computing, Computational Intelligence and Applications*, World Scientific and Engineering Society Press, Singapur, pp. 361-366, 2001.

[7] F. J. Moreno-Velo, I. Baturone, S. Sánchez-Solano , A. Barriga: The Parametric Definition of Membership Functions in XFL3. En: *Proc. IEEE International Conference on Fuzzy Systems*, Budapest, 2004.

[8] D. Nauck: GNU Fuzzy. En*: Proc. IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE 2007)*, Londres, pp. 1-6, 2007.

[9] D. Nauck, R. Kruse: NEFCLASS-J - a Java-based soft computing tool. En*: Intelligent Systems and Soft Computing: Prospects, Tools and Applications, ser. Lecture Notes in Artificial Intelligence*, B. Azvine, N. Azarmi, and D. Nauck, Eds. Berlin: Springer-Verlag, no. 1804, pp. 143-164, 2000.

[10] A. Niirnberger, D. Nauck, R. Kruse: Neuro-fuzzy control based on the NEFCON-model: Recent developments. En: *Soft Computing*, vol. 2, no. 4, pp. 168-182, 1999.

[11] R. A. Orchard: Fuzzy reasoning in Jess: The Fuzzy J Toolkit and Fuzzy Jess. En: *Proceedings of the Third International Conference on Enterprise Information Systems (ICEIS 2001)*, Setubal (Portugal), pp. 533-542, 2001.

[12] M. Zarozinski: An Open Source Fuzzy Logic Library. En: *AI Game Programming Wisdom*, Ed. Charles River Media, pp. 90-103, 2002.