A NEW APPROACH TO DESCRIBE DEVS MODELS USING BOTH UML STATE MACHINE DIAGRAMS AND FUZZY LOGIC

Stéphane Garredu Paul-Antoine Bisgambiglia Evelyne Vittori Jean-François Santucci
University of Corsica – UMR CNRS 6132 - Dept. of Computer Science
Quartier Grossetti Batiment 018
20250 Corte FRANCE

ABSTRACT

This paper deals with a method which enables to describe a system using both UML State Machine Diagrams and Fuzzy-DEVS (to describe uncertain data) then to perform its simulation using DEVS formalism. The goal of the paper is to simplify the modeling of DEVS models, and also to take into account possible uncertainties on the transitions between states, by using a language based on UML State Machine Diagrams . This language is a part of a larger approach which final purpose is to create a high level intuitive language to enable non-computer scientists to describe DEVS models.

1 INTRODUCTION

In some research fields, the importance of modeling and simulation to describe complex systems has strongly increased. The purpose of modeling is to make a system study be more simple by considering only some of its properties. Simulation makes the model evolve with time, in order to get results.

Our research lab (CNRS SPE) has been working for several years on DEVS formalism defined by Pr. Zeigler during the 70's (2000).

DEVS is a low-level formalism based on the general systems theory; it was introduced by Pr. Zeigler. It allows to describe a system in a modular and hierarchical way, and to consider a global system as a set of other more simple systems, in order to reduce the system's complexity. The simulation is "automatically" performed; the user does not have to focus on the conception of the simulator (Zeigler, Praehofer, and Kin 2000).

DEVS also has great genericity properties, it can be used in many study domains, always considering the fact that a system evolves with events; moreover, DEVS has a good evolutivity, and has often been extended to be able to take into account various systems: dynamic systems (i.e. which structure evolves with time) with DSDE (Barros 1995) and DynDEVS (Uhrmarcher 2001), systems with uncertain parameters with Fuzzy-DEVS (Kwon, Park, Jung, and Kim 1996) and min-max-DEVS (Hamri, Giambiasi, and Frydman 2006), systems with evolutions in their interfaces with Cell-DEVS (cellular approach) (Ameghino, Glinsky, and Wainer 2003) and Vector-DEVS (vectorial approach) (Filippi 2003).

Our goal is to create a high level and intuitive specification language for DEVS models, in order to enable non-computer scientists to create their own models. This language will take place in a Model Driven Architecture, it means that mappings between it and DEVS will be performed using a MDA approach.

The purpose of this paper is to simplify the modeling phasis by defining an intermediate language (based on UML State Machine Diagrams and fuzzy logic) which will take place between the intuitive language which is being developed and DEVS models. This intermediate language will be based on UML State Machine Diagrams (used to describe states and transitions) and fuzzy logic (used to describe uncertain transitions with a set of fuzzy words).

In the first part, we will present the DEVS formalism; we briefly describe Unified Modeling Language and Approximate Reasoning. Those three formalisms can be seen as a basis for this work. DEVS is the core of our approach, and the main purpose of our work is to create a different reasoning (included in our intermediate language) to create DEVS models, using both UML State Machine Diagrams and Fuzzy reasoning.

The second part deals with the description of our approach: we discuss about the importance of the modeling phasis.

The third part will show, through a simple example, how our approach can be applied.

Before giving a conclusion, we discuss in the last part about our results and make some comments

2 BACKGROUND

In this part we present some formalisms and theories which will be used later to explain our approach. We begin with DEVS formalism, the core of our works, then we briefly introduce UML formalism and the State Machine Diagrams included in UML 2.0.

After that, we present the fuzzy reasoning and explain its interest when it is needed to introduce uncertainties when modeling a system. Fuzzy sets theory

and possibilities theory are described, and we present Fuzzy-DEVS formalism which is an extension of DEVS formalism able to take into account uncertainties.

2.1 DEVS formalism

The basic concepts of DEVS approach are intended to control the difficulty of the studied problem, by reducing the analysis of the system to a study composed of a sum of simpler subsystems. Our environment, based on the concept of hierarchy, makes it possible to apprehend the complexity of a problem in a completely gradual way.

The representation of complex systems is generally hard to implement. Indeed, it implies to take into account a multitude of elements, linked themselves by many connections. To reduce the impact of this problem, we use the hierarchical approach of modeling introduced by B.P. Zeigler (2000), which aims at the gradual introduction of the successive components of the system by successive masking of under components.

This approach uses the concepts of Atomic Model and Coupled Model.

An atomic model (black box see Figures 1) makes it possible to account for the behaviours of inputs/outputs and the changes of states of the studied system. The coupled model describes how to connect in a hierarchical way several components (atomic model and/or coupled model) to get a higher level coupled model.

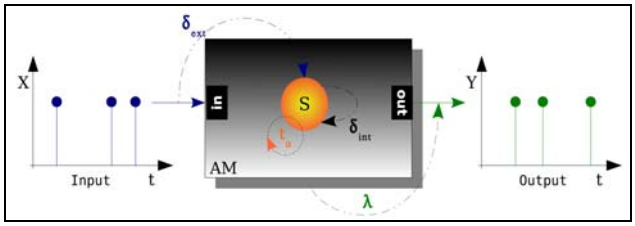

Figure 1 : Atomic Model DEVS.

After receiving an input event, the external transition function is executed. This function updates the state of the system. If there was no entry, after the expiration of the lifespan of the state (t_a), the internal transition functions and the output function are executed. The internal transition function updates the state of the model.

The atomic model (see Figure 1) is defined by:

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

With:

- X, the input ports set, through which external events are received;
- Y, the output ports set, through which external events are sent;
- S, the states set of the system;
- t_a, the time advance function (or lifespan of a state);
- δ_{int}, the internal transition function;
- δ_{ext}, the external transition function;
- λ, the output function;

The simulation is carried out by associating to each component a simulator making it possible to implement the corresponding simulation algorithms.

Thus, DEVS enables the specialist to be completely abstracted from the simulators implementation. Once the model is built and whatever its form is, B.P. Zeigler defined a simulator able to take into account any model described according to DEVS formalism, and developed the concept of abstract simulator (Zeigler, Praehofer, and Kin 2000). The architecture of such a simulator represents an algorithmic description making it possible to implement the implicit instructions of the models resulting from formalism DEVS, in order to generate their behaviour. The major advantage of such a simulator is the fact that its construction is independent from the model.

DEVS is a very good tool for the modeling and the simulation of complex systems but it does not allow a lambda user, who is not a computer scientist, to specify his models.

A specification language establishing the link between knowledge of an expert and a DEVS environment would make it possible to extend the advantages of DEVS.

2.2 UML

Unified Modeling Language is a graphical modeling language and provides a toolkit which enables to model the structural aspects of a system as well as its behaviour (Booch, Rumbaugh, and Jacobson 1998).

Its developpement is driven by the Object Management Group, and its current version is UML 2.0.

One of the main interests of UML is that it is a part of the Model Driven Architecture approach.

MDA[1] (Model Driven Architecture) is a software design approach initiated by the OMG (Object Management Group) in 2001 to introduce a new way of development based upon models rather than code. It defines a set of guidelines for defining models at different abstraction levels, from platform independent models (PIMs) to platform specific models (PSMs) tied to a particular implementation technology. The translation between a (1) PIM and one or more PSMs is to be performed automatically by using transformation tools.

Another interest of UML is a language which is provided to model behavior, and can be used in software engineering (for instance, to model the behavior of class instances) as well as in modeling and simulation (to model states and transitions) : this language is named Statecharts (State Machine Diagrams in UML 2.0).

[1] http://www.omg.org/mda/

Statecharts are a high-level graphical-oriented formalism used to describe complex reactive systems. They were developped by D. Harel (1987) and are an extension of state-transition diagrams (the diagrams representing a Finite State Machine or Automaton) (Hopcroft, Motwani and Ullman 2001). They were added three concepts : orthogonality (the way parallel activities are achieved), composition hierarchy (depth nesting of states) and broadcast communication (events sent from one to many elements).
Statecharts are composed of eight basic elements: Labels, Transitions, States, Actions, Conditions, Events, Expressions and Variables.
Statecharts formalism has been evolving for years now. The most popular derived formalisms are the Classical UML, and those implemented by Rhapsody (Crane and Dingel 2005; Harelvand and Kuglerv 2004). Moreover, several other semantics provide them a pretty good evolutivity.
We chose as an intermediate language UML State Machine Diagrams, because they provide a set of graphical elements to specify models. Of course we modified some aspects of this formalism to take into account fuzzy logic.

A state is represented by a rounded rectangle named *blob*.

A transition is an arrow which joins two states. It is triggered by an event.
The general syntax for a transition is *m*[*c*]/*a* where :

- *m* is the event which triggers the transition;
- *c* is the guard, that is to say the condition (or set of conditions) which guards the transition;
- *a* is the action (or the set of actions) which must be executed when the transition fires.

The values in square brackets are guards, and we use them in our approach to express possibilities on transitions.

An initial state is shown as a black filled circle, it cannot have any incoming transitions i.e. once this state left, it is not possible to re-enter it.

A final state is shown as a circle surrounding a small solid filled circle. Once entered in this state, it is not possible to go back (i.e. it can only have incoming transitions, and it cannot have any outgoing transitions). In other words, it is the end of the simulation.

A state can be associated to an action, or several actions : they are written under the name of the state.

2.3 Fuzzy Reasoning

Fuzzy logic was introduced by Goguen (1968-1969) and presented by Zadeh (1975) as a framework for approximate reasoning, and in particular to handle knowledge expressed using the natural language. Thus, fuzzy logic can be seen as an approach of human reasoning.

Fuzzy logic is a set of mathematical theories which allow representing and handling inaccurate or uncertain data.

2.3.1 Fuzzy Sets Theory

Zadeh introduced in 1965 the fuzzy sets theory (1965), the first theory of fuzzy logic, which gives to an element the possibility to belong to a set, according to a membership degree. This membership degree belongs to the [0...1] interval, where 0 is the case where the condition can not be fulfilled, and where 1 is the case where the condition is always fulfilled. Classical logic handles Boolean variables, which possible values are 0 or 1.

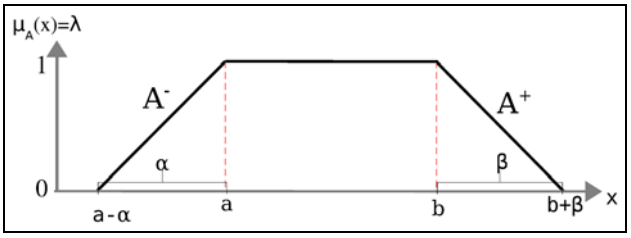

Figure 2: Membership function example. This representation allows us to define the membership degree of an element in a set.

As the purpose of fuzzy logic is to reason with partial knowledge, it replaces Boolean variables by Fuzzy variables. A fuzzy variable can be represented by an interval described as follows:

- a = value, expresses the membership degree = 1;
- b = value, expresses the membership degree = 1;
- alpha = confidence interval;
- beta = confidence interval.

Thanks to this representation mode, we are able to represent a real value by fixing a=b=constant and alpha=beta=0.

2.3.2 Possibilities Theory

As we said before, fuzzy sets theory was the first theory of fuzzy logic, it extends the classical set theory in order to take into account inaccuracy, it is based on membership functions.

Possibilities theory (Zadeh 1978) is a part of fuzzy logic which enables to take into account uncertainties which are impossible to describe with probabilistic theory. It is based on possibility functions.
Reasoning using probabilities implies to be able to define, for each event, its probability.

Hence, events must be well known. If this knowledge is not totally available, another possible solution is to reason in terms of possibilities.

Let X be a reference set, P(X) the set of parts. Each element of P(X) is given a possibility coefficient between 0 (impossible) and 1 (always possible).

2.3.3 Fuzzy-DEVS

In this subsection we focus on Fuzzy-DEVS, an initial extension of DEVS which takes into account the fuzzy transitions between states.
The Fuzzy-DEVS formalism was introduced by Kwon in (1996), drifts DEVS formalism while keeping its semantics, some of its concepts and its modularity. It is based on fuzzy logic, the "Max-Min" rules shown in (Kwon, Park, Jung, and Kim 1996) and the methods of fuzzification and defuzzification.

To allow the simulation, imprecise parameters must be transformed into crisp parameters (defuzzification) ; to be exploited, the output data is again transformed into fuzzy data (fuzzification).

A Fuzzy-DEVS model takes into account the different possibilities of transitions (δ_{ext} and δ_{int}) between states. The various possibilities of input, output and state update are represented by matrices and the evolution of the model by possibilities trees (Kwon, Park, Jung, and Kim 1996; Anglani, Caricato, Grieco, Nucci, Matta, Semeraro and Tolio 2000).

Fuzzy-DEVS does not address the fuzzy values of a model, but proposes a methodology that provides a tree of options describing various transitions between the states of the system. Fuzzy-DEVS is a theoretical formalism still in research phasis. This approach does not appear fully consistent with the DEVS formalism, but it provides avenues for good work, like the ability to define the lifespan of a state (t_a) with a linguistic label.

In the following part we describe our modeling approach. DEVS can be seen as a multi formalism : it is the core of our works.

There are often several uncertainties when a natural system is being modeled using a state/transitions method. We chose to simplify the problem by taking into account only uncertainties linked to transitions. Fuzzy-DEVS, as a DEVS extension, is a part of the DEVS multi formalism and offers the possibility to include such uncertainties during the modeling and simulation process.

As we want to work at an higher level than DEVS, we use State Machine Diagrams which, thanks to its properties (it is graphical and located at an higher level than DEVS), is a good modeling tool which fulfills many criteria as shown in (Garredu, Vittori, Santucci and Muzy 2006).

3 OUR APPROCH

We chose to focus on the modeling phasis, because once the system is described, the simulator is provided (cf. DEVS properties).

Our approach tries to use both Fuzzy-DEVS and State Machine Diagrams to help a scientist to describe models. It is an intermediate approach, because the final purpose is to use a high-level language (Garredu, Bisgambiglia, Vittori and Santucci 2007).
State Machine Diagrams , such as DEVS, is a formalism based on states and transitions. Moreover, it is graphical : hence, it is easier to represent a system with State Machine Diagrams than DEVS. It could be a good intermediate language to specify DEVS models.

We thought it was interesting to add to the description of the model the possibility for the scientist to express inaccurate data using linguistic terms.

We consider that once the states known, the only differences will be on the way they will connect to each other, in other terms inaccuracies will be expressed only in the transitions between those states.

That is why Fuzzy-DEVS will help to specify for the transitions:

- the possible date before a transition fires, using linguistically terms;
- an execution coefficient.

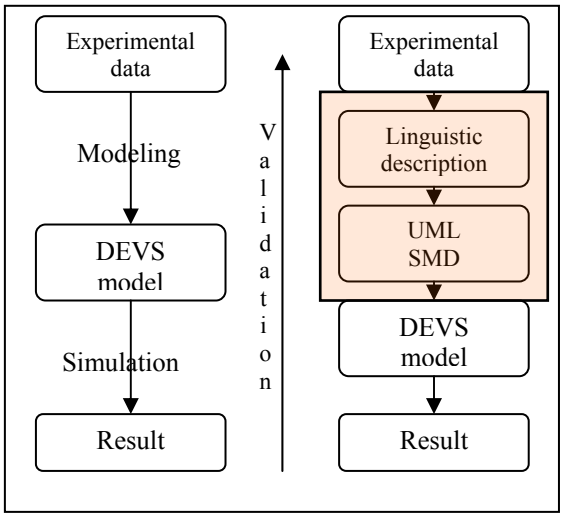

Figure 3: Linguistic description to specify UML State Machine Diagrams models.

3.1 Goals

In this section we give a graphical representation of the goals of this paper, shown in figure 3. The main part of our approach is to give a method to model a system using both State Machine Diagrams and a linguistic description.

Moreover, uncertainties on transitions are taken into account.

3.2 General Reasoning

Before starting the modeling phasis, some steps have to be followed.

The first step for the scientist is to identify all the states of the model, and among them the one which will be the initial state. The second one is to define how those states will behave with each other, i.e. he will have to define the transitions.

The last step is to define both the inputs and the outputs of the system, and how the simulation can be started or stopped.

Once those steps finished, we are able to begin the modeling phasis.

The states become *blobs* : for every different state, there is a different blob.

In order to take into account the lambda function, we chose to represent it with actions on transitions and the variable *out*.

The *delta int* and *delta ext* functions are always defined on transitions, because *delta* functions describe how to change from one state to another (i.e. a transition).
The *delta int* function can be translated using the keyword *after* (followed by a duration).
Some times, the expert who wants to model the system does not know exactly the lifetime of a state, represented by *ta* function.

Thanks to fuzzy logic, we can represent *ta* using a membership function, composed of several typical durations expressed with fuzzy words.
So, instead of giving a numerical duration after the keyword *after*, the scientist will be given the possibility to give a fuzzy word.

Fuzzy logic also provides the ability to put a possibility on a transition. Using state machine diagrams, such a possibility can be expressed with a guard. A high possibility will have a value close to 1, and a possibility which value is 1 is certain to happen.

4 EXAMPLE

In this part we apply our method on a simple two-state example. The studied system is a reset-set system, with uncertainties on transition durations.

In the first part, we textually describe the system, as a non-computer scientist could have described it. We provide a list of fuzzy words (to describe durations) to help him during the description process.

In the second part, we create the model using the method explained in chapter III.

In the last part, we translate our model into a DEVS atomic model.

4.1 System

This system is a simple Reset-Set system. It can be described by an expert as follows : the system remains in the reset state (which is the initial state), until it receives an external event which sends a 1 to the input port (for instance, an event from another model). When this external event is received, the *set* state is activated.

The system waits into this state for an unknown duration and then and goes back to the *reset* state. This duration is not exactly known but can be described with a word. Moreover, the transitions between the two states are not certain, even if their possibility is very high.

4.2 Modeling the system

With the expert's textual description given in A there are two states (*reset* and *set*) when we model the system with State Machine Diagrams.

In this example, there is an unknown duration on the transition from *set* to *reset*. The duration is defined by a membership function and linked to a keyword chosen from the list {null; short; medium; long}.

The membership function is described as follows (Figure 4):

- Null = [0,1]
- Short = [2,4,1]
- Medium = [6,8,1]
- Long = [10, ,1]

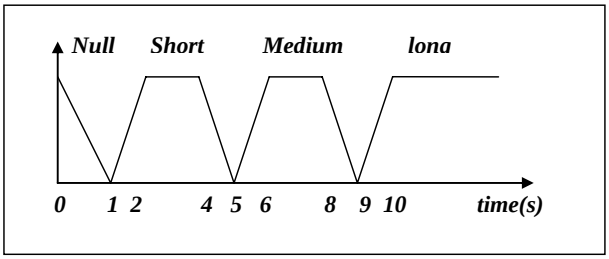

Figure 4: Time membership function.

This unknown duration will be translated and applied to the transition from *set* to *reset* as a time event, using the keyword after. Moreover, the two transitions between *reset* and *set*, are not certain.

They are given a probability degree, expressed with a guard. When an external event arrives on the input port of the system, the transition from *reset* to *set* is triggered. The probability degree of this transition is high but not certain. The final model is given in Figure 5.

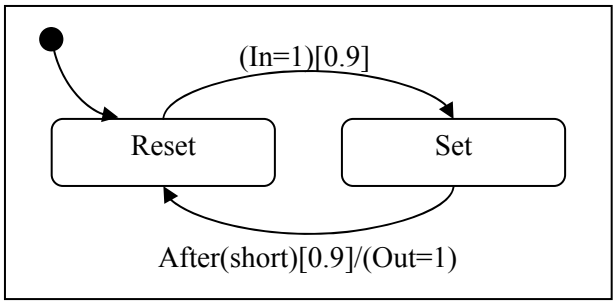

Figure 5: Representation of the system using State Machine Diagrams and Fuzzy-DEVS for the transitions.

4.3 DEVS model

$Atomic_{RS} = < X, Y, S, t_a, \delta_{int}, \delta_{ext}, \lambda >$

- $X = \{In\}$
- $Y = \{Out\}$
- $S = \{Reset, Set\}$
- $\delta_{ext}(Reset, In=1) = \{Set\}$
- $\delta_{int}(Set) = \{Reset\}$
- $\lambda(Set) = \{Out=1\}$

The following diagram is a possibilities tree which gives us an idea of the most possible path followed by the model (Figure 6).

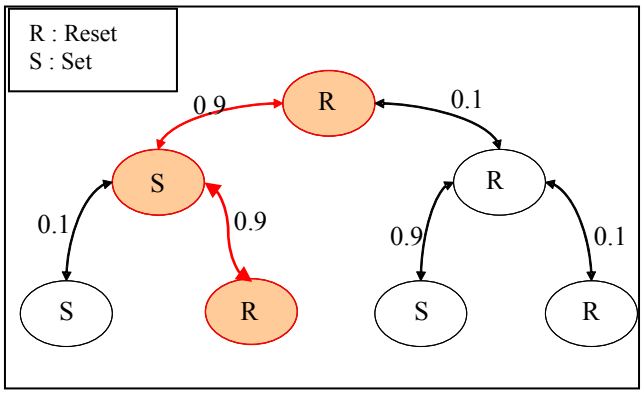

Figure 6: The possibilities tree for the simulation of the model defined by Min-Max Rules Fuzzy-DEVS.

5 REMARKS AND DISCUSSIONS

This example was chosen as simple as possible: it represents an atomic model. The number of states is finite and not too high.

Increasing the number of states would increase the size of the possibilities tree, and of course the computation time to create it.

The "classical" transitions are modified because we use possibilities as guards, and fuzzy words instead of absolute (or relative) numerical values to specify the lifespan of a state. So we think it would be useful to create a specific profile, or make an extension of State Machine Diagrams meta model.

We only treated possibilities on transitions, and we will try in a near future to include imprecise data on inputs/outputs using iDEVS method (Bisgambiglia, De Gentili, Santucci and Bisgambiglia 2006).

We also plan to implement an automatic code generation between this intermediate language and DEVS formalism.

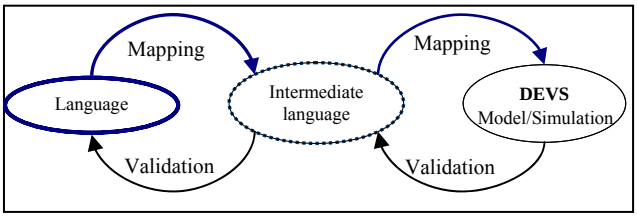

Figure 7: Our global approach.

6 CONCLUSION

DEVS is an interesting tool for M&S. A community of scientists works all around the world in order to add DEVS other modules which can fit several applications fields.

We think it is important to enable the largest part of the scientists to use DEVS formalism that is why we follow an approach which purpose is to offer the possibility to model systems using a more intuitive way.

In this paper, we proposed a new idea, using both the possibility theory and a high level specification language.

The ultimate goal of this idea is to enable a non-computer scientist to create a model using a simple and intuitive specification language.

To reach this goal, we use several tools:

- UML which allows the definition of models using a graphical way based on diagrams, because a graphical description is often easier than code writing.
- The possibilities theory, which enables the description of a system with pre-defined terms. Those pre-defined terms are useful to describe a system and its inaccuracies using a suitable language.

Those tools are linked to the M&S DEVS formalism, because the simulation of the models is easy to perform with DEVS.

This approach has to be replaced in a larger project, which is the definition of a high-level and intuitive language. UML State Machine Diagrams will only be used to create a link between this language and DEVS, as an intermediate formalism. We will probably define a UML profile for our intermediate language; this will be helpful to reuse some interesting parts of State Machine Diagrams , and to specialize them. This new language is being developed, and will be defined in order to be used

by the largest part of the scientists. For instance, it will be possible to define models in a graphical way, or/and using a formal language.

REFERENCES

Zeigler, B., H. Praehofer, and T. Kim. 2000. Theory of Modeling and Simulation, Second Edition. Academic Press.

Booch, G., J. Rumbaugh, and I. Jacobson. 1998. The Unified Modeling Language User Guide. Addison-Wesley.

Harel, D. 1987. Statecharts : A visual formalism for complex systems. S*cience of Computer Programming*, 8(3):231-274.

Hopcroft, J.E., R. Motwani, and J.D. Ullman. 2001. Introduction to Automata Theory, Languages, and Computation, 2nd edition. Addison Wesley.

Crane M.L. and J. Dingel. 2005. UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal. Model Driven Engineering Languages and Systems: 8th.

Harel, D., and H.Kugler. 2004. The Rhapsody Semantics of Statecharts. Integration Of Software Specification Techniques for Applications in Engineering.

Barros, F. 1995. Dynamic structure discrete event system specification : a new formalism for dynamic structure modelling and simulation. In *Proceeding of the 1994 Winter Simulation Conference*.

Uhrmarcher, A. 2001. Dynamic Structures in Modeling and Simulation : A Reflective Approach. *ACM Transactions on Modeling and Computer Simulation vol. 11 2001*, 206-232.

Kwon, Y., H. Park, S. Jung, and T. Kim. 1996. Fuzzy-DEVS Formalism : Concepts, Realization and Application. Proceedings AIS 1996, 227-234.

Hamri, A., N. Giambiasi, and C. Frydman. 2006. Min-Max-DEVS modelling and simulation. Simulation Modelling Practice and Theory (SIMPAT), vol. 14, pp. 909–929. Ed. Elsevier, ISSN 1569–190X.

Ameghino, J., E. Glinsky, and G. Wainer. 2003. Applying cell-devs in models of complex systems. In *Summer Computer Simulation Conference*, Montreal QC, Canada.

Filippi, J. 2003. *Une architecture logicielle pour la multi-modélisation et la simulation à évènement discrets de systèmes naturels complexes*. PhD Thesis, University of Corsica.

Goguen, J. A. 1968-1969. *The logic of inexact concepts*. *Synthese* 19 : 325-373.

Zadeh, L. A. 1975. *Fuzzy logic and approximate reasoning. Synthese* 30 : 407-428.

Zadeh, L. A. 1965. Fuzzy sets. *Information and Control* 8: 338-353.

Zadeh L. 1978. *Fuzzy sets as a basis for a theory of possibility*. Fuzzy Set and Systems, vol. 1, 1978, p. 3-28.

Anglani, A., P. Caricato, A. Grieco, F. Nucci, A. Matta, G. Semeraro, and T. Tolio. 2000. *Evaluation of capacity expansion by means of fuzzy-devs*. (citeseer.ist.psu.edu/499458.html), 14th European Simulation MultiConference. Ghent, Belgium.

Garredu, S., V. Evelyne, J.F. Santucci, A. Muzy. 2006. *Specification languages as front-ends towards the DEVS formalism*. In Proceeding of the Environment Identities and Mediterranean Area, ISEIMA '06. 104-109.

Garredu, S., P.A. Bisgambiglia, E. Vittori and J.F. Santucci. 2007. *Towards the definition of an intuitive specification language*. In Proceeding of the Simulation and Planning in High Autonomy Systems (AIS) & Conceptual Modeling and Simulation (CMS).

Bisgambiglia, P.A., E. De Gentili, J.F. Santucci, and P. Bisgambiglia. 2006. *DEVS-Flou: a discrete events and fuzzy sets theory-based modeling environment*. International Symposium on Systems and Control in Aeronautics and Astronautics – Harbin (China). 95-100.