2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Extending FML with Evolving Capabilities through a Scripting Language Approach

Giovanni Acampora, Marek Reformat and Autilia Vitiello

Abstract— **The introduction of Fuzzy Markup Language (FML) in 2004 has initiated an important trend in Computational Intelligence research: the application of new web technologies to create more flexible and hardware independent environment for deploying "fuzzy ideas". FML allows researchers and engineers to focus on problem solving activities bypassing additional difficulties related to programming or physical equipment constraints. From that moment on, many researches have been using FML and other XML-based languages for modeling and developing fuzzy systems. However, in spite of their hardware interoperability, XML languages are able to model a fuzzy system in static way and, consequently, they do not provide any support for modelling "evolving" and temporal-based fuzzy systems, as such Timed Automata based Fuzzy Controllers. To address this deficiency, this paper introduces an extension of FML called FMLScript. It is based on a scripting language concept and allows for modeling XML-based systems that can dynamically modify their configurations. As the consequence, a better expressive power can be achieved when compared with static modelling approaches. This is shown in a case study involving a smart grid control.**

I. INTRODUCTION

IN 1965, Lofti Zadeh has introduced the Fuzzy Logic as a mathematical tool useful for describing inference engines similar to human reasoning in its capabilities to manipulate knowledge and generate decisions in uncertain and vague environments. Indeed, fuzzy reasoning integrates an alternative way of thinking, which allows the modelling of uncertain systems by using a higher level of abstraction arising from human knowledge, skills and expertise. However, in spite of its original and pioneering aims, currently the fuzzy computation is predominantly aimed at designing and developing intelligent systems belonging to the area of industrial controllers and decision making frameworks. This choice is mainly due to the fact that, as stated by Lotfi Zadeh in his principle of incompatibility, fuzzy frameworks are simpler to design, develop and maintain than their counterparts based on traditional engineering approaches such as the proportional plus integral plus derivative (PID) methodology.

Since its first application in the area of intelligent systems, by E.H. Mamdani, fuzzy logic has been used in numerous application scenarios both in the area of pure engineering and social, economical and political sciences. However, in spite of its massive applicability, the design activity of a fuzzy system may be affected by some strong troubles related to the implementation of a given fuzzy system on heterogeneous hardware architectures, each one characterized by a proper set of electrical/electronic/programming constraints. These difficulties could become more critical in ubiquitous and pervasive scenarios where the different components of a fuzzy system can be distributed and deployed to a collection of interacting and heterogeneous hardware devices.

In the early 2000s, XML technologies were experiencing a kind of rebirth due to their use in defining new protocols for distributed communication and data abstraction such as Web Services. This second life of XML has led researchers to investigate the possibility of integrating these technologies for data abstraction with the theory of fuzzy sets in order to create a software tool capable of supporting scientists and engineers to design fuzzy systems and controllers in a hardware independent way. The name of the first XML-based language for designing fuzzy systems was Fuzzy Markup Language(FML). In detail, FML is a XML-based language whose main aim is to bridge the fuzzy systems design gaps by introducing an abstract and unified approach for describing fuzzy systems. Thanks to its benefits, FML has been sponsored by the IEEE CIS to become the first IEEE standard technology in the area of computational intelligence.

However, due to the rigidity and staticity of XML representation, both FML and other XML-based languages for modelling fuzzy systems are not able to describe innovative topologies of fuzzy systems methodologies that change the configuration of a given system (in terms of rules and variables) during the time in order to always provide the better performance in different scenarios such as, for example, the *Timed Automata-based Fuzzy Systems (TAFCs).*

In order to bridge this design gap, in this paper, we aim at extending FML with a new feature able to dynamically change the XML structure of a fuzzy systems at the same way as the JavaScript approach evolves a HTML webpage. In particular, our approach enhance FML with a scripting language able to modify the *object model* that represents a fuzzy system and, as a consequence, improves the modelling capabilities of this markup language. As shown through a case study on smart grid control, FML scripts enable a full representation of TAFC and they allow systems' designers to develop hardware-independent fuzzy systems yielding better performance than conventional FML modelling.

Giovanni Acampora is with School of Science and Technology, Nottingham Trent University, NG11 8NS, Nottingham, UK (email: giovanni.acampora@ntu.ac.uk);

Marek Reformat is with Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 2V4, Canada (email: reformat@ualberta.ca);

Autilia Vitiello is with the Department of Computer Science, University of Salerno, Fisciano, 84084, Italy (email: avitiello@unisa.it).

This work is related to the standardisation activity for Fuzzy Markup Language sponsored by IEEE Computation Intelligence Society (IEEE SA-1855).

II. FML and other XML-based languages for Fuzzy Systems Design

The term fuzzy-XML could be interpreted in two ways. One of them focuses on expressing fuzzy relations between XML (eXtensible Markup Language) elements. In such a case, XML language itself experiences some level of fuzziness [1]. And the other way - addressed here - focuses on XML-based languages suitable for scripting specifications of fuzzy systems. One of the first works addressing the issue of specifying fuzzy-based systems using XML language has been reported in [2]. It is a simple approach that uses DTD (Document Type Definition) to define fuzzy related terminology and fuzzy data. A fuzzy inference is accomplished via applying a "fact-base" to a "rule-base". A "fact-base" is a file containing facts defined as a combination of real world observations and linguistic variables represented by continuous fuzzy sets. Multiple fuzzy sets, numbers, and intervals are defined in DTD files for reference purposes. A "rule-base", on the other hand, includes a complex structure describing if-then rules and fuzzy operators. Each rule requires identification of compatibility, certainty, aggregation and inference operators, as well as linguistic variables and terms as parts of antecedent(s) and consequent. An accumulation operator is required in the case of multiple rules. Another methodology is presented in [3]. The approach targets reuse and adopts the idea of building complex constructs based on simple ones. The following composition has been adopted to depict a fuzzy system: a component "fuzzy rule system" - which is a special case of a map, layout suitable for description of any system - contains a "fuzzy block" that can be considered a primitive building block mapping inputs to outputs. Further, a "fuzzy block" contains "fuzzy rule bases", "bindings" and "linguistic contexts". Linguistic contexts contain definitions of linguistic variable and crisp variables types. An interesting aspect is an arrangement of contexts and variable types into a tree-like structure, i.e., contexts as directories, and types as files. This allows for an elegant way of defining and storing variables in variety of contexts. Fuzzy rule base is a one type fuzzy rule - absoluteRule - that carries a real valued weight, antecedent and consequent. It is like a template with parameters. These parameters are associated with real objects via bindings. Syntax and semantics are defined using XML Schema and XML Transformation. Disadvantages of the approach are redundant information and complexity of representation of fuzzy systems - it requires a special attention to ensure correctness of bindings and uniqueness of variables. The important advantage is the ability to use the approach to represent any Soft Computing structures as elements of a map/layout describing any possible system. An approach that combines DTD with XML Schema is described in [4]. The authors encapsulate common fuzzy elements so they can be used in any description of a fuzzy system. They propose a "Fuzzy System Model" that is represented as a hierarchy of multiple main components. The top level contains "Input base", "Membership function repository", "Inference Engine", "Operator repository", "Rule base", "Defuzzification", and "Output base". The lower levels are dedicated to fuzzy system data types. Such data types as: linguistic variables, linguistic terms, membership functions, operators and rules are defined and used on numerous occasions in main components. A model specification is a composition of nested (based on Fuzzy System Model) DTD files and XML Schemas referenced from an XML file. If XML Schemas are used, the fuzzy design can be more precise but it requires longer schema descriptions then. Another proposed specification language is XFSML - it is an acronym of compleX/eXtensible Fuzzy Systems Modeling/Markup Language [5]. This language focuses on the structure of a system, not on its functionality. There are no definitions of fuzzy functions - they are defined externally, and provided as libraries. A fuzzy system is an XML file - its name is the system's name - that contains four nodes: "domains", "partitions", "relations", and "modules". A "domain" describes the universe of discourse of variables of the system. "Partitions" describe membership functions on the domains. The "relation" node defines a membership functions over a pair of domains. The node "modules" included multiple subsections describing different types of components that are included in the system: fuzzy decision trees, tables and sets of fuzzy rules, conjunctive fuzzy rules and complex fuzzy rules. The approach seems relatively simple, however it seems that it can lead to redundant information and difficulties in having control over the whole described system. Based on the description if looks a fuzzy system is a single XML file - there is no indication if XML schema is used at all - and any fuzzy operators (membership functions, norms, defuzzification methods) are imported from external libraries. A very interesting approach to represent fuzzy business process models is presented in [6]. The authors propose a machine-readable representation of fuzzy-EPC models in XML. It is called fuzzy-EPML and is based on EPC Markup Language (EPML [7]). Fuzzy-EPC (Even-driven Process Control [8]) models are expressed using ARIS (architecture of integrated information system) language constructs [9], and enriched with fuzzy attributes, functions and relations used for control and regulation of decision situations relevant to companies' processes. Multiple XML Schemas are used to define fuzzy-EPML data types and elements including fuzzy attributes, defuzzification methods, fuzzy operators and variables, as well as fuzzy functions, fuzzy classification systems and classic fuzzy systems. Many of them are extensions of simpler elements or EMPL-basic types and operators. XML Schemas used here prevent redundancy of definitions of system components. This is accomplished via cross-referencing of XML elements. The Fuzzy Markup Language (FML) used here and proposed in [10][11] is a specification language that provides an integrated approach to design centralized and distributed fuzzy controllers in a hardware independent way. It embraces two well-know fuzzy controllers: the Mamdani controller, and the Takagi-Sugeno-Kang controller. FML contains three components: XML documents as descriptions of a fuzzy logic control systems using a markup language, XML

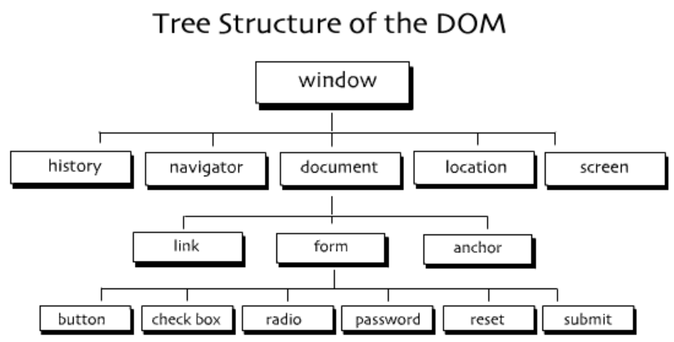


Fig. 1. The Document Object Model (DOM) defined by the HTML representation of webpage

Schema that defines elements of the language and its building blocks, and eXtensible Stylesheet Language Transformations (XSLT) used to convert a fuzzy controller description into a specific programming language. The fundamental idea of FML is to treat a fuzzy controller as a labeled tree. Each node of this tree is labeled with an XML tag and represents a single part/aspect/concept of a controller. The XML tags are equipped with attributes that allow customize and instantiate a model. The “Knowledge Base” subtree contains tags identifying used fuzzy variables, fuzzy terms, and shapes of fuzzy membership functions. The “Rule Base” subtree is used to define a rule base set. All elements of rules are defined using nodes and their respective XML tags. The FML context-free grammar is modeled via XML Schema.

III. FMLScript: a Scripting Language for FML

In this section, FMLScript, the scripting language for enhancing FML capabilities, is introduced. Our idea is based on the same motivations that led to the emergence of JavaScript (or similar technologies) in the context of the design of HTML webpages. Initially, HTML was introduced as a markup language for describing the presentation and visualisation of distributed and digital contents through a computer program as a web browser. However, the capabilities of HTML were too limited to enable a full interaction among users and webpages. For this reason, JavaScript has been designed to improve HTML features and achieve a more dynamic interaction in web navigation. This improvement was mainly due to the capability of JavaScript to dynamically modify the *Document Object Model (DOM)*, a hierarchical and object-oriented representation of a HTML webpage. As shown in Fig. 1, the DOM contains all of the information related to a given HTML page such as links, forms, text boxes and so on. JavaScript uses an object-oriented syntax to access this information and modify it appropriately.

As both HTML and FML grammars have been designed by using markup language technologies, at the same way as a HTML description induces an object model that JavaScript programs can access to change the configuration of a webpage, a FML-based description of a fuzzy system generates

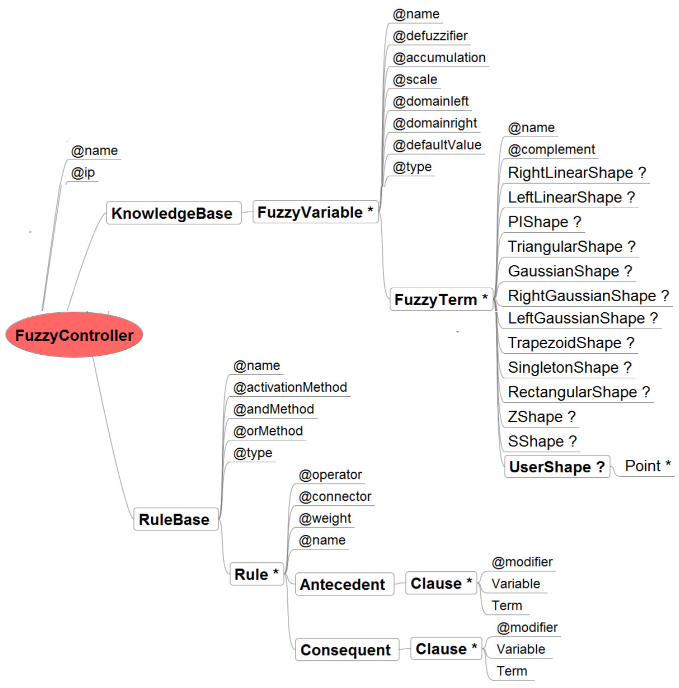


Fig. 2. The Fuzzy Object Model (FOM)

a hierarchical structure (object model) composed of all the components of the system: variables, fuzzy sets, clauses, rules and so on. We show this structure, called *Fuzzy Object Model (FOM)*, in Fig. 2.

The existence of the FOM in the FML scenario naturally leads to the definition of a new object-oriented language, the FMLScript. This language allows to modify the collection of FOM objects and develop FML-based models of fuzzy systems that change their configuration over time. Hence, FMLScript comes with a built-in library of objects related to the components of a fuzzy system. Each object has properties that describe it and that can be accessed through a collection of setter and getter methods. Table I reports a set of correspondences among FML tags, FOM objects and a subset of FMLScript methods acting on the related FOM objects. FMLScript instructions are inserted in FML codes at the same way as JavaScript instructions are embedded in HTML codes, that is, by means of two tags: <FMLScript> and </FMLScript>. These tags usually go between the </Rulebase> tag and the </FuzzyController> tag, although can be placed anywhere in the FML code. As other object-oriented languages, FMLScript uses variables to store information. The declaration of a variable contains the data type such as int, float and so on, followed by the variable name. FMLScript uses the naming conventions of several languages: variables must start with either a letter or the underscore character, and, after the first character, they can have numbers. Besides, the variable names are case-sensitive.

The Listing 1 shows the described basics of FMLScript, i.e., where FMLScript is placed in FML code, how to comment, how to declare a variable and how to change the

TABLE I

A SUBSET OF CORRESPONDENCES AMONG FML TAGS, FOM OBJECTS AND FMLSCRIPT METHODS.

FML Tag	FML Tag Attribute	FOM objects	FMLScript Methods
<FuzzyController>	name, ip	FuzzyController	setName(*nameController*), setIp(*ip*), ...
<KnowledgeBase>	ip	KnowledgeBase	setIp(*ip*)
<FuzzyVariable>	name, scale, domainLeft, domainRight, type, defuzzifier, accumulation, defaultValue	FuzzyVariable	setName(*nameVariable*), setDefuzzifier(*defuzzifier*), setDomainRight(*upperValue*),...
<Rulebase>	name, type, activationMethod, andMethod, orMethod	RuleBase	setName(*nameRulebase*), setOrMethod(*orMethod*), ...
<Rule>	name, connector, operator, weight	Rule	setName(*nameRulebase*), setOperator(*method*), ...

shape of a triangular fuzzy set, named "rancid", belonging to the fuzzy variable named "food".

```
<FuzzyController>
<KnowledgeBase>
   <FuzzyVariable name=''food''
      domainleft=''0.0'' domainright=''10.0''
      scale='''' type=''input''>
         <FuzzyTerm name=''delicious''
            complement=''false''>
            <LeftLinearShape
               param1=''5.5''
               param2=''10.0''/>
         </FuzzyTerm>
         <FuzzyTerm name=''rancid''
            complement=''false''>
            <TriangularShape
               param1=''0.0''
               param2=''2.0''
               param3=''5.5''/>
         </FuzzyTerm>
   </FuzzyVariable>
...
</KnowledgeBase>
<RuleBase>
...
</RuleBase>

<FMLScript>
   /*
      Access to FOM objects
      to modify the FML Fuzzy System
   */

   FuzzyVariable v;
   FuzzyTerm t;

   v = KnowlegdeBase.getVariable(''food'');
   t = v.getFuzzyTerm(''rancid'');
   t.getTriagularShape().setParam3(6.75);
</FMLScript>
</FuzzyController>
```

Listing 1

AN EXAMPLE OF FMLSCRIPT

FMLScript provides the abilities to modify the logic flow of the program and to perform loops such as if-then and if-then-else, while, for, break and continue. In order to call over and over bits of FMLScript without having to rewrite them every time, FMLScript allows to define functions as below.

```
typeReturn funtionName(parameterList){
 statement;
 return(ret_val);
}
```

Functions' naming rules are the same as those for variables. After the function name comes a list of parameters and, closed in curly braces, the body of the function containing a set of statements you want to run when the function is called. On reaching a return statement, control of the program returns to the calling function. The bracketed value ret_val is the value which is returned from the function whose type is defined by *typeReturn*.

However, modifications of the FOM by means of getter and setter methods provided by FMLScript are not enough to allow FML to improve its modeling capabilities. Indeed, the FOM provides FML with a mechanism to select "what" to change in a fuzzy system, but no information about "when" to change "what" is provided. As a consequence, the definition of a collection of additional "time-related" classes in the FMLScript language is necessary to define the temporal evolution of the FOM and the corresponding fuzzy system. In particular, new classes such as *Timer* and *TimeConstraint* can enable FMLScript to create a collection of additional objects that, opportunely used, allow FML to establish when to change the FOM structure and represent fuzzy systems in a more dynamic way. The Timer class is used to define new variables that count the number of milliseconds elapsed from a given time. In particular, let t be a FMLScript object defined by using the Timer class, then the method named *start()* is invoked on t (formally *t.start()*) to activate the timer t; the method *reset()* is invoked on t (formally *t.reset()*) to set the counter of t to zero; the method *stop()* (formally *t.stop()*) is invoked on t to stop it and the method *restart()* (formally *t.restart()*) is invoked on t to resume the time progress for it. Variables defined as Timer are necessary to create so-called "temporal constraints", i.e.

conditional expressions used by FMLScript to check when the FOM structure should be changed. FMLScript enables the definition of these temporal constraint variables by using the timeConstraint class. In particular, a timeConstraint object is a binary conditional expression based on $t \circ m$, where t is Timer object, m is a real number and $\circ \in \{==, <, >, <=, >=\}$. Together with the Timer and TimeConstraints classes, FMLScript provides the TimeConstraints objects with a powerful method, named *onTime*, used to trigger the appropriate script when a temporal constraint on a given timer object is satisfied. In general, the triggered script is used to access to the FOM and modify some components of the FML fuzzy system. The syntax of the onTime method is:

$$c.\text{onTime}(UserFunctionName())$$

where c is a timeConstraints object defined on a Timer t and *UserFunctionName* is the name of a user defined script.

Listing 2 shows how FMLScript uses Timer and TimeConstraints objects to change the shape of a fuzzy variable exactly after 20 seconds from the moment in which the FML system started its execution.

```
<FuzzyController>
<KnowledgeBase>
   ...
</KnowledgeBase>
<RuleBase>
   ...
</RuleBase>

<FMLScript>

   /*
      Access to FOM objects
      to modify the FML Fuzzy System
   */

   FuzzyVariable v;
   FuzzyTerm t;
   Timer t = new Timer(0);
   TimeConstraint c;
   c = new TimeConstraint(``t == 2000'');

   c.onTime(``changeFuzzyVariableFunction()'');

   t.start();

   void changeFuzzyVariableFunction() {
      v = KnowlegdeBase.getVariable(``food'');
      t = v.getFuzzyTerm(``rancid'');
      t.getTriagularShape().setParam3(6.75);
   }
</FMLScript>
</FuzzyController>
```

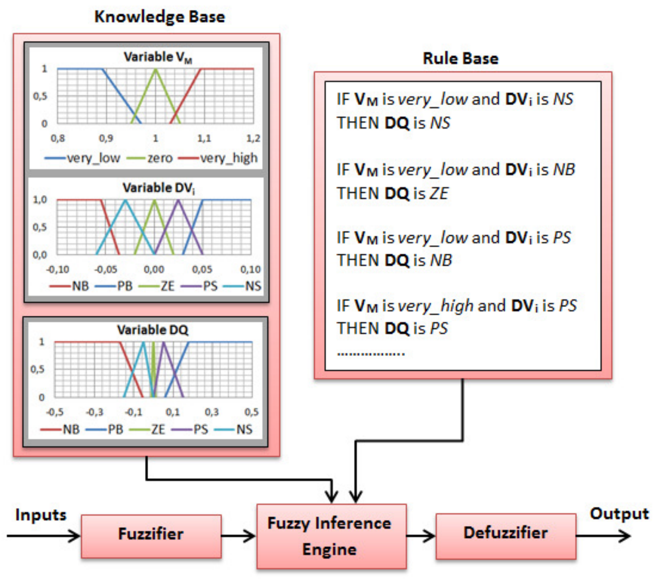
Listing 2
An example of FMLScript with Timer and TimeConstraint Objects.

IV. Case Study: Smart Grid Control

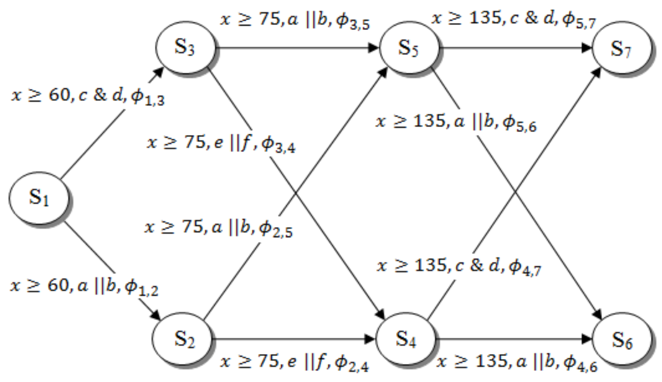
This section illustrates the benefits provided by the introduction of FMLScript. In order to achieve this aim, we refer to a case study in the smart grid domain. We show how FMLScript is able to model new emergent topologies of fuzzy systems such as TAFCs (Timed Automata based Fuzzy Controllers [12]) which allow to control voltage regulation in a smart grid better than conventional ones.

Smart grids are active, flexible and self-healing energy networks which marries information technology with the current electrical infrastructure in order to empower consumers to manage their energy usage and save money without compromising their lifestyle. In distributed and heterogeneous environments as the smart grids, it is desirable to have a software tool capable of supporting scientists and engineers to design fuzzy systems and controllers in a hardware independent way. Therefore, FML is highly suitable to be applied in smart grid scenarios. However, because of the rigidity and staticity of XML representation, FML is not completely able to manage the intrinsic time varying phenomena affecting the smart grid operation. FMLScript overcomes the FML limitation thanks to the capability to model new dynamic fuzzy systems such as TAFCs. Hereafter, after a brief description of TAFCs, we describe how FMLScript can model a TAFC for voltage regulation in a smart grid.

TAFCs are evolvable fuzzy systems highly suitable for modeling dynamic systems thanks to the capability of modifying models' behavior over time. In detail, TAFCs' behaviour is modeled by means of an opportune fuzzy controller which represents the so-called control configuration of the TAFC. According to time related events, the current control configuration of a TAFC can be changed by applying some so-called transformation operators. These operators perform changes on fuzzy controllers such as adding or deleting a variable, adding, removing or changing rules in the rule base, adding, deleting or changing a term to a variable and so on. In this vision, a TAFC lives a sequence of discrete temporal periods, named control eras, each one characterized by a specific control configuration. In order to manage the switching between control eras, TAFCs exploit a modified version of Timed automata [13]. Briefly, a timed automaton is a standard finite-state automaton extended with a finite collection of real-valued *clocks* providing a straightforward way to represent time related events. In particular, timed automaton's transitions are labeled with so-called *guards* representing conditions on clocks and *symbols* on alphabet Σ representing events. A transition may be taken only if the current values of the clocks satisfy the guard and the specified event is occurred. By using a timed automaton, TAFCs are able to manage the control configurations by associating each of them with a state in the timed automaton, whereas, symbols and clocks characterizing transitions determine respectively how and when to execute the switching among successive control eras. Futher, TAFCs exploit an extended version of timed automata since transition edges are characterized also by a set of the aforementioned transformation operators aimed at changing the current control configuration of the modeled system in the next one. In conclusion, it is possible to define a TAFC as a pair (F^0, T_C) where F^0 is the initial control configuration, represented by a fuzzy controller, modeling the control behaviour of system during first phase of its existence, and T_C is an extended timed automaton describing the dynamic evolution of the system.


Knowledge Base
Variable V_M
very_low
zero
very_high
Variable DV_i
NB
PB
ZE
PS
NS
Variable DQ
Rule Base
IF V_M is very_low and DV_i is NS THEN DQ is NS
IF V_M is very_low and DV_i is NB THEN DQ is ZE
IF V_M is very_low and DV_i is PS THEN DQ is NB
IF V_M is very_high and DV_i is PS THEN DQ is PS
....
Inputs
Fuzzifier
Fuzzy Inference Engine
Defuzzifier
Output

(a) Fuzzy Controller C



(b) Timed automaton T_A

Fig. 3. The TAFC T regulating the reactive power flows injected by a DG unit in a smart grid.

In [14], TAFCs are used to regulate power flow injected by the Distribution Generation (DG) units embedded in a smart grid to enable the integration of renewable energy resources (such as wind and solar). The designed TAFC $T = (C, T_A)$ is shown in Fig. 3. In detail, the fuzzy controller C is composed of two input variables, the mean grid voltage magnitude (V_M) and the difference between the mean grid voltage magnitude and the local bus voltage magnitude (DV_i) and an output one, the reactive power flow (DQ). The rule base is composed of 11 rules defined by a domain expert. The designed fuzzy inference engine computes the conventional Mamdani's inference method [15], whereas, the defuzzifier is the mean of maxima method [16]. As for the timed automaton T_A, it is composed by seven states $S_1, S_2, \ldots, S_7$. Each state stores a control configuration able to regulate the reactive power flow during the corresponding control era. The transitions are labeled with guards on clock x (which keeps track of elapsing time in minutes) and the symbols a, b, c, d, e, f which represent events related to the voltage magnitude values as follows:

- $a : M < 0.99$
- $b : M > 1.01$
- $c : M > 0.99$
- $d : M < 1.01$
- $e : M < 0.97$
- $f : M > 1.03$

For sake of simplicity, in the Fig. 3, the description of the transformation operators is replaced by a function $\phi_{i,j}$ capable of defining the set of operators necessary for transforming the control configuration related to the state S_i in the control configuration related to the state S_j. For example, the function $\phi_{1,2}$ moves the control configuration in the state S_1 to the control configuration in the state S_2 by changing some rules and the names and parameters of two fuzzy terms of the fuzzy variable V_M.

In spite of FML advantages, it is not possible to model a dynamic fuzzy system such as a TAFC. Indeed, FML succeeds to model the initial control configuration of a TAFC but it cannot do the same with the dynamic features of TAFCs such as changing and switching concepts, the clock idea, and so on. FMLScript overcomes the FML limitation by succeeding to model TAFCs in a hardware independent way as shown below. As said, the fuzzy controller C composing the TAFC T for voltage regulation can be modeled through the static capabilities of FML. As an example, listing 3 models a portion of knowledge base of the controller C, i.e., the input variable V_M, and listing 4 defines the first rule of the rulebase.

```
......
<KnowledgeBase>
    <FuzzyVariable name=``V_M'' domainleft=``0.8''
                   domainright=``1.2'' scale=``''
                   type=``input''>
        <FuzzyTerm name=``very_low''
                   complement=``false''>
            <LeftLinearShape Param1=``0.89''
                             Param2=``0.98''/>
        </FuzzyTerm>
        <FuzzyTerm name=``zero''
                   complement=``false''>
            <TriangularShape Param1=``0.95''
                             Param2=``1.0''
                             Param3=``1.05''/>
        </FuzzyTerm>
        <FuzzyTerm name=``very_high''
                   complement=``false''>
            <RightLinearShape Param1=``1.03''
                              Param2=``1.09''/>
        </FuzzyTerm>
    </FuzzyVariable>
    ........
<KnowledgeBase>
.....
```

Listing 3
FML CODE TO MODEL A PORTION OF THE KNOWLEDGE BASE OF THE CONTROLLER C

```
.......
<RuleBase name=``Rulebase1''
          activationMethod=``MIN'' andMethod=``MIN''
              orMethod=``MAX'' type=``mamdani''>
<Rule name=``reg1'' connector=``or''
       operator=``MAX'' weight=``1.0''>
             <Antecedent>
                 <Clause>
                     <Variable>$V_M$</Variable>
                     <Term>very_high </Term>
                 </Clause>
                 <Clause>
                     <Variable>$DV_i$</Variable>
                     <Term>PS</Term>
                 </Clause>
             </Antecedent>
             <Consequent>
                 <Clause>
                     <Variable>$DQ$</Variable>
                     <Term>PB</Term>
                 </Clause>
             </Consequent>
         </Rule>
      ........
</RuleBase>
........
```

Listing 4

FML CODE TO MODEL THE FIRST RULE OF THE RULE BASE OF THE CONTROLLER C

As for the dynamic features of the TAFC T, they can be modeled through FMLScript as shown in listing 5. In detail, the automaton states can be represented as a FMLScript variable, in our case, the variable s. The clock x, instead, can be modeled by using a Timer object, in our case, the Timer t. As for the guards and the symbols on transitions, they can be defined by using, respectively, TimeConstraint objects ($c1$, $c2$, and so on) and FMLScript variables (M). The management of the transitions having the same guard can be done through a FMLScript function. In our case, the function $transitionOnC1()$ is related to the first two transitions having the guard $x >= 60$, instead, the function $transitionOnC2()$ is related to the next four transitions having the guard $x >= 75$. Finally, the transformation operators can be directly implemented through the exploitation of the predefined FMLScript methods acting on FOM. As an example, listing 6 shows the definition of the aforementioned function $\phi_{1,2}$ of the timed automaton T_A by using FMLScript code.

The introduction of FMLScript allows to improve voltage regulation in the considered smart grid scenario as shown in Fig. 4. Indeed, thanks to the definition of the TAFC T by using FMLScript, both reactive power costs and voltage magnitude are improved with respect to a FML-based fuzzy controller F.

V. CONCLUSIONS

In the recent years, new topologies of fuzzy systems capable of modelling real scenarios in a more detailed and accurate way have been emerging. These new fuzzy system modelling approaches, such as Timed Automata based Fuzzy Controllers, are more realistic thanks to their ability to

```
<FMLScript>
int s=1; //the current state
Timer t=new Timer(0); //timer modeling clock x
float M; //the symbols

/*guard on transitions from S1 to S2 and
from S1 to S3 in milliseconds*/
TimeConstraint c1;
c1 = new TimeConstraint(``t >= 3600000'');
c1.onTime(``transitionOnC1()'') ;

/*guard on transitions from S2 to S4,from S2 to S5,
from S3 to S4 and from S3 to S5 in milliseconds*/
TimeConstraint c2;
c2 = new TimeConstraint(``t >= 4500000'');
.....
t.start(); // timer is started

void transitionOnC1() {
t.stop();
if(s==1)
 if(M<0.99 || M>1.01){
   //moves the control configuration from S1 to S2
   phi12();
   s=2;
   c2.onTime(``transitionOnC2()'') ;
 }
 else if(M>0.99 && M<1.01){
   //moves the control configuration from S1 to S3
   phi13();
   s=3;
   c2.onTime(``transitionOnC2()'') ;
 }
  t.restart();
}

void transitionOnC2() {
t.stop();
if(s==2)
 if(M<0.97 || M>1.03){
   //moves the control configuration from S2 to S4
   phi24();
   s=4;
   .....
 }
 else if(M<0.99 || M>1.01){
   //moves the control configuration from S2 to S5
   phi25();
   s=5;
   .....
 }
else
 if(s==3)
   if(M<0.97 || M>1.03){
    //moves the control configuration from S3 to S4
    phi34();
    s=4;
    .....
   }
   else if(M<0.99 || M>1.01){
    //moves the control configuration from S3 to S5
    phi35();
    s=5;
    .....
   }
t.restart();
}
......
<FMLScript>
```

Listing 5

FMLSCRIPT CODE FOR MODELING TAFC T

```
<FMLScript>
void phi12(){

FuzzyVariable v;
FuzzyTerm t ;
v = KnowlegdeBase.getVariable(``$V_M$'');
t = v.getFuzzyTerm(``very_low'');
t.setName(``low'');
t.getLeftLinearShape().setParam1(0.95);
t.getLeftLinearShape().setParam2(0.99);

t = v.getFuzzyTerm(``very_high'');
t.setName(``high'');
t.getLeftLinearShape().setParam1(1.02);
t.getLeftLinearShape().setParam2(1.05);

Rule r;
Clauses c;
r=RuleBase.getRule(``reg1'');
c=r.getConsequent().getClauses();

for(int i=0; i<c.size(); i++)
if(c.getClause(i).getVariable().equals(``DQ'')){
    c.getClause(i).setTerm(``PS'');
    break;
    }
.....
}
<FMLScript>
```

Listing 6

FMLSCRIPT CODE FOR MODELING THE FUNCTION $\phi_{1,2}$ OF THE TIMED AUTOMATON T_A

express not only static but particularly dynamic and temporal aspects of systems. However, XML-based modelling languages, including FML, are not able to model these new topologies of fuzzy systems. They lack mechanisms and constructs to represent time managed events and modifications. In this paper we introduce and describe FMLScript. It is a script-based language that enhances FML modeling capabilities to handle dynamic features of new and innovative fuzzy structures. FMLScript possesses multiple features that improve modeling of real-world industrial systems. The application of both FMLScirpt and FML to model a smart grid controller is presented and described in the paper.

REFERENCES

[1] Z.M. Ma and L. Yan, "Fuzzy XML data modeling with the UML and relational data models," *Data & Knowledge Engineering*, vol. 63, pp. 972-996, 2007.

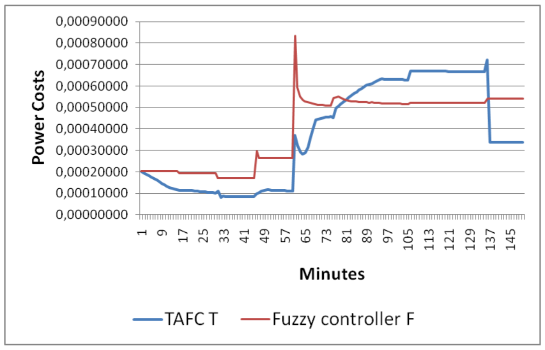
[2] K. Turowski and U. Weng, "Representing and processing fuzzy information - an XML-based approach," *Knowledge-based Systems*, vol. 15, pp. 67-75, 2002.

[3] A.R. de Soto, C.A. Capdevila and E.C. Fernndez, "Fuzzy Systems and Neural Networks XML Schemas for Soft Computing," *Mathware and Soft Computing*, vol. 10, no. 2-3, pp. 43-56, 2003.

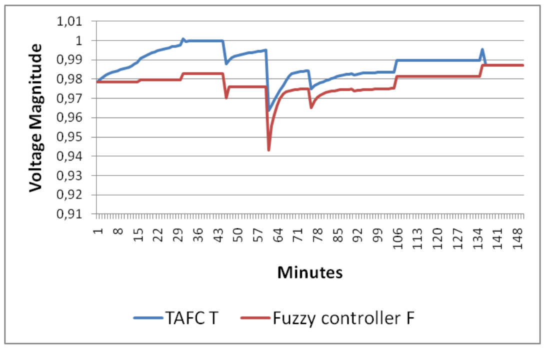
[4] C. Tseng, W. Khamisy and T. Vu, "Universal Fuzzy System Representation with XML," *Computer Standards and Interfaces,* 28, 218-230 (2005)

[5] Moreno-Velo, F.J., Barriga, A., Snchez-Solano, S., and Baturone, I.: "XFSML: An XML-based Modeling Language for Fuzzy Systems," WCCI 2012 IEEE World Congress on Computational Intelligence, Brisbane, Australia, 2012.

[6] O. Thomas and T. Dollmann, "Fuzzy-EPC Markup Language: XML Based Interchange format for Fuzzy Process Model," in Ma., Z., and Yan., L. (Eds.): Soft Computing in XML Data Management, STUDFUZZ 255, pp. 227-257, 2010.



(a)



(b)

Fig. 4. The comparison between a FMLScript-based definition of the TAFC T and the FML-based modeling of a conventional fuzzy controller F in terms of the reactive power costs values in (a) and the voltage magnitude values in (b)

[7] J. Mendling and M. Nttgens, "EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC)," ISeB, vol. 4, no. 3, pp. 245-263, 2006.

[8] E. Kindler, "On the semantics of EPCs: Resolving the vicious circle," *Data & Knowledge Engineering*, vol. 56, no. 1, pp. 23-40, 2006.

[9] A.-W. Scheer, "ARIS - business process frameworks," 2., completely rev. and enl. ed. Springer, Berlin, 1998.

[10] G. Acampora and V. Loia, "Fuzzy control interoperability and scalability for adaptive domotic framework," *IEEE Transactions on Industrial Informatics*, vol. 2, pp. 97-111, 2005.

[11] G. Acampora and V. Loia, "A proposal of ubiquitous fuzzy computing for Ambient Intelligence," *Information Sciences*, vol. 178, no. 3, pp. 631-646, 2008.

[12] G. Acampora, V. Loia and A. Vitiello, "Hybridizing fuzzy control and timed automata for modeling variable structure fuzzy systems," 2010 IEEE International Conference on Fuzzy Systems, pp. 1-8, 2010.

[13] R. Alur, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183235, 1994.

[14] G. Acampora, V. Loia and A. Vitiello, "Exploiting Timed Automata based Fuzzy Controllers for voltage regulation in Smart Grids," 2011 IEEE International Conference on Fuzzy Systems, pp. .223-230, 2011.

[15] E.H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller,“ *International Journal of Man-Machine Studies,* vol. 7, no. 1, pp. 1-13, 1975.

[16] C.C. Lee,"Fuzzy Logic in Control System: Fuzzy Logic Controller - Part II," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 2, pp. 404-435, 1990.