Fuzzy Discrete-Event Systems Modeling and Simulation with Fuzzy Control Language and DEVS Formalism

Jean-François Santucci and Laurent Capocchi

SPE UMR CNRS 6134 Laboratory
University of Corsica
Email: {santucci, capocchi}@univ-corse.fr

***Abstract*—There is an increasing use of fuzzy data in the field of discrete-event system modeling. This paper deals with an approach based on the use of Fuzzy Control Language (FCL) allowing to facilitate the modeling and simulation of Discrete EVent Systems (DEVS) involving uncertainty. The main contribution of this paper is to integrate the Fuzzy Control Logic three basic steps (fuzzification, defuzzification, fuzzy rules) into DEVS models in a generic way using a FCL-based implementation. The implementation has been performed with the DEVSimPy environment through a classical example. The DEVSimPy modeling of the example highlights how the DEVS modeler can realize the three basic steps inside the DEVS transition functions.**

Keywords–DEVS; Fuzzy systems; Modeling; Simulation; Discrete-event system.

I. INTRODUCTION

In the recent years, a set of work has concerned the introduction of fuzzy notions into DEVS concepts in order to propose incorporation of uncertainty into models [1]. Approaches [2]–[4] developed around the DEVS formalism have been proposed in order to apply fuzzy set theory [5] to the sets and functions defined on crips sets in DEVS formalism. Fuzzy modeling and simulation can be useful in the study of different kinds of systems for example in the case of the modeling of systems for which we have few observations of how and where the human being acts as a sensor or expert. The already developed approaches integrating fuzzy sets into DEVS bring nice formalisms allowing to deal with fuzzy events, fuzzy transition functions and even fuzzy time advance function. However, the modeling and simulation of concrete applications involving fuzzy notions based on human expertise are not easily implemented using the previous approaches. In order to set up this, the fuzzy logic provides methods closed to expert human labor. Fuzzy logic has already proved the ability to reason about imprecise or subjective data in wide range of fields, from finance to industrial control, through consumer electronics and weapons systems. Fuzzy logic is closed to reasoning and human language, as it is known to use vague concepts (such as “hot ”, “weak”, “strong enough”, etc.).

The fuzzy approach allows the computer modeling of vague notions (hotter, colder, etc.), commonly used by consumers or by plant operators. Fuzzy technology has the advantage of using explicit knowledge (the system works by applying rules) by highlighting three distinct steps: (i) initially the fuzzification process is studied in order to consider the inclusion of linguistic information. (ii) the second part is devoted to the identification of fuzzy rules mainly when observations on the behavior of the system are imprecise and uncertain. Fuzzy rules (such as “if X ... then Y”) allow to express the knowledge concerning the problem to address; (iii) the last part concerns the defuzzification which is to convert the fuzzy domain to the digital domain, with a conversion preferences. The method of defuzzification is used to exploit the information encoded in the output fuzzy sets corresponding to expert knowledge on the output variable of the model.

The integration of fuzzy logic into the DEVS formalism involves three steps: (i) the definition of concepts allowing to deal with fuzzy logic in the framework of DEVS; (ii) the selection of a fuzzy logic programming language in order to accomplish the integration of DEVS and fuzzy logic; (iii) the implementation of the concepts into the DEVSimPy framework [6][7]. The first step allows to define the concepts allowing to perform with the DEVS formalism the three required steps involved in fuzzy logic control: fuzzication, the fuzzy rules definition and firing and finally defuzzification. Among a set of available fuzzy theory programming languages (FCL - Fuzzy Control Language [8], FPL-Fuzzy Programming Language [9], FTL - Fuzzy Technology Language [10], FSTDS-fuzzy STDS [11], LPL - Linguistic oriented Programming Language [10], FSML-Fuzzy System Modeling Language [12], etc.), we choose the FCL one for different reasons: it is a standardized Domain Specific Language created for fuzzy control applications; it allows to define blocks of fuzzy functions with fuzzy inputs and outputs; it allows also to simplify the writing of variables and linguistic values, as well as the writing of logical rules using syntax such as “IF, THEN”.

The main contribution of this paper is not concept-oriented as in Kwon et al. [4], but concerns the implementation of a Fuzzy Control Logic library (including fuzzification, defuzzification, fuzzy rules) into the DEVSimPy framework in a generic way using a FCL-based implementation. The DEVSimPy framework is a DEVS modeling and simulation tool written in Python language. Python is one of the programming languages that provides a framework for defining fuzzy inference systems through PyFuzzy package [13]. In addition, Python also allows the definition of these fuzzy inference systems from the FCL language. In order to facilitate for a DEVS modeler the use of fuzzy logic when performing simulation of a given application we choose to integrate FCL systems into the DEVSimPy framework. FCL facilitates the DEVS modeler to transform the imprecision in users request into defuzziyfied values which can be used easily in DEVS simulations. The rationale behind the coupling of FCL system with DEVS is that an “expert” human operator can control a process without understanding the details of its underlying dynamics. The effective and real control strategies that the

 ISBN: 978-1-61208-371-1

expert learns through experience can often be expressed as a set of condition-action, "IF, THEN" rules, that describe the process state and recommend actions using linguistic, fuzzy terms instead of classical, crisp rules.

The rest of the paper is organized as follows: Section 2 presents the background of the work including the DEVS formalism and the FCL language. Section 3 gives the concepts that have been defined in order to integration fuzzy logic into the DEVS formalism. The implementation of the previously defined concepts is given is Section 4. A pedagogical example allows to illustrate the feasibility of the approach inside the DEVSimPy environment. Finally, in the last part, we conclude the work and present the future work around fuzzy inductive modeling.

II. BACKGROUND

A. The DEVS Formalism

Since the seventies, some formal works have been directed in order to develop the theoretical basements for the modeling and simulation of dynamical discrete event systems [14]. DEVS [15] has been introduced as an abstract formalism for the modeling of discrete event systems, and allows a complete independence from the simulator using the notion of abstract simulator.

DEVS defines two kinds of models: *atomic models* and *coupled models*. An atomic model is a basic model with specifications for the dynamics of the model. It describes the behavior of a component, which is indivisible, in a timed state transition level. Coupled models tell how to couple several component models together to form a new model. This kind of model can be employed as a component in a larger coupled model, thus giving rise to the construction of complex models in a hierarchical fashion. As in general systems theory, a DEVS model contains a set of states and transition functions that are triggered by the simulator.

A DEVS atomic model AM with the behavior is represented by the following structure:

$$AM =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a > \quad (1)$$

where:

- $X : \{(p, v)|(p \in inputports,\ v \in X_p^h)\}$ is the set of input ports and values,
- $Y : \{(p, v)|(p \in outputports,\ v \in Y_p^h)\}$ is the set of output ports and values,
- S: is the set of states,
- $\delta_{int} : S \to S$ is the internal transition function that will move the system to the next state after the time returned by the time advance function,
- $\delta_{ext} : Q \times X \to S$ is the external transition function that will schedule the states changes in reaction to an external input event,
- $\lambda : S \to Y$ is the output function that will generate external events just before the internal transition takes places,
- $t_a : S \to R_\infty^+$ is the time advance function, that will give the life time of the current state.

The dynamic interpretation is the following:

- $Q = \{(s, e)|s \in S^h, 0 < e < t_a(s)\}$ is the total state set,
- e is the elapsed time since last transition, and s the partial set of states for the duration of $t_a(s)$ if no external event occur,
- δ_{int} : the model being in a state s at t_i , it will go into s' , $s' = \delta_{int}(s)$, if no external events occurs before $t_i + t_a(s)$,
- δ_{ext} : when an external event occurs, the model being in the state s since the elapsed time e goes in s', The next state depends on the elapsed time in the present state. At every state change, e is reset to 0.
- λ : the output function is executed before an internal transition, before emitting an output event the model remains in a transient state.
- A state with an infinite life time is a passive state *(steady state),* else, it is an active state *(transient state)*. If the state s is passive, the model can evolve only with an input event occurrence.

The DEVS abstract simulator is derived directly from the model. A simulator is associated with each atomic model and a coordinator is associated with each coupled model. In this approach, simulators allows to control the behavior of each model, and coordinators allows the global synchronization between each of them.

B. Fuzzy Control Language

A Fuzzy Control Language (FCL) system allows to define the process for the mapping from a given input to an output using fuzzy logic. It is based on Mamdani [16] method for fuzzy inference which is the most suitable for capturing expert knowledge, as its rules allow us to describe the expertise in more intuitive and human-like manner. The fuzzy inference process comprises the following steps: fuzzify the input, evaluate the fuzzy rules, aggregate the outputs to reach the final decision, and defuzzify the output to obtain a crisp value.

A FCL system can be formalized as follows:

- V1: a set of input variables.
- V2: a set of output variables.
- Fuzzification part:
  <term (or set) name> := <points that make up the term> for each input variable.
- Rules description part:
  <operator>:<algorithm>;
  ACCUM:<accumulation method>;
  RULE <rule number>: IF <condition> THEN <conclusion>;
- Defuzzifiation part:
  METHOD:<defuzzification method>;

Each one of these parts has to be defined when considering a FCL system.

 ISBN: 978-1-61208-371-1

Fuzzyfication part: The values of V1 have to be converted into degrees of membership for the membership functions defined on linguistic variables. These linguistic variables shall be described by one or more linguistic terms. The linguistic terms are introduced and described by membership functions in order to fuzzify the variable.

$$membership\_function ::= (pointi), (pointj), \ldots \quad (2)$$

A membership function is defined by a table of points under the format presented in 2.

Rules description part: Two definitions are required in order to defined the rules: (i) The Fuzzy operators are used inside the rules; (ii) The accumulation method.

Fuzzy Operators: To fulfill de Morgans Law, the algorithms for operators AND and OR shall be used pair-wise e.g.:

- MAX shall be used for OR if MIN is used for AND.
- ASUM for Or if PROD for AND.
- BSUM for OR if BDIF for AND.

Accumulation method: Three methods are available:

- Maximum: MAX.
- Bounded Sum: BSUM.
- Normalized sum: NSUM.

The inputs of a set of rules are linguistic variables with a set of linguistic terms. Each term has a degree of membership assigned to it. It is possible to combine several sub-conditions and input variables in one rule. The general format is the following:

$$subcondition := linguistic\_variable\ IS\ [NOT] \quad linguistic\_term \quad (3)$$

The conclusion can be split into several subconclusions and output variables:

$$subconclusion := linguistic\_variable\ IS \quad linguistic\_term \quad (4)$$

Optionally it is possible to give each subconclusion a weighting factor which is a real with a value between 0.0 and 1.0. This shall be done by the keyword WITH followed by the weighting factor. The weighting factor shall reduce the membership degree (membership function) of the subconclusion by multiplication of the result in the subconclusion with the weighting factor. The generic format of a rule is:

$$IF\ condition\ THEN\ subconclusion \quad [WITH\ weighting\_factor]\ subconclusion \quad (5)$$

Defuzzification part: A linguistic variable for an output variable has to be converted into a value. The variable which is used for the defuzzification corresponds to an output variable.

A defuzzification method has to be chosen; The following defuzzification methods are possible:

- COG Centre of Gravity.
- COGS Centre of Gravity for Singletons.
- COA Centre of Area.
- LM Left Most Maximum.
- RM Right Most Maximum.

The following Section introduces the proposed approach based on the previous concepts.

III. Proposed Approach

The section introduces the concepts that have been defined in order to integrate FCL into DEVS models. In order to succeed in this integration, the following requirements have to be fulfill:

- Ability to define of membership functions (see Equ. 2) for creating fuzzy inference systems (V1, V2).
- Ability to support the AND, OR, and NOT logic operators in user-defined rules.
- Ability to embed a fuzzy inference system in a DEVS model.
- Ability to generate executable fuzzy inference engines.

The proposed solution to integrate FLC into DEVS models is to offer the possibility to define aggregate fuzzy logic to each one of the functions involved in the DEVS formalism (δ_{ext}, δ_{int}, λ and t_a). For that X, Y and S will contain variables which are going to be fuzzyfied. Then inferences rules (see Equ. 3,4,5) will be written inside of DEVS function and fired as soon as the corresponding function is activated. Finally, a defuzzyfication mis performed in order to obtain the result of the respective function (new state for δ_{ext} and δ_{int}, an output for λ and a real representing the σ for t_a). During the simulation, the execution of one of the traditional DEVS functions consists in firing the fuzzy engine associated with each DEVS function.

A DEVS atomic model AM integrating a fuzzy behavior using the FCL language and inference engine is represented by specifying the following functions:

- $\delta_{int} : S \to S$ is defined as a FCL system involving the three previously presented steps (fuzzyfication, rules definition, defuzzyfication). It is defined as a FCL system $\delta_{int}^{FCL}(V1 : S, V2 : S)$.
- $\delta_{ext} : Q \times X \to S$ is defined as a FCL system $\delta_{ext}^{FCL}(V1 : Q \times X, V2 : S)$
- $\lambda : S \to Y$ is defined as a FCL system $\lambda^{FCL}(V1 : S, V2 : Y)$
- $t_a : S \to R_{\infty}^{+}$ is defined as a FCL system $t_a^{FCL} : (V1 : S, V2 : real)$.

 ISBN: 978-1-61208-371-1

The proposed approach allows a modeler using the DEVS formalism to integrate fuzzy logic into the modeling scheme of the system under study just by specifying for one or more of the 4 atomic model functions (δ_{ext}, δ_{int}, λ or t_a): The fuzzification method of the input variables; Write the inference rules linked with the behavior of the system under study; Select the defuzzification method for the output variables. The classical DEVS modeling and simulation has been unchanged and it allows to perform the associated fuzzy inference engine. The next Section will point out how this approach has been implemented in the framework of the DEVSimPy environment.

IV. DEVSimPy Implementation

A. The DEVSimPy Framework

DEVSimPy [6] is an open Source project (under GPL V.3 license) supported by the SPE team of the University of Corsica "Pasquale Paoli". This aim is to provide a GUI for the modeling and simulation of PyDEVS [17] models. PyDEVS is an Application Programming Interface (API) allowing the implementation of the DEVS formalism in Python language. Python is known as an interpreted, very high-level, object-oriented programming language widely used to quickly implement algorithms without focusing on the code debugging [18]–[20]. The DEVSimPy environment has been developed in Python with the wxPython [21] graphical library without strong dependences other than the Scipy [22] and the Numpy [23] scientific python libraries. The basic idea behind DEVSimPy is to wrap the PyDEVS API with a GUI allowing significant simplification of handling PyDEVS models (like the coupling between models or their storage).

Figure 1 depicts the general interface of the DEVSimPy environment. A left panel (see bag 1 in Figure 1) shows the libraries of DEVSimPy models. The user can instantiate models by using a drag-and-drop functionality. The bag 2 in Figure 1 shows the modeling part based on a canvas with interconnection of instantiated models. This canvas is a diagram of atomic or coupled DEVS models waiting to be simulate.

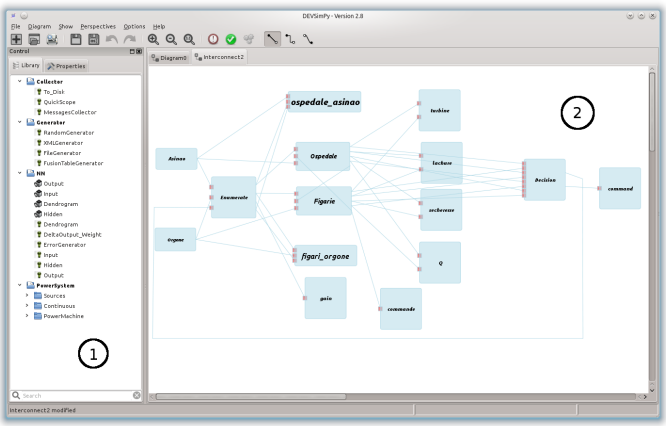

Figure 1. DEVSimPy general interface.

A DEVSimPy model can be stored locally in the hard disk or in cloud through the web in the form of a compressed file including the behavior and the graphical view of the model separately. The behavior of the model can be extended using specific plug-ins embedded in the DEVSimPy compressed file. This functionality is powerful since it makes it possible to implement new algorithms above the DEVS code of models in order to extend their handling in DEVSimPy (exploit behavioral attributes, overriding of DEVS methods, ...). A plug-in can also be global in order to manage several models through an generic interface embedded in DEVSimPy. In this case, the general plug-in can be enabled/disabled for a family of selected models.

B. Case of Study: A Burner tank

1) Description: We have validated the implementation using a classical pedagogical example issued from [15]. This example concerns a fuzzy modeling of a boiler system. The system consists of two subsystems, a burner and a boiler tank. The behavior of the boiler tank is specified as a deterministic one while the behavior of the burner system relies on fuzzy rules. Of course we are mainly interested in this paper into the burner fuzzy modeling and simulation. The burner model has two inputs ports: "on" port and "off" port. Events on "on" (resp. "off") port allow to turn on (resp. off) the burner. A set of specifications involving possibilities that a faulty behavior of the burner emerges has been defined in order to study some unusual or faulty operations. The specifications are expressed as follows (see Figure 2):

- When an input event "on" is applied to the model whose initial state is "OFF" , the possibility to remain in "OFF" is 0.3 and the possibility of state transition to state "IGN" is 0.8.
- When an input event "off" is applied to the model whose initial state is "OFF" , the possibility to remain in "OFF" is 0.9 and the possibility of state transition to state "EXT" is 0.3.
- When an input event "off" is applied to the model whose initial state is "ON" , the possibility to remain in "ON" is 0.2 and the possibility of state transition to state "EXT" is 0.9.
- When an input event "on" is applied to the model whose initial state is "ON" , the possibility to remain in "ON" is 0.7 and the possibility of state transition to state "IGN" is 0.5.
- When an input event "on" is applied to the model whose initial state is "IGN" , the possibility to remain in "IGN" is 0.3 and the possibility of state transition to state "ON" is 0.7.
- When an input event "off" is applied to the model whose initial state is "IGN" , the possibility of state transition to state "ON" is 0.2 and the possibility of state transition to state "OFF" is 0.8.
- When an input event "on" is applied to the model whose initial state is "EXT" , the possibility of state transition to state "ON" is 0.8 and the possibility of state transition to state "OFF" is 0.3.
- When an input event "off" is applied to the model whose initial state is "EXT" , the possibility to remain in "EXT" is 0.4 and the possibility of state transition to state "OFF" is 0.7.

 ISBN: 978-1-61208-371-1

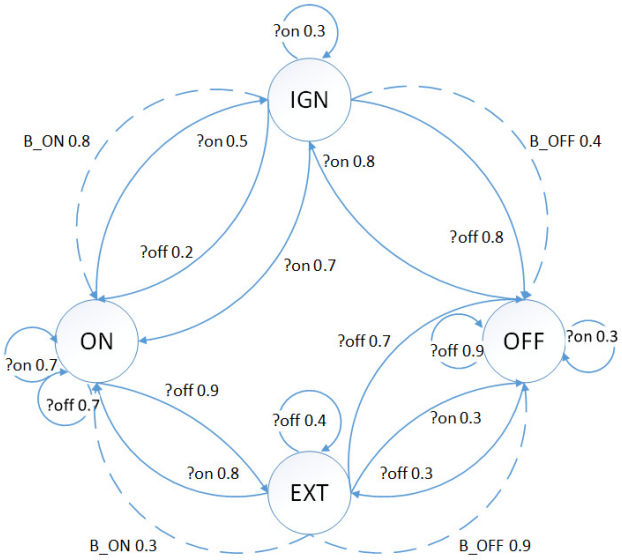

Figure 2. State automaton of the model $Burner$.

- In the state "EXT" ,there are two possible internal transitions: one is to state "ON" with possibility 0.3 and the other to the state "OFF" with possibility 0.9. The model generates an output event ("B_ON" or "B_OFF") according to the transition.
- In the state "IGN" there are two possible internal transitions: one is to state "ON" with possibility 0.8 and the other to the state "OFF" with possibility 0.4. The model generates an output event ("B_ON" or "B_OFF") according to the transition.

2) DEVSimPy Modeling: The DEVSimPy model is given in Figure 3. It is a coupled model involving three atomic models:

- An atomic model $Generator$ allowing to generate the inputs events sent to the burner atomic model;
- The atomic model $Burner$ which implements the fuzzy behavior of the system using three FCL systems; it has two input ports corresponding to the "on" and "off" ports and two outputs corresponding to the "B_ON" and "B_OFF" ports. The automaton of this model (see Fig. 2) is expressed through three set of rules included in three different files embedded in the model. Figure 3 depicts the file corresponding to the δ_{int} function.
- An atomic model $Observer$ which allows to print the obtained solutions issued from the burner atomic model which implements a fuzzy behavior. Some of the obtained printed results are shown in Figure 4.

We have to highlight that, as it can be seen in Figure 3, the defuzzification method is called Dict. In fact it is not a true defuzzification method since the fuzzification does not deal with numerical value but instead a complex structure (a dictionary) involving the potential states. For this reason, no classical defuzzification methods (as COG, COGS, LM, RM, etc.) can be used. The user has to define manually after

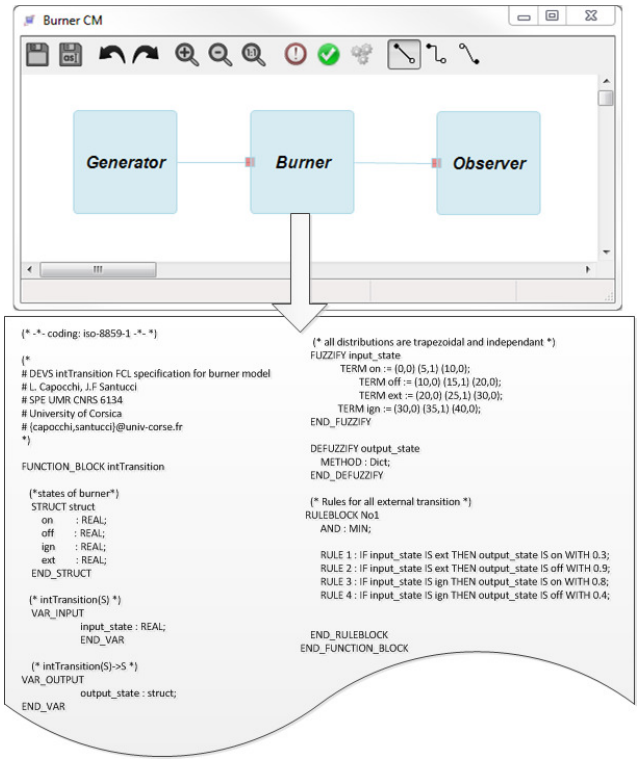

Figure 3. The DEVSimPy implementation of the $Burner$ coupled model.

a function that according to the possibility values obtained after applying the "defuzzification" method Dict, what is the selected state. In regular cases, the user can give priority to the state value possibility being the greatest.

We can point out the three following main advantages of the proposed implementation. First, the use of the FCL language allows a DEVS modeler to access and manipulate both numerical and linguistic information under one common framework. The rules generated by the fuzzy inference are easy to extract and they are already in an understandable, human readable format. Second, the design of the fuzzy controller system is flexible because membership functions can be defined in a large variety of shapes. Third, a FCL system can be developed with the use of a straightforward one-pass build-up procedure. This will permit to avoid a tedious and time consuming activity, especially for the novice fuzzy modeler (as it is often the case for a DEVS modeler).

3) DEVSimPy Simulation: Some simulation results collected by the $Observer$ model are presented in Figure 4. They highlight the interest in using fuzzy logic in the study of such a system. For example, the possibility that the burner sub-system can stay in the state "OFF" despite the fact that an input event should have changed the state of the burner sub-system is expressed in Figure 4. The results obtained by simulation of the model of Figure 3 with an input event "on" and an initial state of the burner system being "OFF" show that the possibility to be in the state "OFF" after the external transition execution due to the input event "on" is 0.3 while the possibility to change to state "IGN" is 0.8. In the same way when the state of the burner sub-system is "EXT" after an internal transition, an output event is generated by the burner sub-system: the

 ISBN: 978-1-61208-371-1

possibility of a "B_OFF" event is 0.9 while the possibility of a "B_ON" event is 0.3.

```
extTransition (with an initial state OFF and an input event on)
ON      : 0.00
EXT     : 0.00
OFF     : 0.30
IGN     : 0.80
outputFnc (with an initial state EXT)
B_OFF   : 0.90
B_ON    : 0.30
```

Figure 4. Simulation results for the external transition function and the output function of the $Burner$ model.

V. Conclusion and future work

This paper proposes an approach to integrate fuzzy logic control into the DEVSimPy framework using FCL language. This approach leans on the coupling of FCL systems with the classical DEVS formalism. The concepts of fuzzy logic control have been introduced in the field of DEVS through an uncomplicated integration scenario. Fuzzy logic controller systems are relatively simple to design in the DEVSimPy framework using FCL system specifications. Involved mathematical insight was not required and the process relied more on intuition and experience for a DEVS modeler. A pedagogical example has been introduced and implemented into DEVSimPy in order to validate the presented approach. This work can further be developed in the following areas: (1) The fuzzification of the time advance function has not yet been validated. We are currently working on this validation using a pedagogical example requiring the modeling of uncertainty of the time advance function. (2) We plan to combine fuzzy logic and neural networks in the framework of the DEVSimPy environment. By cooperating, the two technologies could combine their advantages in order to explain the results of neural networks and to simplify the development of fuzzy rules. (3) A last future orientation concerns fuzzy Inductive modeling [24]. The idea is to design models from observations of Input/Output Behavior. Fuzzy inductive modeling leans on the following four steps: (i) Discretization of quantitative information; (ii) Reasoning about discrete categories; (iii) Inferring consequences about categories; (iv) Interpolation between neighboring categories using fuzzy logic. The future work will also concern the application of the fuzzy modeling and simulation on real cases such as the prediction of the water consumption on the Corsica Island in order to plan the production of water using a water distribution network all over the country or the forecasting of rainfall on the island in order to predict flood or land drought.

References

[1] K. Saleem, "Fuzzy time control modeling of discrete event systems," in Proc. of the World Congress on Engineering and Computer Science, International Association of Engineers (IAENG), 2008, pp. 683–688.

[2] M.-J. Son and T.-W. Kim, "Torpedo evasion simulation of underwater vehicle using fuzzy-logic-based tactical decision making in script tactics manager," Expert Syst. Appl., vol. 39, no. 9, Jul. 2012, pp. 7995–8012.

[3] P. A. Bisgambiglia, L. Capocchi, P. Bisgambiglia, and S. Garredu, "Fuzzy inference models for discrete event systems," in Fuzzy Systems (FUZZ), 2010 IEEE International Conference on, July 2010, pp. 1–8.

[4] Y. Kwon, H. Park, S. Jung, and T. Kim., "Fuzzy-devs formalism : Concepts, realization and application," in Proc. of AIS, 1996, pp. 227–234.

[5] L. A. Zadeh, "Fuzzy sets," Information and Control, vol. 8, 1965, pp. 338–353. [Online]. Available: http://www-bisc.cs.berkeley.edu/Zadeh-1965.pdf [accessed: August, 2014]

[6] L. Capocchi, J. F. Santucci, B. Poggi, and C. Nicolai, "DEVSimPy: A collaborative python software for modeling and simulation of devs systems," in Proc. of 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, June, pp. 170–175.

[7] L. Capocchi. DEVSimPy software. [Online]. Available: http:// code.google.com/p/devsimpy/ [accessed: August, 2014]

[8] I. E. C. technical committee, "Industrial process measurement and control," IEC 61131 - Programmable Controllers, Tech. Rep., 2000, part 7: Fuzzy control programming. IEC.

[9] J. Kelber, S. Triebel, K. Pahnke, and G. Scarbata, "Automatic generation of analogous fuzzy controller hardware using a module generator concept," in Proc. of 2nd European congress on intelligent techniques and soft computing, 1994, p. 8 pages.

[10] G. Nhivekar, S. Nirmale, and R. Mudholkar, "A survey of fuzzy logic tools for fuzzy-based system design," vol. ICRTITCS, no. 9, February 2013, pp. 25–28, published by Foundation of Computer Science, New York, USA.

[11] M. Umano, M. Mizumoto, and K. Tanaka, "FSTDS system: A fuzzy-set manipulation system." Inf. Sci., vol. 14, no. 2, 1978, pp. 115–159.

[12] W. L. Fellinger, "Specification for a fuzzy systems modelling language." Ph.D. dissertation, 1978, ph.D. Thesis, Oregon State Univ., Corvallis.

[13] R. Liebscher, "PyFuzzy python package," http://pyfuzzy.sourceforge.net/ [accessed: August, 2014], 2013.

[14] B. P. Zeigler, "An introduction to set theory," ACIMS Laboratory, University of Arizona, Tech. Rep., 2003. [Online]. Available: http://www.acims.arizona.edu/EDUCATION/ [accessed: August, 2014]

[15] B. P. Zeigler, H. Praehofer, and T. G. Kim, Theory of Modeling and Simulation, Second Edition. Academic Press, 2000.

[16] O. Cordón, "A historical review of evolutionary learning methods for mamdani-type fuzzy rule-based systems: Designing interpretable genetic fuzzy systems," Int. J. Approx. Reasoning, vol. 52, no. 6, Sep. 2011, pp. 894–913. [Online]. Available: http://dx.doi.org/10.1016/j.ijar.2011.03.004 [accessed: August, 2014]

[17] J. S. Bolduc and H. Vangheluwe, "The modelling and simulation package pythondevs for classical hierarchical devs," McGill University, Tech. Rep., 2001, mSDL technical report MSDL-TR-2001-01.

[18] F. Pérez, B. E. Granger, and J. D. Hunter, "Python: An ecosystem for scientific computing," Computing in Science and Engineering, vol. 13, no. 2, 2011, pp. 13–21.

[19] K. J. Millman and M. Aivazis, "Python for scientists and engineers," Computing in Science and Engineering, vol. 13, no. 2, 2011, pp. 9–12.

[20] H. Langtangen, A primer on scientific programming with Python. Springer-Verlag New York Inc, 2011, vol. 6.

[21] N. Rappin and R. Dunn, WxPython in action. Greenwich, Conn: Manning, 2006.

[22] F. J. Blanco-Silva, Learning SciPy for Numerical and Scientific Computing. Packt Publishing, 2013.

[23] T. Oliphant, A Guide to NumPy. Trelgol Publishing, Spanish Fork, UT, 2006, vol. 1.

[24] J. Moreno-Garcia, J. Castro-Schez, and L. Jimenez, "A fuzzy inductive algorithm for modeling dynamical systems in a comprehensible way," Fuzzy Systems, IEEE Transactions on, vol. 15, no. 4, Aug 2007, pp. 652–672.

 ISBN: 978-1-61208-371-1