XFSML: An XML-based Modeling Language for Fuzzy Systems

F.J. Moreno-Velo
Departamento de Tecnologías de la Información
Universidad de Huelva
Huelva, SPAIN

A. Barriga, S. Sánchez-Solano, I. Baturone
Instituto de Microelectrónica de Sevilla
Centro Nacional de Microelectrónica - CSIC
Sevilla, SPAIN

***Abstract*— This paper presents a new modeling language for fuzzy systems called XFSML. It is an XML-based language and it is proposed as a starting point for the definition of a standard modeling language in the fuzzy community. The main features of the language are its high expressiveness and its independence from specific platforms, tools or programming languages.**

Keywords- XML; Fuzzy Modeling; Fuzzy Software

I. INTRODUCTION

Fuzzy systems have been used successfully in many research fields such as control engineering, image processing, pattern recognition, data mining or decision-making systems, among others. This has created a thriving scientific community devoted to theoretical and practical development of fuzzy systems, with a production level of hundreds of scientific papers a year.

The situation is quite different regarding the industrial applications of fuzzy systems. Most applications of fuzzy systems in the industry are dedicated to solving problems of automatic control. In other fields of application, the presence of fuzzy systems in the industry is almost residual. This contrasts with the situation of other paradigms of artificial intelligence, like neural networks or classical rule-based systems, which have been successfully incorporated into business applications, such as knowledge discovery on databases, expert systems or decision making systems.

Some authors, such as D. Nauck [1], consider that fuzzy systems have not yet enjoyed widespread success in the domain of business applications among other reasons due to the lack of a standard and easy-to-use fuzzy software. This standard tool would help to spread the use of fuzzy systems in the wider industry.

In our opinion, it's difficult for the fuzzy community to adopt a particular tool as the standard fuzzy system design software, mainly due to the variety of existing tools and their different fields of application. For instance, there are several commercial tools like TILShell by Togai Infralogic Inc., FIDE by Aptronix, or FuzzyTech by Inform. There are also fuzzy logic modules (toolboxes) for commercial environments such as MatLab or Mathematica. Regarding the open source software, we can mention some libraries of fuzzy functions (e.g. FFLL [2] or FuzzyJ [3]) and many tools dedicated to different fields like data mining (e.g. KEEL [4][5]), imperfect data generation (e.g. Nip1.5 [6]), fuzzy classifiers (e.g. NEFCLASS [7]), fuzzy controllers (e.g. NEFCON [8]), or the design of fuzzy systems (e.g. FisPro [9], GUAJE [10] or Xfuzzy [11]).

We consider that it would be very difficult to replace all the existing tools by a new standard software, giving the diversity of approaches, application areas, programming languages and execution platforms used in these existing tools. Our view is that it would be more appropriate to define a modeling language for fuzzy system that could be accepted as standard in the fuzzy community. Thus, software developers could adapt their applications to use this common fuzzy system representation. In this case, a designer could easily use a fuzzy system generated with one of these tools into other tool.

Following this philosophy, this paper presents a new modeling language for fuzzy systems (named XFSML) which is expressive enough to be used as representation language for most tools in the fuzzy area.

II. THE XFSML LANGUAGE

XFSML is a new modeling language for fuzzy systems based on XML. The name is the acronym of compleX/eXtensible Fuzzy Systems Modeling/Marckup Language. The definition of the language is based on the experience in the design of XFL3 [12], the representation language used in Xfuzzy3.

The language is focused on the description of the structure of fuzzy systems and not in its functional description. This means that the language does not include the definition of the functions associated with fuzzy operators (such as membership functions, t-norms, t-conorms, or defuzzification methods), but the way in which these operators are combined to describe a given system. These functions are referenced by the name of the function and the name of the external library which contains their definitions. This way, the available functions for each fuzzy operator can be freely extended by creating and

This work was supported in part by the Projects TEC2011-24319, TEC2008-04920, and TIN2008-06681-C06-06 from the spanish Ministerio de Ciencia y Tecnología, and by the Project P08-TIC-03674 from the Andalusian Regional Government, with support from the European Regional Development Fund (ERDF) and the European Social Fund (ESF).

referencing new external libraries (including, for example, the definition of new types of membership functions or defuzzification methods).

The main feature of XFSML is its high expressiveness. This means that numerous ways to represent fuzzy systems have been included, so the language can model most of the fuzzy systems used by the scientific and technical community.

As XFSML is based on XML, it's independent of tools, platforms or specific programming languages. So, software developers could easily adapt their applications to this new representation language, regardless of the programming language or operative system used on the computer where the application is intended to run.

A. General structure

The definition of a fuzzy system in XFSML is an XML file consisting of a single element (*fuzzysystem*). This root element has the attribute *name* with the name of the fuzzy system and contains four child nodes describing the system:

- *domains*: containing the descriptions of the domains (universes of discourse) of the different variables that are part of the system.
- *partitions*: describing the fuzzy partitions (i.e., linguistic concepts) related with each domain.
- *relations*: describing fuzzy relationships between pair of domains. This allows to express fuzzy concepts such as *X is approximately equal to Y*.
- *modules*: containing the modules (knowledge bases) that compose the system. The overall description of the fuzzy system corresponds to the module called *system*.

Figure 1 shows an outline of the contents of a fuzzy system described in XFSML.

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

Figure 1. Overview of an XFSML specification

B. Domains

A domain describe the universe of discourse of one or more variables included in the fuzzy system. XFSML supports domains of three different data types: real numbers, integer numbers and nominal values. Fuzzy controllers usually work with real numbers. However, other applications such as data mining often work with integer or nominal variables.

In order to define the domains of a fuzzy system, the *domains* element in the XML definition is formed by a list of three kinds of nodes:

- *real*: describing an interval of real numbers. This element is defined in terms of the attributes *name* (the identification of the domain), *min* (the minimum of the interval) and *max* (the maximum of the interval).
- *integer*: describing an interval of integer numbers. The element contains also the attributes *name* (the name of the domain), *min* (the minimum of the interval) and *max* (the maximum of the interval).
- *enum*: describing a domain of nominal values. The element contains the attribute *name* (the name of the domain) and a list of *elem* tags. The *elem* tags are nodes with the attribute *value* (defining a nominal value of the domain).

Figure 2 shows an example of these three kinds of domain definitions.

```
<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>
```

Figure 2. Example of domain definitions.

C. Partitions

A fuzzy partition describes a set of membership functions on a certain domain. These membership functions can be interpreted as linguistic concepts. Fuzzy partitions are used to define the data type of linguistic variables, as it contains both the universe of discourse of the variable and their linguistic concepts.

As XFL3 language [13], XFSML includes two kinds of membership functions: free membership functions (which have no restrictions with respect to other membership functions) and membership functions of a family. A family is a set of membership functions which have restrictions to each other. For example, functions which shares some parameters, functions which must be symmetrical, or functions which must remain in order. Families can be used to express these constraints in the structure of the fuzzy system. This way, when using algorithms for inducing or tuning membership functions, these constraints are maintained by construction.

The functional description of free membership functions or families of membership functions is not included in XFSML. This means that XFSML does not include a predefined set of functions. Any reference to a membership function (or a family) is done using the function (or family) name and the name of the external library in which this function (or family) is defined. Moreover, the parameters defining these functions are described as strings so that the function is responsible for checking the correctness of these values.

The *partitions* node in the XML definition is formed by a list of *partition* elements. A *partition* element has two attributes: *name* (the name of the fuzzy partition) and *domain* (the universe of discourse of the fuzzy partition). The element contains a list of three kinds of child nodes:

- *freelabel*: describing a free membership function. The element contains the attributes *name* (the name of the linguistic concept), *library* (the name of the external library), and *function* (the name of the function in the library), and a list of *param* tags. The *param* tags are nodes with the attribute *value* (defining a value for a parameter).
- *family*: describing a family of membership functions. The element is formed by the attributes *name*, *library* and *function* and a list of *param* tags.
- *familylabel*: describing a reference to the membership function of a family. The element has the attributes *name* (the name of the linguistic concept), *family* (the name of the previously defined family) and *index* (the reference to the membership function in the family).

Figure 3 shows the example of two fuzzy partitions. The first one is composed by two trapezoidal free membership functions. The second partition is defined by a family of three triangles sharing their parameters.

```
<partitions>
 <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="fam" library="std"
          function="FAMtriangular">
    <param value="-10.0" />
    <param value="0.0" />
    <param value="10.0" />
  </family>
  <familylabel name="NEG" family="fam" index="0"/>
  <familylabel name="ZE"  family="fam" index="1"/>
  <familylabel name="POS" family="fam" index="2"/>
 </partition>
</partitions>
```

Figure 3. Example of the definition of two fuzzy partitions

D. Relations

Fuzzy relations are defined as membership functions over a pair of domains. Given two variables, X and Y, the membership function R(X,Y) indicates the degree of relationship between X and Y. Fuzzy relations can be used to express concepts such as "X is approximately equal to Y" or "X is much greater than Y". So far, relations have not been extensively used in fuzzy modeling, probably because software tools have not supported this kind of concept.

The *relations* element in the XML definition is composed of a list of *relationset* nodes. A *relationset* node is quite similar to a partition, but in this case relating to two domains. This node has two attributes, *domain1* and *domain2*, and a list of *relation* nodes. Each *relation* node describes a fuzzy relation. The *relation* element contains the attributes *name* (the name of the concept), *library* (the name of the external library) and *function* (the name of the two-dimensional membership function in the library), and a list of *param* tags defining the parameters of the function.

The following figure shows an example of the definition of a relation set with two fuzzy relations (named *EqualTo* and *MuchGreaterThan*) over a pair of domains (named *dom1* and *dom2*).

```
<relations>
 <relationset domain1="dom1" domain2="dom2">
   <relation name="EqualTo" library="std"
         function="RELEqual">
     <param value="0.5" />
   </relation>
   <relation name="MuchGreaterThan"
           library="std" function="RELMuchGT">
     <param value="8.0" />
   </relation>
 </relationset>
</relations>
```

Figure 4. Example of the definition of a relation set

E. Modules

Modules are the knowledge bases that define the behavior of the fuzzy system. In general, a fuzzy system contains one or more modules. The global description of the fuzzy system is defined in a module called *system*.

XFSML includes a wide variety of knowledge representations such as fuzzy rule sets, fuzzy decision trees or hierarchical modules. When modeling a fuzzy system, a designer should choose the most adequate structure to represent the system behavior. Software developers may also implement specific algorithms for each kind of structure. For example, there exists different algorithms for inducing fuzzy decision trees, fuzzy rule sets or ordered list of fuzzy rules. The code generation for decision trees or rule set are different. Each type of module has its own optimization and simplification techniques.

The following subsections describe the different types of modules included in XFSML.

1) Fuzzy decision trees

Decision trees are modules with N input variables and one output variable. The knowledge is expressed as a tree where the leaf nodes are decision on the value of the output variable and the internal nodes are questions on the value of an input variable.

```
<tree name="module1">
 <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>
```

Figure 5. Example of the definition of a fuzzy decision tree.

The XML definition of a fuzzy decision tree is an XML element called *tree*. The element has the attribute *name* (the name of the knowledge base) and five child nodes in the following order:

- *inputs*: describing the input variables of the module. This node contains a list of *input* tags. Each tag define an input variable and includes the attributes *name* (the name of the variable) and *partition* (the linguistic description of the variable).
- *outputs*: describing the output variables of the module. The node contains a list of *output* tags with the attributes *name* and *partition*.
- *and*: describing the AND operator used in the inference process. The element contains the attributes *library* and *function* (and a list of *param* tags when necessary) to reference the function assigned to the operator.
- *defuz*: describing the defuzzification method. The element contains the attributes *library* and *function* and a list of *param* tags if necessary.
- *root*: describing the contents of the tree from the root node. Leafs nodes are represented by *decision* tags with the attributes *var* (name of the output variable) and *label* (the membership function assigned to the decision). The inner nodes describe a question on an input variable and are defined by a *switch* node. This node contains the attribute *var* (the name of the input variable to test) and a list of *case* elements. The *case* elements includes the attribute *label* (the membership function of the input variable) and a new child node.

An example of the definition of a fuzzy decision tree is shown in figure 5.

2) Arrays of fuzzy rules

The arrays of fuzzy rules are a compact way to express a knowledge base with two input variables, one output variable and a complete set of conjunctive rules. The rows of the array are related with each membership function of the first input variable while the columns refer to the membership function of the second input variable. Each cell contains a membership function of the output variable and represents a fuzzy rule like "*if INPUT1 is* MF_i *and INPUT2 is* MF_j *then OUTPUT is* MF_c".

The XML description of an array of fuzzy rules is contained in element called *array*. This element includes five nodes: *inputs*, *outputs*, *and*, *defuz* and *cells*. The first four

```
<array name="module2">
 <inputs>
    <input name="X1" partition="part1" /> <input name="X2" partition="part1" />
 </inputs>
 <outputs>
    <output name="Y" partition="part2"/>
 <outputs>
 <and library="std" function="product" />
 <defuz library="std" function="FuzzyMean" />
 <cells>
    <row> <cell label="NEG” /> <cell label=”NEG” /> <cell label=”ZE”  /> </row>
    <row> <cell label="NEG” /> <cell label=”ZE”  /> <cell label=”POS” /> </row>
    <row> <cell label="ZE”  /> <cell label=”POS” /> <cell label=”POS” /> </row>
 </cells>
</array>
```

Figure 6. Example of the definition of an array of fuzzy rules (the partition *part1* is assumed to have three membership functions)

```
<table name="module3">
 <inputs>
   <input name="X1" partition="part1" /> <input name="X2" partition="part1" />
   <input name="X3" partition="part1" />
 </inputs>
 <outputs>
   <output name="Y1" partition="part2"/> <output name="Y2" partition="part2"/>
 </outputs>
 <and library="std" function="product" />
 <defuz library="std" function="FuzzyMean" />
 <rows>
   <row><cell label="SM"/><cell label="SM"/><cell label="MD"/><cell label="POS"/><cell label="POS"/></row>
   <row><cell label="SM"/><cell label="MD"/><cell label="BG"/><cell label="NEG"/><cell label="ZE" /></row>
   <row><cell label="MD"/><cell label="SM"/><cell label="SM"/><cell label="POS"/><cell label="ZE" /></row>
 </rows>
</table>
```

Figure 7. Example of the definition of a table of conjunctive fuzzy rules

nodes are identical to those used by decision trees. The node *cells* describes the contents of the array. This node is composed by *row* elements. Each *row* element contains a list of *cell* nodes with the attribute *label* indicating the conclusion (the output membership function) of the fuzzy rule assigned to that cell. Figure 6 shows an example of an array of fuzzy rules.

3) Tables of conjunctive fuzzy rules

A table of conjunctive fuzzy rules defines a module with N input variables and M output variables. The content of the knowledge base is formed by a set of rules. The antecedent of these rules is a conjunction of terms including all the input variables while the consequent includes terms for all the output variables. The difference between tables and arrays of fuzzy rules is that tables have no limit on the number of input or output variables and the set of rules could be incomplete. That is, some combination of terms should not be present on the set of rules.

A table in XFSML is defined by a *table* element with five nodes: *inputs*, *outputs*, *and*, *defuz* and *rows*. The first four nodes are similar to those used in arrays and decision trees. The *rows* node contains the set of conjunctive fuzzy rules. Each rule is described by a *row* element, which is formed by a list of *cell* tags. These *cell* tags have the attribute *label* with the membership function selected in that cell of the table. Figure 7 shows the definition of a table of fuzzy rules.

4) Sets of conjunctive fuzzy rules

A set of conjunctive fuzzy rules defines a module with N input variables and M output variables. The difference with respect to the table of rules is that, in this case, the antecedent and consequent of the rules may be incomplete. That is, the rule can include not all of the variables.

The XML definition of a set of rules is formed by a *ruleset* element with five nodes: *inputs*, *outputs*, *and*, *defuz* and *rules*. The *rules* node contains a list of *rule* elements. Each *rule* element includes an *antecedent* node and a *consequent* node. Both nodes contain a list of terms. The *term* tags have the attributes *var* and *label*, representing a condition like "*X is BIG*".

5) Ordered lists of conjunctive fuzzy rules

An ordered list of rules is a knowledge base with N input and M output variables. The difference with respect to a set of rules is that the evaluation of the rules follow a prescribed order. Therefore, the activation degree of a rule depends on the

```
<ruleset name="module4">
<inputs>
   <input name="X1" partition="part1" /> <input name="X2" partition="part2" />
   <input name="X3" partition="part3" />
 </inputs>
 <outputs>
   <output name="Y1" partition="part3"/> <output name="Y2" partition="part2"/>
 </outputs>
 <and library="std" function="product" />
 <defuz library="std" function="FuzzyMean" />
 <rules>
   <rule>
     <antecedent> <term var="X1" label="LB1" /> <term var="X3" label="LB5" /> </antecedent>
     <consequent> <term var="Y2" label="LB0" /> </consequent>
   </rule>
   <rule>
     <antecedent> <term var="X2" label="LB0" /> </antecedent>
     <consequent> <term var="Y2" label="LB3" /> <term var="Y1" label="LB1" /> </consequent>
   </rule>
 </rules>
</ruleset>
```

Figure 8. Example of a set of conjunctive fuzzy rules

```
<rulelist name="module5">
<inputs>
   <input name="X1" partition="part1" /> <input name="X2" partition="part2" />
   <input name="X3" partition="part3" />
 </inputs>
 <outputs>
   <output name="Y1" partition="part3"/> <output name="Y2" partition="part2"/>
 </outputs>
 <and library="std" function="product" />
 <defuz library="std" function="FuzzyMean" />
 <rules>
   <rule>
     <antecedent> <term var="X1" label="LB1" /> <term var="X3" label="LB5" /> </antecedent>
     <consequent> <term var="Y2" label="LB0" /> </consequent>
   </rule>
   <rule>
     <antecedent> <term var="X2" label="LB0" /> </antecedent>
     <consequent> <term var="Y2" label="LB3" /> <term var="Y1" label="LB1" /> </consequent>
   </rule>
   <else>
     <consequent> <term var="Y2" label="LB3" /> <term var="Y1" label="LB1" /> </consequent>
   </else>
 </rules>
</rulelist>
```

Figure 9. Example of an ordered list of conjunctive fuzzy rules

activation of the previous rules. The list can be interpreted as the construction "if ... elseif ... elseif ... else...".

The definition of an ordered list of rules in XFSML consists in a *rulelist* node. The node contains five elements: *inputs*, *outputs*, *and*, *defuz* and *rules*. These elements are similar to those of the *ruleset* node, but the *rules* element can contain a final *else* node describing the consequent of a default rule.

6) Sets of complex fuzzy rules

Sets of complex rules differ from the sets of conjunctive rules in the complexity of the antecedent of the rules. The antecedent of a conjunctive rule is formed by a list of terms joined with the AND operator. On the other hand, the antecedent of a complex rule is formed by the combination of logical operators (AND, OR and NOT), linguistic hedges (Strongly, MoreOrLess, Slightly), fuzzy relations (e.g., *X is greater than Y*) and complex fuzzy terms (e.g., *X is greater than MF0*).

A set of complex rules is defined with a *complexruleset* element. This element includes five nodes: *inputs*, *outputs*, *operators*, *defuz* and *rules*. The *operators* element can include the definition of the operators *and*, *or*, *not*, *strongly*, *moreorless*

```
<complexruleset name="module6">
 <inputs>
   <input name="X1" partition="part1" /> <input name="X2" partition="part2" />
   <input name="X3" partition="part3" />
 </inputs>
 <outputs>
   <output name="Y1" partition="part3"/> <output name="Y2" partition="part2"/>
 </outputs>
 <operators>
    <and package="std" function="product" />
    <or package="std" function="max" />
    <strongly package="std" function="pow"><param value="2"></strongly>
 </operators>
 <defuz package"std" function="WeightedFuzzyMean" />
 <rules>
   <rule>
     <antecedent>
       <and_expr>
         <eq_expr var="X1" label="lb0"/>
         <or_expr><stronglyeq_expr var="X2" label="Z" /><ge_expr var="X1" label="lb3" /></or_expr>
         <strongly_expr> <eq_expr var="X2" label="P"/> </strongly_expr>
         <rel_expr relation="MuchGreaterThan" var1="X1" var2="X3"/>
        </and_expr>
     </antecedent>
     <consequent> <term var="Y2" label="LB0" /> </consequent>
   </rule>
 </rules>
</complexruleset>
```

Figure 10. Example of a set of complex fuzzy rules

```
<hierarchy name="system">
 <inputs>
   <input name="X1" domain="dom1" /><input name="X2" domain="dom2" /><input name="X3" domain="dom2" />
 </inputs>
 <outputs>
   <output name="Y1" domain="dom3"/> <output name="Y2" domain="dom4"/>
 <outputs>
 <calls>
   <call module="module1" >
     <inputs> <input var="X1"/> <input var="X2"/> </inputs>
     <outputs> <output var="Y1"> <output var="I0"> </outputs>
   </call>
   <call module="module2">
     <inputs> <input var="I0"/> <input var="X3"/> </inputs>
     <outputs> <output var="Y2"> </outputs>
   </call>
 </calls>
</hierarchy>
```

Figure 11. Example of a hierarchical module

and *slightly*. These definitions includes the attributes *library* and *function* and a list of *param* tags if necessary. The antecedent of each rule is a complex expression defined by the following elements: *and_expr*, *or_expr*, *not_expr*, *strongly_expr*, *moreorless_expr*, *slightly_expr*, *eq_expr*, *ne_expr*, *gt_expr*, *ge_expr*, *lt_expr*, *le_expr*, *stronglyeq_expr*, *approxeq_expr*, *slightlyeq_expr*, and *custom_expr*. Figure 10 includes an example of a set of complex rules.

7) Ordered lists of complex fuzzy rules

Ordered lists of complex rules are similar to the ordered list of conjunctive rules, but using complex antecedents within the rules as described above. The XFSML definition of an ordered list of complex rules is contained in a *complexrulelist* element.

8) Hierarchical fuzzy systems

Hierarchical modules are composed of a list of calls to other modules. This way, the structure of a fuzzy system can be defined by a modular composition of several knowledge bases. The calls can refer to any kind of module, even to another hierarchical module. The modular structure is generated by using the output variables of a call as the input variables of other calls. In order to make these connections, the output and the input variable must be defined on the same domain (but can be described by different partitions).

The XML description of a hierarchical module is based on a *hierarchy* node. This node contains three elements: *inputs*, *outputs* and *calls*. The *inputs* node contains a list of *input* elements, describing the input variables with the attributes *name* and *domain*. The *outputs* node includes a list of *output* elements, describing the output variables with their name and domain. Finally, the *calls* node contains a list of *call* elements. Each *call* element includes the attribute *module* (with the name of the called module) and the child nodes *inputs* (describing the input variables of the module call) and *outputs* (describing the output variables of the module call). An example of a hierarchical module definition can be found in Figure 11.

9) Crisp modules

Crisp modules are non-fuzzy modules devoted to arithmetic operations between variables of the fuzzy system (e.g. addition, multiplication, ...). These modules are typically used within hierarchical modules (e.g., the input variable of call can be the difference between the output variables of two previous calls).

The XFSML definition of a crisp module is included into a *crisp* node. This node contains three elements: *inputs* (defining the input variables with their names and domains), *outputs* (defining the output variables) and *description*. The description element contains the attributes *library* and *function* and a list of *param* tags if necessary. Figure 12 shows an example of a crisp module.

III. CONCLUSIONS

Fuzzy systems are widely used by the scientific community, but their use in industrial applications is still low. In our opinion, a standard modeling language for fuzzy systems could help to spread the use of fuzzy systems beyond the academic field.

```
<crisp name="Summation">
 <inputs>
   <input name="X1" domain="dom1" /> <input name="X2" domain="dom2" /><input name="X3" domain="dom2" />
 </inputs>
 <outputs>
   <output name="Y1" domain="dom1"/>
 <outputs>
 <description library="std" function="sum"> <param value=”3” /> </description>
</crisp>
```

Figure 12. Example of a crisp module

The language presented in this work can be a good starting point for creating such a standard. XFSML uses the XML syntax and supports numerous ways for defining fuzzy systems (from simple to complex systems). At this point, it's easy to introduce changes into XFSML as it is in the early stages of its definition.

REFERENCES

[1] D. Nauck, "GNU Fuzzy," Proc. IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE 2007), London (UK), pp. 1-6, 2007.

[2] M. Zarozinski, "An open source fuzzy logic library," AI Game Programming Wisdom, Ed. Charles River Media, pp. 90-103, 2002.

[3] R. A. Orchard, "Fuzzy reasoning in Jess: The Fuzzy J Toolkit and Fuzzy Jess," Proc. Third International Conference on Enterprise Information Systems (ICEIS 2001), Setubal (Portugal), pp. 533-542, 2001.

[4] J. Alcala-Fdez, L. Sanchez, S. Garcia, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernandez, F. Herrera. "KEEL: A Software Tool to Assess Evolutionary Algorithms to Data Mining Problems," Soft Computing, vol. 13 (3) pp. 307-318, 2009.

[5] J. Alcala-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," Journal of Multiple-Valued Logic and Soft Computing, vol. 17 (2-3), pp. 255-287, 2011.

[6] J.M. Cadenas, J.V. Carrillo, M.C. Garrido, R. Martínez-España, E. Muñoz, "NIP 1.5 – A tool to handle imperfect information on datasets," Murcia University, 2008 (http://heurimind.inf.um.es/heurimind/NIP-Tool.shtm)

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

[8] A. Niirnberger, D. Nauck, R. Kruse, "Neuro-fuzzy control based on the NEFCON-model: Recent developments," Soft Computing, vol. 2 (4), pp. 168-182, 1999.

[9] S. Guillaume, B. Charnomordic, "Learning interpretable fuzzy inference systems with FisPro," Information Sciences, vol. 181 (20), pp. 4409-4427, 2011.

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

[11] 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," Proc. IEEE International Conference on Fuzzy Systems, London (UK), 2007.

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

[13] F. J. Moreno-Velo, I. Baturone, S. Sánchez-Solano, A. Barriga, "The parametric definition of membership functions in XFL3," Proc. IEEE International Conference on Fuzzy Systems, Budapest, 2004.