DISCRETE EVENT MODELING AND SIMULATION ASPECTS TO IMPROVE MACHINE LEARNING SYSTEMS

Laurent Capocchi
Jean-Francois Santucci

SPE UMR CNRS 6134
University of Corsica
Campus Grimaldi
20250 Corte, France
{capocchi, santucci}@univ-corse.fr

Bernie P. Zeigler

RTSync Corp
12500 Park Potomac
Potomac, MD
zeigler@rtsync.com

ABSTRACT

Discrete Event Modeling and Simulation (M&S) and Machine Learning (ML) are two frameworks suited for system modeling which when combined can give powerful tools for system optimization for example. This paper details how discrete event M&S could be integrated into ML concepts and tools in order to improve the design and use of ML frameworks. An overview of different improvements are given and three concerning Reinforcement Learning (RL) are implemented in the framework of the DEVS formalism.

Keywords: DEVS, Machine learning, volatility, Artificial Intelligence.

1 INTRODUCTION

Planning and decision problems belong to an area of Artificial Intelligence (AI) that aims to produce plans for an autonomous agent, that is, a set of actions that it must perform in order to reach a given goal from a known initial state. Because of the stochastic nature of the problems, the planning and decisions very often lean on Machine Learning (ML) (Alpaydin 2016). However, the computer implementation of models and algorithms associated with ML may require lot of work because: (i) it is difficult to select which model (algorithm) will fit with the problem to solve (there is a large choice of models allowing to develop a ML algorithm) (ii) once the model has been chosen, it is difficult to select the hyper-parameters of the chosen model that will allow to give good results after the learning phase (iii) classical tools dealing with ML do not allow to connect learning agents involved in ML with a powerful simulation environment. This lack of connection implies difficulties to deal with temporal aspects which may appear in the problem definition or to deal with multi-agent or even harder to deal with dynamical aspects as agents that may appear or disappear during simulation. The idea of combining M&S with ML makes it easier to solve difficult problems such as: smart parking management (which involves simulations of parking time prediction) or the management of healthcare systems (which involves simulations to help and support decision-making in the increasingly complex patient treatment process). Also the help simulation-based multi-agent ML could give to the human/human and human/machine collaboration (Busoniu, Babuska, and Schutter 2006).
In order to bring a solution to these important problems we propose an approach based on discrete-event Modeling and Simulation (M&S) using the DEVS (Discrete EVent system Specification) (Zeigler, Praehofer, and Kim 2000) and SES (System Entity Structure) (Zeigler, Sarjoughian, Duboz, and Souli 2012)

 ©2019 Society for Modeling & Simulation International (SCS)

formalisms. The DEVS formalism has been introduced as a mathematical abstract formalism for the M&S of discrete-event systems allowing a complete independence from the simulator using the notion of abstract simulator. DEVS defines two kinds of models: atomic 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 atomic/coupled 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. A fundamental representation of DEVS hierarchical modular model structures is the SES which represents a design space via the elements of a system and their relationships in hierarchical and axiomatic manner. SES is a declarative knowledge representation scheme that characterizes the structure of a family of models in terms of decompositions, component taxonomies, and coupling specifications and constraints.
In this paper we will concentrate on exploring a set of subjects that has been much neglected in the context of ML concerning in particular Reinforcement Learning (RL)(Sutton and Barto 1998) and Markov Decision Processes (MDPs) (Puterman 1994). It is important to add the following features: (i) definition of a DEVS explicit modeling aspects with agents and environment components involved in RL (ii) introduction of temporal DEVS aspects in RL (iii) proposition of DEVS-based multi-agents M&S scheme. We will focus on the descriptions of set of topics which can be called as "difficult problems" in the field of RL and on the way these problems can be treated within the paradigms involved by the DEVS and SES formalisms. In doing this, we will offer to designers of ML systems based on MDPs an easy way to perceive, interpret, structure and solve the difficult problems which have been previously pointed out.
The rest of the paper is as follows: the section 2 will describe how DEVS M&S could be integrated into ML concepts and tools in order to improve the design and use of ML frameworks. Section 3 will point out how DEVS has been use to: (i) explicitly separate agent and environment components in RL implementations (ii) take into account temporal aspects in RL (iii) extend RL towards multi-agents M&S. The last section will conclude and describe future work.

2 MODELING AND SIMULATION AND MACHINE LEARNING

Modeling and Simulation (M&S) and Artificial Intelligence (AI) are two domains that can complement each other. For instance, the AI can help the "simulationist" in the modeling of complex systems that are impossible to represent mathematically (Nielsen 1991). On the other hand, M&S can help AI models failed to deal with complex systems for lack of simple or unworkable heuristics.

Systems that already use AI, such as digital supply chains, "smart factories" and other industrial processes in Industry 4.0, will inevitably need to include AI in their simulation models (Foo and Peppas 2004). For example, with simulation analysis systems, AI components can be directly integrated into the simulation model to enable testing and forecasting. In (Meraji and Tropper 2010) the authors use a recursive learning algorithm (Q-Learning) to combine a dynamic load balancing algorithm and a bounded-window algorithm for discrete-event simulation of VLSI circuits (Very Large Scale Integration) at gate level.

The Figure 1 depicts some possible integration of M&S aspects into ML framework. Machine learning is a type of AI that use three types of algorithm (Supervised Learning, Unsupervised Learning, Reinforcement Learning) in order to build models that can get input set to predict an output set using statistical analysis. For the simulation part, Monte Carlo simulation is often used to solve ML problems by using a "trial and error" approach. Monte Carlo simulation can also be used to generate random outcomes from a model estimated by some ML technique.
ML for the optimization is another opportunity for integration into simulation modeling. Agent-based systems often have many hyper-parameters and require significant execution times to explore all their permutations to find the best configuration of models. The ML can speed up this configuration phase and provide more efficient optimization. In return, simulation can also speed up the learning and configuration process

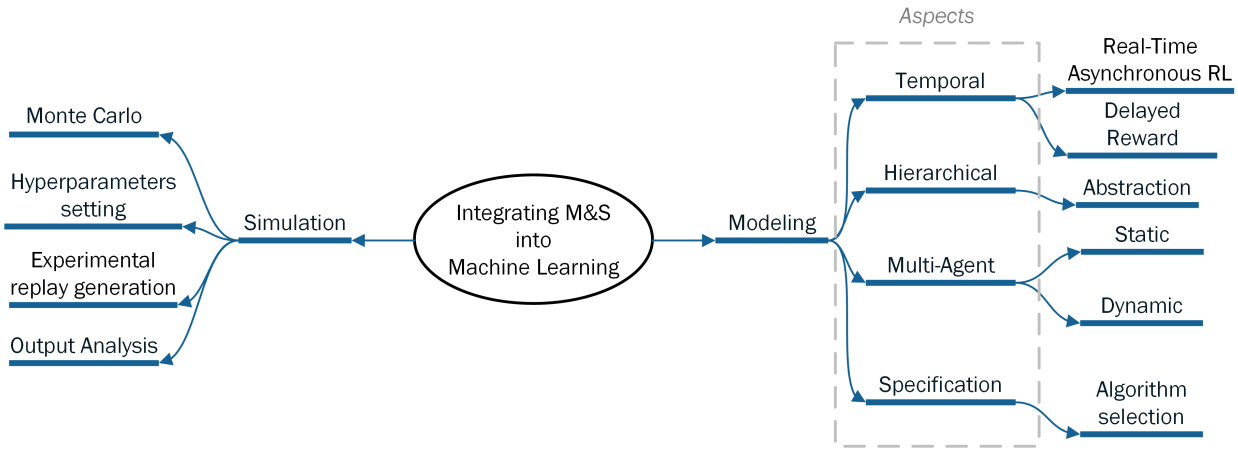

Figure 1: M&S into Machine Learning.

of AI algorithms.
Simulation can also improve the experimental replay generation in RL problems where the agent's experiences are stored during the learning phase.
RL components can be developed to replace rule-based models. This is possible when considering human behavior and decision-making. For example, in (Floyd and Wainer 2010) the authors consider that by observing a desired behavior, in the form of outputs produced in response to inputs, an equivalent behavioral model can be constructed (Output Analysis in figure 1). These learning components can either be used in simulation models to reflect the actual system, or be used to train ML components. By generating the data sets needed to learn neural networks, simulation models can be a powerful tool in deploying the algorithms of recursive learning.

Concerning the modeling part, many modeling aspects can be highlighted to help ML implementation:

- Temporal: Basically, the temporal aspect is implicitly in the RL models. For instance, MDPs consider the notion of discrete and continuous time in order to consider the life time of a state. In RL models, the introduction of time in the awarding of rewards allows the modeling of non-immediate response of a system. The use of the notion of time in the RL models also makes possible to carry out asynchronous simulations from real data.
- Hierarchical: The abstraction hierarchy allows the modelling of RL systems with different levels of detail. This makes possible to determine optimal policies by levels of abstraction and therefore to have more or less precise policies depending on the level chosen by the user.
- Multi-Agent: The multi-agent modeling can be used as part of RL where the optimal policy is based on a communication between an environment and one or more agents that are instantiated in a static or dynamic way.
- Specification: Assembling all the ML pieces needed to solve problems can be a daunting task. There are a lot of ML algorithms to choose from and deciding from where to start can be discouraging. By using specifications based on models libraries and system input analysis, the choice of the appropriate algorithm becomes simpler.

This paper presents the benefits of DEVS formalism aspects to assist in the realization of RL models.

3 REINFORCEMENT LEARNING AND DEVS FORMALISM

AI learning techniques have already been used in a DEVS simulation context. Indeed, in (Saadawi, Wainer, and Pliego 2016) the authors propose the integration of some predictive algorithms of automatic learning in the DEVS simulator in order to considerably reduce the execution times of the simulation for many applications without compromising their accuracy. In (Toma 2014), the comparative and concurrent DEVS simulation is used to test all the possible configurations of the hyper-parameters (momentum, learning rate, etc.) of a neural network. In (Seo, Zeigler, and Kim 2018), the authors present the formal concepts underlying DEVS Markov models and how they are implemented in MS4Me software (Zeigler, Sarjoughian, Duboz, and Souli 2012). Markov concepts of states and state transitions are fully compatible with the DEVS characterization of discrete event systems. In (Rachelson, Quesnel, Garcia, and Fabiani 2008), temporal aspects has been considered into Generalized Semi-MDPs with observable time and a new simulation-based RL method has been proposed.

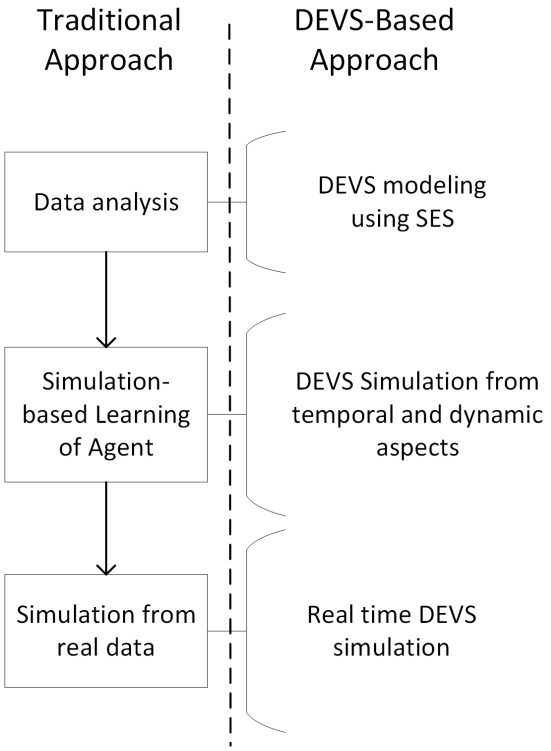

Figure 2: Traditional RL process and DEVS formalism.

More generally, the DEVS formalism can be used to facilitate the development of the three traditional phases involved in a learning process by reinforcement of a given system (Figure 2):

- The Data Analysis phase consists of an exploratory analysis of the data which will make it possible to determine the type of learning algorithm (supervised, unsupervised, reinforcement) to deal with a given decision problem. In addition, this phase also makes it possible to determine the state variables of the future learning model. This phase is one of the most important in the modeling process. As noted in Figure 2, the System Entity Structure (SES) (Zeigler, Sarjoughian, Duboz, and Souli 2012) can be used to define a family of RL algorithm models (DQN, DDQN, A3C, Q-Learning, SARSA,

etc.) (Sutton and Barto 1998) based on the results of the data analysis that uses both statistic tools and the nature of the RL model (model-free/model-based and on-Policy/off-Policy).

- The Simulation-based learning of an agent consists of simulating entry sets of a learning model to calibrate it by avoiding over-learning (learning phase). The DEVS formalism makes it possible by simulating the environment as an atomic model. However, the environment that is interacting with the agent as part of a traditional RL scheme, can also be considered as a coupled model composed of several interconnected atomic models. It is in this context that the environment is considered as a dynamic multi-agent DEVS model in which the number of agents can vary over time.
- The real-time simulation phase consists of submit the model to the actual input data (test phase). The DEVS formalism and its experimental frame is an excellent candidate to simulate in real time the decision policies from real simulation data.

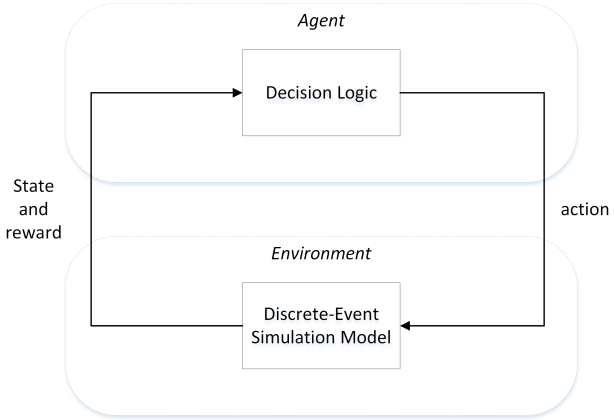

Figure 3: Learning by reinforcement.

The DEVS formalism allows to formally model a complex system with a discrete-event approach. The diagram of a RL system (Figure 3) suggests that the Agent and Environment components can be represented by DEVS models (atomic or coupled depending on the modeling approach).

The coupling of RL and DEVS can be done by two ways:

- DEVS can integrate the algorithms of the RL into its modeling specifications (transition functions, SES - modeling part) or within its simulation algorithms (sequential, parallel or distributed - simulation part) in order to benefit from an AI for example, improving the pruning phase in the specification process using SES or improving simulation performance by introducing a neural network-based architecture into the simulation engine.
- AI algorithms can benefit from the DEVS formalism to improve the convergence or search for optimal hyper-parameters, for instance. Moreover, multi-agent RL algorithm can be modeled by DEVS due to its modular and hierarchical modeling possibilities. The paper focused in this last point.

There are several ways to integrate DEVS modeling into RL algorithms. Basically, an agent and an environment communicate in order to converge the agent towards an optimal policy. Thanks to the modular and hierarchical aspect of DEVS, the separation between the agent and the environment within the RL algorithms is improved.

The atomic model Environment will respond to the solicitations of the agent atomic model by assigning it a new state s, a reward r according to an action it has received. In addition, the model will inform if it reaches

the final state or not thanks to a variable Boolean d. It is therefore through an external transition function that this communication will take place. The output function will be activated immediately after the external transition function to send the pair (s, r, d) to the Agent model. The internal transition function will update the model's state variables without generating an output. Finally, the initial state will be to determine the list of possible states S and actions A and the reward matrix R. The model is responsible for the generation of episodes (when a final state is reached).

The atomic model Agent is intended to respond to events that occur from the Environment model. When it receives a new tuple (s, r, d), it will return an action following a policy depending on the implemented algorithm (ε-greedy for example). The model has a Q state variable which is an SxA dimension matrix for implementing the learning algorithm (Q-Learning or SARSA for example). When convergence is reached (depending on the values of Q), the model becomes passive and no longer responds to the environment. The update of the Q attribute is done in the external transition function after receiving the tuple (s, r, d). The internal transition function makes the model passive. The output function is activated when the external function is executed.

This section presents how DEVS is used to specify a RL system represented by an interconnection of an Agent and Environment DEVS models. The next subsections introduce how DEVS could be used when dealing with temporal and multi-agents aspect.

3.1 Temporal Aspect

DEVS is based on a formal representation of time in finite state automata. When we talk about deferred reward, we can think of a simulation time exploitation in order to make possible a reward shifted in time (example of an agent that does not respond immediately to his environment). It would be interesting to see what would be the consequences on the final policy in term of time to take a decision. Still on the notion of time, it would be interesting to invoke the determination of optimal policy at separate time intervals and to compare the best policy over time.

Markov Modeling is among the most commonly used forms of model expression. Indeed, Markov concepts of states, state transitions and state life time are fully compatible with the DEVS characterization of discrete-event systems. These concepts are a natural basis for the extended and integrated Markov modeling facility developed within M&S environments (e.g., MS4Me (Zeigler, Sarjoughian, Duboz, and Souli 2012) or DEVSimPy (Capocchi, Santucci, Poggi, and Nicolai 2011)). Due to their explicit transition and time advance structure, DEVS Markov models (Seo, Zeigler, and Kim 2018) can be individualized with specific transition probabilities and transition times/rates which can be changed during model execution for dynamic structural change. The basic interpretation of a transition probabilities is of a Markov chain, i.e., a set of states that generate sequences determined by the probability structure. The transition times/rates allows us to work with the times of the transitions. These can be referred to variously and equivalently as sojourn times, transition times, time advances, elapsed times, or residence times depending on the context.

3.2 Multi-agent Aspect

This section provides an introduction towards multi-agent RL. We mainly focus on the possibility to introduce multi-agents in RL in order to solve dynamic tasks online, using algorithms that originate in the classical temporal-difference RL used previously.

For example, you could imagine simultaneously training a large number of competing agents and eventually also mix the types of agents that we are training. One could also use the real-world data as a supervised feedback signal to "force" the agents in the simulation to collectively behave like the real world.

3.2.1 DEVS Modeling

In order to model a multi-agent RL, a generic framework is proposed to represent the environment part of RL as a DEVS coupled model connected with one or more agents.

The coupled model describing the Environment has 2 outputs corresponding to the reward and the state computed by simulation and one input corresponding to the action selected by a learning agent. Two kinds of agents may be defined as atomic models: (i) learning agent whose behavior is the same as defined previously (ii) agents driven by other kinds of algorithms than a learning procedure. The different agents share the same global vision of the environment embedded into a single atomic model (global environment atomic model). Furthermore, the Q-learning algorithm is implemented using the atomic models presented previously (Figure 3): (i) a DEVS atomic model corresponding to the learning agent (ii) an environment DEVS atomic model allowing to isolate the information specific to the learning algorithm associated with a learning agent.

3.2.2 Pursuit-Evasion Case Study

In order to illustrate the use of the DEVS formalism to implement a RL framework involving multi-agents, we propose to use a case study known as Pursuit-Evasion (Hespanha, Kim, and Sastry 1999).
One or more pursuing agents, here called "cat", must have one or more fugitive agents, called "mouse". This type of problem distinguishes Search and Rescue problems in the sense that the fugitive agent tries at all costs to avoid pursuers. The mouse-cat problem is played one by one: each agent performs an action and must wait for all other agents to perform an action before executing the next one. In this paper, we consider only two agents: one mouse and one cat. We also consider a discrete-time: each action corresponds to a unit of time $t \in \tau := 1, 2, ..., T$. The reflection and observations are considered to be performed in zero time.

Like time, the space of the problem is discrete. The space in which the part is unfolded is thus a finite and discrete set of cells $x := 1, 2, ..., X$ arranged in two dimensions. All movements are therefore discrete and concern the displacement of an integer number of squares, defined by the unit by default. We will formally define the distance between the agent 1 and 2 as the number of actions necessary for the agent 1 to occupy the place of the agent 2, this one not being deployed. This amounts to calculating the Manhattan distance if the agents are not allowed to use the diagonals.

Two primordial RL notions of the environment are the distance and the identification of a state. The distance is used as the reward function since it allows to measure the distance between the cat and the mouse. The identification of a state corresponds to the position of an agent (cat or mouse) on the matrix. For example in a 9x9 matrix, there are 81 states, each one corresponding to a given position.

A cat is an agent whose goal is to catch a mouse agent. Clearly, this means performing a series of actions that would lead to a final state where the cat-mouse distance would be zero. At the level of perception, a cat has a perfect acuity: at any time, he knows the position of all the agents in the space. Information on the state of the environment is therefore complete and correct. A cat can perform 5 actions which are listed below:

- Action 0: Stay still.
- Action 1: Up, move one box north.

- Action 2: Right, move forward one space to the East.
- Action 3: Bottom, advance one box to the South.
- Action 4: Left, move one space west.

Note that our cat cannot move diagonally. The mouse agent enjoys exactly the same properties as the cat: global vision of the space and the same five possible actions.

Our mouse aims to escape cats. We have decided to develop a custom algorithm called best-escape in order to perform the way the mouse is moving. The principle of the best-escape algorithm is very simple. It consists first of all in refusing any movement which would throw the mouse into the claws of a cat or any movement which would lead the mouse into the immediate neighborhood of a cat (we mean by the adjacent box). These two actions would indeed be synonymous with certain death. For each of the remaining actions, we compute the escape value, the sum of the squares of the distances of the arrival point of the action with each of the mice present in the game. The goal is obviously to maximize this value. We then choose the action giving the greatest possible escape value. If several actions have the same escape value, we take the first one from the list. The choice is therefore completely deterministic and can be predicted for each given situation. We introduced a fear factor as a variable in the mouse agent. This very simple parameter actually introduces the fact that the mouse will not perform any action at a given moment and this without regard to the rest of the environment. It can also be seen as the fact that the mouse is frozen with fear.

3.2.3 DEVSimPy Simulation

The DEVS implementation of the mouse-cat problem has been realized using the DEVSimPy framework (Capocchi, Santucci, Poggi, and Nicolai 2011). If this implementation was developed specifically for the mouse-cat problem, we wanted it to be general enough to allow it to be adapted or extended as easily as possible. The main difference between DEVS multi-agents learning and the single one is the fact that the learning agent is connected to a DEVS coupled model instead of an atomic model (Figure 4).

This coupled model embeds others agents, the global environment vision and the specific environment associated with a learning agent. The learning agent called *CatAgent* in figure 4 is connected to the coupled DEVS model (called *Environment*) embedding other agents (as *MouseAgent*), the global environment vision (as *EnvironmentMat*) and the specific environment (as *CatEnv*) associated with the learning agent (as *CatAgent*).

The global environment (*EnvironmentMat*) defines the space in which the agents will evolve. In our implementation, this space is a simple matrix of integers. 0 means an empty box. Each other integer means an agent (1 for the cat and 2 for the mouse). It is quite possible to imagine extending this environment to, for example, search in a space with 3 dimensions or more. An action is denoted by an integer and corresponds to a movement of the agent concerned. If the number of possible actions is not problematic, it is also necessary to assign for each action the corresponding movements of the concerned agent. The chosen implementation consisted of a simple dictionary assigning, for each action, the corresponding movements.

The heart of Q-Learning algorithm (Sutton and Barto 1998) is implemented in the agent (*CatAgent*) and is associated with the specific environment atomic model (*CatEnv*). From this observation, we will compute the reward to associate with the action. Both the new state and the computed reward are sent to the inputs of the *CatAgent* atomic model. The Q-Learning algorithm, assumes that *CatAgent* has received a reward and new state after doing an action.

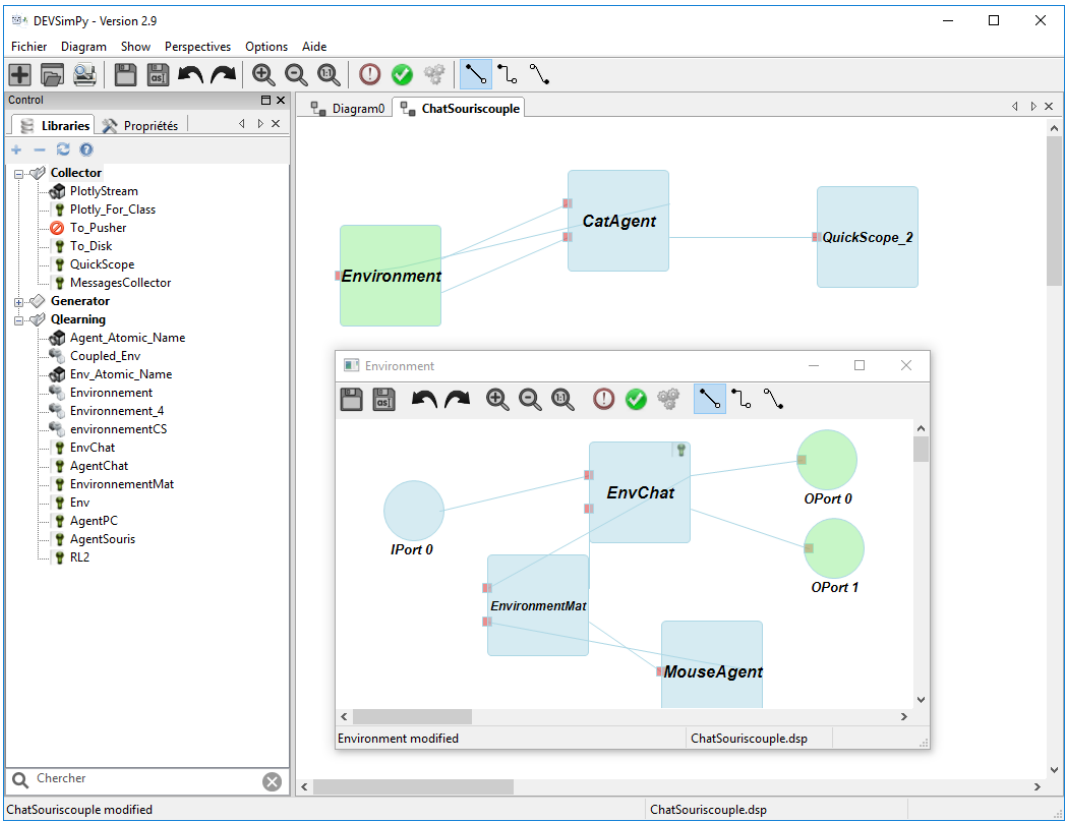

Figure 4: Mouse-cat M&S into the DEVSimPy framework.

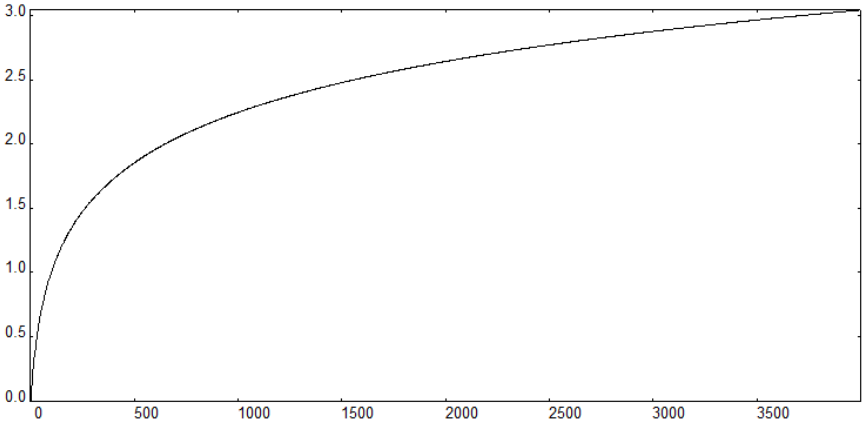

Figure 5: Q-Value mean per episode with fear factor.

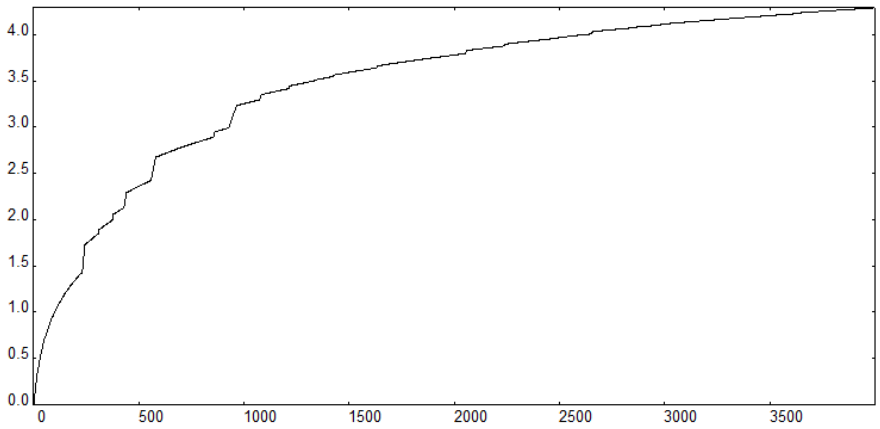

Figure 6: Q-Value mean per episode without fear factor.

The mouse agent does not use a RL technique. To choose its action, the mouse uses the Best Escape algorithm: the mouse tries to minimize the sum of the squares of the distances between itself and the cats and, if no solution is possible, the mouse remains on the spot.

The simulation has been performed using the fear factor parameter for the mouse agent: (value 0 with fear factor / value 1 without fear factor). The number of episodes needed by the Q-learning algorithm has been fixed to 4000.

Figures 5 and 6 gives the evolution of the Q-value mean during the learning phase with and without the fear factor between one mouse agent and one cat agent. One can notice that since the mouse is not able to move, the figure 5 is a regular curve. While since the mouse tries to escape when the fear factor is equal to 0, the figure 6 is not regular. These irregularity of the curve fits with the fact that the mouse is escaping.

3.2.4 Discussion

In the previous example, only one cat agent and one mouse agent are involved. However, when considering multiple cats, one of the questions is to decide if the agents will perform the learning phase collectively as a team or individually. Furthermore, when considering multiple mice, we have to decide if the goal is finally catching a mouse or all mice. These problems are general problems inherent to multi-agent learning since the research in this domain is interested in studying software agents that learn and adapt to the behavior of other software agents (Shoham, Powers, and Grenager 2003). In addition the presence of other learning agents complicates learning since the environment may become non-stationary (a situation of learning a moving target similar with the fear factor of the mouse agent). Of course the reward function will be also harder to define since it depends on both the learning agents and the multiple and moving targets.

4 CONCLUSION

The paper deals with the modeling of ML systems using discrete-event M&S features. We have detailed how discrete-event M&S aspects could be integrated into RL framework. An overview of different RL improvements have been proposed and three of them have been implemented in the framework of the DEVS formalism: (i) explicit representation of the agent and environment involved in RL (ii) introduction of temporal aspects when dealing with RL (iii) possibility to deal with multi-agents in the framework of RL.
The future work we envision concerns the following features: (i) the possibility to deal with dynamic environment by allowing the systems to dynamically add or delete agents in the environment (ii) introduction

of hierarchical learning agents in order to deal with RL according to different levels of abstraction (in the follow-up to (Kessler, Capocchi, Santucci, and Zeigler 2017)).

REFERENCES

Alpaydin, E. 2016. *Machine Learning: The New AI*. The MIT Press.

Busoniu, L., R. Babuska, and B. D. Schutter. 2006. "Multi-Agent Reinforcement Learning: A Survey". In *Ninth International Conference on Control, Automation, Robotics and Vision, ICARCV 2006, Singapore, 5-8 December 2006, Proceedings*, pp. 1–6.

Capocchi, L., J. F. Santucci, B. Poggi, and C. Nicolai. 2011. "DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems.". In *WETICE*, pp. 170–175, IEEE Computer Society.

Floyd, Michael W. and Wainer, Gabriel A. 2010, July. "Creation of DEVS Models using Imitation Learning".

Foo, N. Y., and P. Peppas. 2004. "Systems Theory: Melding the AI and Simulation Perspectives". In *Artificial Intelligence and Simulation, 13th International Conference on AI, Simulation, and Planning in High Autonomy Systems, AIS 2004, Jeju Island, Korea, October 4-6, 2004, Revised Selected Papers*, pp. 14–23.

Hespanha, J. P., H. J. Kim, and S. Sastry. 1999, Dec. "Multiple-agent probabilistic pursuit-evasion games". In *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*, Volume 3, pp. 2432–2437 vol.3.

Kessler, C., L. Capocchi, J.-F. Santucci, and B. Zeigler. 2017. "Hierarchical Markov Decision Process Based on Devs Formalism". In *Proceedings of the 2017 Winter Simulation Conference*, WSC '17, pp. 73:1–73:12. Piscataway, NJ, USA, IEEE Press.

Meraji, S., and C. Tropper. 2010, Sept. "A Machine Learning Approach for Optimizing Parallel Logic Simulation". In *2010 39th International Conference on Parallel Processing*, pp. 545–554.

Nielsen, N. R. 1991. *Application of Artificial Intelligence Techniques to Simulation*, pp. 1–19. New York, NY, Springer New York.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st ed. New York, NY, USA, John Wiley & Sons, Inc.

Rachelson, E., G. Quesnel, F. Garcia, and P. Fabiani. 2008. "A Simulation-based Approach for Solving Generalized Semi-Markov Decision Processes". In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pp. 583–587. Amsterdam, The Netherlands, The Netherlands, IOS Press.

Saadawi, H., G. Wainer, and G. Pliego. 2016, April. "DEVS execution acceleration with machine learning". In *2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS)*, pp. 1–6.

Seo, C., B. P. Zeigler, and D. Kim. 2018. "DEVS Markov Modeling and Simulation: Formal Definition and Implementation". In *Proceedings of the Theory of Modeling and Simulation Symposium*, TMS '18, pp. 1:1–1:12. San Diego, CA, USA, Society for Computer Simulation International.

Shoham, Y., R. Powers, and T. Grenager. 2003. "Multi-agent reinforcement learning: a critical survey". Technical report.

Sutton, R. S., and A. G. Barto. 1998. *Introduction to Reinforcement Learning*. 1st ed. Cambridge, MA, USA, MIT Press.

Toma, S. 2014, September. *Detection and identication methodology for multiple faults in complex systems using discrete-events and neural networks: applied to the wind turbines diagnosis*. Theses, University of Corsica.

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

Zeigler, B. P., H. S. Sarjoughian, R. Duboz, and J.-C. Souli. 2012. *Guide to Modeling and Simulation of Systems of Systems*. Springer Publishing Company, Incorporated.

AUTHOR BIOGRAPHIES

LAURENT CAPOCCHI has been an assistant professor in computer sciences at the University of Corsica (France), SPE CNRS 6134 Research Lab since 2007. His email address is capocchi@univ-corse.fr.

JEAN-FRANCOIS SANTUCCI has been a full professor in computer sciences at the University of Corsica (France), SPE CNRS 6134 Research Lab since 1995. He was assistant professor at the EERIE School, Nimes, France from 1987 to 1995. His email address is santucci@univ-corse.fr.

BERNARD P. ZEIGLER is Chief Scientist at RTSync Corp., Professor Emeritus of Electrical and Computer Engineering at the University of Arizona, Tucson and Co-Director of the Arizona Center for Integrative Modeling and Simulation. His email address is zeigler@rtsync.com.