https://doi.org/10.52326/ic-ecco.2022/CS.03

Performability Modeling of Self-Adaptive Systems Based on Extension Neural Rewriting Stochastic Petri Nets

Alexei Sclifos [1], ORCID: 0000-0003-4531-7944
Emilia Sclifos [1], ORCID:0000-0003-1986-7256
Emilian Guțuleac [1], ORCID: 0000-0001-6839-514X
[1] Technical University of Moldova, Bd. Ștefan cel Mare, 168, MD-2004, Chișinău, R. Moldova,
emilian.gutuleac@calc.utm.md, https://utm.md

***Abstract*—Traditional mathematical formalisms are unable to model modern self-adaptive discrete event systems (ADES) because they cannot handle behaviors that change at run-time in response to environmental changes. This paper introduces a new extension of Reconfigurable Stochastic reward Nets (*RSRN*), called Extension Neural Rewriting Petri Nets (ExNRPN), which enables the performability modeling and simulation of modern ADESs. ExNRPNs are obtained by incorporating in some special transitions of *RSRN*s an extension neural network (ENN) algorithm where the run-time calculation and reconfiguration is done in the local components, while the adaptation is performed for the whole system. The application of the proposed ExNRPN is illustrated by performability modeling a particular ADES.**

Keywords—adaptive system; extension neural network; performability modeling; rewriting rule; stochatic Petri net

I. INTRODUCTION

Modern dynamic discrete-event systems (DES), such as computing systems and networks; mobile dynamic Ad-hoc computer networks and many technological new solutions, based on cloud computing and Internet of Things applications, etc., must adapt in response to unpredictable changes in their states and environment to remain usefully performing. Traditionally, this adaptation has been handled during system downtime, but currently there is an increased demand to automate this process and perform it while the system is operating [1].

The concept of self-adaptive DES (ADES) was introduced as a realization of continuously adapting systems [1]. ADES are capable of changing their behavior and/or structure at run-time in response to their perception of the environment, the states of the system itself, and its requirements. To achieve this, ideally, systems should have certain adaptive characteristics known as *self* -* properties introduced in the autonomic computing paradigm [2].

The focus of this paper is on the dynamic performability modeling of ADES in the context of dynamic self-reconfiguration (SR), a key and essential property. SR of system is the ability to automatically and dynamically reconfigure itself in response to changing of states and/or environment.This may include installing, removing, and composing/decomposing elements of the system [1, 2].

There are many formalisms that can be used for this purpose such as transition systems, finite automata, process algebras [3] and different extensions of Petri nets (*PN*s) [3, 4] such as generalized stochastic *PN* (*GSPN*) and stochastic reward nets (*SRN*) [4, 5]. For example, the paper [6] presents a new learning Petri net based on artificial neural networks (ANN) for modeling adaptive software systems. Nevertheless, run-time reconfigurable *SRN* (*RSRN*) [5] seem a good candidate for performance mo-deling of ADES. However, *RSRN* lack the ability to mo-del learning and adaptation based on environmental changes with incompatible or contradictory parameters.

Next, we present a new extension of matrix hybrid *RSRN* [7], called Extension Neural Rewriting Petri Nets (ExNRPN), with adaption ability to performance modeling of ADES. It contains extension adaption transitions (AT) with learning ability based on extension neural network (ENN) [8] which describes environmental changes.

The application of the proposed ExNRPN is illustrated by performability modeling of a particular system.

II. EXTENICS THEORY FEATURES

Extenics Theory (ET) was proposed by Wen Cai [9] to solve intelligently incompatible or contradictory problems that cannot be solved by given conditions until a proper transformation of the conditions is implemented

and reformalizes the concepts to give a solution. There are similarities between Fuzzy Set Theory (FST) and ET. In standard set applications, transfer function shows if an element belongs to a class or not. FST extends this set to [0,1], showing the degree an element belongs to the class. In [9], it is explained that ET extends FST from [0,1] to $[-\infty,\infty]$ and therefore, this situation leads up with an element, belonging to each extension set to a different degree. A comparison of Crips logical, Fuzzy and Extension sets is presented in Table I.

Classical mathematics is familiar with the quantity and shapes of objects. But the *Matter-element* method (MEM) in ET considers both the quality and quantity of an object. In the real world, things, objects are represented by their quantity and quality. Therefore, MEM deals with contradictory issues of quality and quantity. ET considers the transformation of these contradictory problems into MEM models and analyzes them through their qualitative and quantitative modification.

Elements of MEM describing a problem must include the name of the problem (object), its characteristics and the value associated with that characteristic. These three basic elements constitute a problem element R, composed of N (object name), the column vectors: $\mathbf{C}=[c_1, c_2,...,c_n]$ (R features, criteria) and $\mathbf{V}=[v_1, v_2,...,v_n]$ (respective feature values) for any given problem [9]. Thus, the formula for R is: $R=[N, \mathbf{C}, \mathbf{V}]$. This triplet represents a fundamental unit to describe a multi-dimensional problem element. In addition, the $\mathbf{V}$ vector can have one or more values. In the multi-valued case, the range covered by the vector $\mathbf{V}$ is called the classical domain. In MEM, two intervals are defined $F_0 =< a, b >$ and $F =< d, e >$ with $F_0 \in F$. The problem element R_0 corresponding to it F_0 is defined as $R_0 = [F_0, \mathbf{C}_j, \mathbf{V}_j]$, where $\mathbf{C}_j = [c_{j,1}, c_{j,2},...,c_{j,n}]$ symbolizes the characteristic of F_0, and the classic domain $\mathbf{V}_j = [< a_{j,1}, b_{j,1} >, < a_{j,2}, b_{j,2} >,..., < a_{j,n}, b_{j,n} >$ corresponds to the value of $c_{j,i}$, $i=1,2,...,n$.

In the same way, the problem element R_F corresponding to F is expressed as $R_F = [F, \mathbf{C}_k, \mathbf{V}_k]$, where $\mathbf{C}_k = [c_{k,1}, c_{k,2},..., c_{k,n}]$ symbolizes the characteristic of F, and the domain $\mathbf{V}_k = [< d_{k,1}, e_{k,1} >, < d_{k,2}, e_{k,2} >,..., < d_{k,n}, e_{k,n} >]$

TABLE I THREE DIFFERENT SORTS OF MATHEMATICAL SETS

Compared item	***Crisp set***	***Fuzzy set***	***Extension set***
Research object	Data variables	Linguistic variables	Contradictory issues
Model	Mathematics model	Fuzzy Mathematics model	Matter-element model
Descriptive	Transfer	Membership	Correlation
function	function	function	function
Proprety	Precision	Ambiguity	Extension
Range of set	$C_A(x) \in (0,1)$	$\mu_A(x) \in [0,1]$	$K_A(x) \in [-\infty, +\infty]$

corresponds to the value of $c_{k,i}$, $i=1,2,...,n$.

The distance between two points in classical mathematics is extended in ET in that to calculate the distance between a point x and an interval correlation function $K(x)$ is constructed. The extension distance (ED) between an arbitrary point x and the interval between $F_0 =< a, b >$ and $F =< d, e >$, $F_0 \subset F$, is expressed as:

$$\rho(x, F_0) = | x-(a+b)/2 | -(b-a)/2, \quad (1)$$
$$\rho(x, F) = | x-(e+d)/2 | -(e-d)/2.$$

The membership degree value $D(x, F_0, F)$ between the point x and the intervals F_0 and F and the *extended correlation* function $K(x)$ are defined as:

$$D(x, F_0, F) = \begin{cases} \rho(x, F) - \rho(x, F_0), & x \notin F_0 \\ -1, & x \in F_0 \end{cases}, \quad (2)$$

$$K(x) = \begin{cases} -\rho(x, F_0), & x \in F_0 \\ \dfrac{\rho(x, F_0)}{D(x, F_0, F)}, & x \notin F \end{cases}. \quad (3)$$

As illustrated in Fig. 1, the maximum value of $K(x)$ referred to a correlation function occurs for $x \in [a, b]$. The point x is excluded from F if $K(x) < -1$, is contained in F_0 if $K(x) > 0$, and is included in the extension field if $-1 < K(x) < 0$. In addition, x can be brought in F_0 when a transformation condition is met on x in the extension field.

III. EXTENSION NEURAL NETWORK

Extension neural network (ENN) [8,10] is a new neural network type that is a combination of ET and artificial neural network (ANN). While ET makes distance measurement and extended correlation function for classification, ANN is used for its fast and adaptive learning capability.

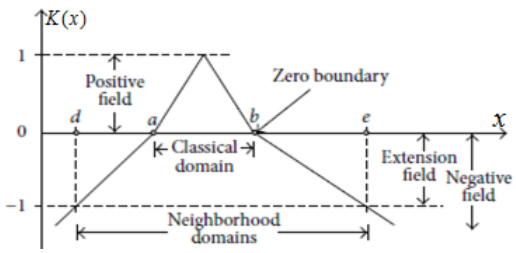

Figure 1. Extension correlation function [9]

ENN was first proposed in 2003 by M. H. Wang [8]. One of important issues in the field of classification and recognition of ENN is how to achieve the best possible classifier with a small number of labeled
training data. In [8, 10], it is shown that ENN gives better or equal accuracy and less memory consumption in classification than Multilayer Perceptron ANN, Probabilistic ANN and Counter Propagation ANN.

The structure of an ENN is presented in Fig. 2. Nodes in the output layer of ENN represent the outputs of the nodes in the input layer by a set of weights. The total number of inputs and outputs are n and n_c, respectively, and the total number of instances is N_p. The *data-points* are denoted x_{ij}^p, meaning the instance $i=1,2,\ldots,N_p$ and the characteristic value $j=1,2,\ldots,n$ correspond to the matter-element p. In ENN, x_{ij}^p becomes the input and o_{ik} the output of the node k for the instance i. Between the input x_{ij}^p and the output o_{ik} there are two sets of weights denoted w_{kj}^L and w_{kj}^U, respec-tively. These two weights are determined by searching for the lower and upper bounds of the j input of the training data. The upper bound w_{kj}^U is found by finding the maximum value for the j input node from all input instances, and the lower bound w_{kj}^L is determined inversely. These two weights are adjusted in each iteration to perform more accurate and efficient classification. Nodes o_{ik} in the output layer are the pointers to which an input vector belongs. If i instances inputs correspond to the class k, then the value o_{ik} of the output layer should be smaller than the other output nodes. This situation indicates that the distance between the i instances inputs of the class k is smaller than between the other classes. The transfer function is presented in (4), where the index of the estimated class is k^*. The weights and are the points where the extension distance $ED_{ik}(x)=1$. More details on the ED shown in (4) and the adjustment of the weights are further discussed.

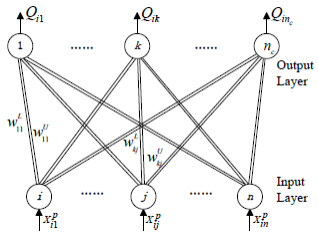

Figure 2. The structure of ENN [8]

$$ED_{ik}(x)=\sum_{j=0}^{n}\left(\frac{|x_{ij}^p-z_{kj}|-(w_{ij}^U-w_{ij}^L)/2}{|(w_{ij}^U-w_{ij}^L)/2|}+1\right), \quad (4)$$

$$o_{ik}\equiv ED_{ik}(x),\ k=1,2,\ldots,n_c.$$

ENN is a supervised learning method which provides an inferring function from supervised training data. Training data is the composition of input and desired output pairs of following *matter-element model* :

$$R_k=[class_k, c_{kj}, V_{kj}],\ k=1,2,\ldots,n_c;\ j=1,2,\ldots,n, \quad (5)$$

where $class_k$ is the name of the k class. The symbols c_{kj}, $j=1,2,\ldots,n$ represent the characteristics and V_{kj} denotes the range for the characteristic c_{kj} of $class_k$. The range value $V_{kj}=[w_{kj}^L,\ w_{kj}^U]$ is determined by w_{kj}^U and w_{kj}^L. The learning process of ENN is as follows: in the initial step, the weights are determined based on the expression (6), searching among all j instances - the maximum and minimum input of the class k to find the respective weights w_{kj}^U and w_{kj}^L.

$$w_{kj}^U=\max_{\forall i}\{x_{ij}^k\};\ w_{kj}^L=\min_{\forall i}\{x_{ij}^k\}, \quad (6)$$

$$c_{kj},\ i=1,2,\ldots,N_p,\ k=1,2,\ldots,n_c,\ j=1,2,\ldots,n.$$

After maintaining the matter-element model, the center of the clusters is determined by V_{kj} as shown in (7). We note that the clusters are the representatives of the classes. Each class has the same number of clusters as the number of inputs.

$$Z_k=\{z_{k1}, z_{k2},\ldots,z_{kn}\},\ z_{kj}=(w_{kj}^U+w_{kj}^L)/2, \quad (7)$$

$$k=1,2,\ldots,n_c,\ j=1,2,\ldots,n.$$

If these initial values are not sufficient for classification, after performing the initial steps, to obtain a more accurate classification the weights and center of the clusters will be updated. The desired classification accuracy is determined by the learning performance rate $E_r=N_m/N_p$, where N_m is the total number of instances misclassified, and where N_p is the total number of instances. The update of the cluster weights and center is continued until the learning performance rate is low enough. All instances must be used during learning. In each iteration, an instance must be randomly chosen from the training data. In (8), for training, the pattern X_i^p, whose desired result should be p is randomly chosen:

$$X_i^p=\{x_{i1}^p, x_{i2}^p,\ldots,x_{in}^p\},\quad 1\le p\le n_c. \quad (8)$$

In the next step, the ED method is used to determine the respective class based on the input vector X_i^p, whose elements are the feature values. Then the distance

between the training instances X_i^p with data-points x_{ij}^p and each cluster is calculated. According to (4), the distance between the input of the randomly taken instance and the k class is calculated. After the distance of each entry is calculated for a given class, these distances are summed to find the total distance. This procedure must be done for each class. The class that gives the minimum distance is the class that ENN classifies the current instance. However, the desired result is p. If the minimum ED shows that $k^* = p$, then no update is needed. If $k^* \neq p$, then updating is required to make a more accurate classification. If in the training phase $k^* \neq p$, the separator is shifted according to the closeness of the inputs to the cluster centers. The amount of change is directly proportional to the ED. The separator k^* of the misclassified class is displaced from that of the input instances, and the separator p of the desired class is moved next to them as formulated by the expressions (9) and (10).The cluster center and weights are both changed.

$$z_{pj}^{new} = z_{pj}^{old} + \eta \cdot (x_{ij}^p - z_{pj}^{old}), \quad (9)$$

$$z_{k^*j}^{new} = z_{k^*j}^{old} + \eta \cdot (x_{ij}^p - z_{k^*j}^{old}).$$

$$w_{pj}^{Lnew} = w_{pj}^{Lold} + \eta \cdot (x_{ij}^p - z_{pj}^{old}), \quad (10)$$

$$w_{pj}^{Unew} = w_{pj}^{Uold} + \eta \cdot (x_{ij}^p - z_{pj}^{old}),$$

$$w_{k^*j}^{Lnew} = w_{k^*j}^{Lold} + \eta \cdot (x_{ij}^p - z_{k^*j}^{old}),$$

$$w_{k^*j}^{Unew} = w_{k^*j}^{Uold} + \eta \cdot (x_{ij}^p - z_{k^*j}^{old}),$$

where η is the learning rate.

Because the ENN just adjusts the weights of the p-th and the k^*-th class, the learning of ENN has a speed advantage over the other supervised learning algorithms, and can quickly adapt to new and important information.

In ENN only one output neuron node in the n_c output layer remains active (the output value is **1**) and the output values of other neuron nodes are zero to indicate a safety status pattern of the input instance.

The distinctive characteristic of ENN is that the ENN can effectively solve the classification and recognition problems whose features are defined over an interval.

IV. Extension Neural Rewriting Petri Nets

In this section we provide a ExNRPN definition, firing rules of the respective transitions and rewriting rules by the current marking and environmental changes. We assume that the readers are familiar with the basic concepts of *RSRN* and hybrid *SRN*, called *HSRN*, with matrix attributes (*HSRNM*) [5, 7]. Next, due to the space restrictions, we will only give a brief overview to this topic. For more comprehensive details to *RSRN* and *HSRNM* we let the readers refer to [4, 5, 7].

Let IN_+ (resp. IR^+) be the set of natural (resp. positive real) numbers.

The definition of an ExNRPN is derived according to [5, 7] and inherits most of the *RSRN* and *HSRNM* characteristics. Thus, the ExNRPN, denoted $R\Gamma$, is defined as a 17-tuple system such that $R\Gamma = < P, T, R, h, A_{rcs}, Pri, G^E, G^R, K^p, \Lambda, \omega, V, \rho, M_0, Lib_R, \phi, Lib_{ENN} >$, where: $P = P^D \cup P^C$ is a finite set of places, where P^D is the set of all discrete places and P^C is the set of continuous places; $T = T^D \cup T^C \cup T^{At}$ is a finite set of transitions, where T^D, T^C and T^{At} are the sets of *discrete*, *continuous* and *adaptive* transitions, respectively; $h: P \cup T \to \{A, C, D\}$ is a mapping to assign an identifier to each node, where "*A*; *C*; *D* " indicate for every node whether it is adaptive, continuous or discrete; Let $E = T^D \cup R$ be a finite set of events, $T^D \cap R = \varnothing$, $P \cap E = \varnothing$, where R is a finite set of *rewriting rules* about the *run-time structural change* (reconfiguration) of $R\Gamma$. The set E is partitioned into $E = E_0 \cup E_\tau$, $E_0 \cap E_\tau = \varnothing$ so that: E_τ is a set of timed events and E_0 is a set of immediate events; A_{rcs} = < *Pre*, *Post*, *Inh* > is a set of *forward*, *backward* and *inhibition* arcs functions, that describes the respectively arcs with marking-dependent weight cardinalities; *Pri* defines the dynamic marking-dependent priority function for the *firing* of each *enabled* $e \in E$. The firing of an enabled event with higher priority potentially disables all events $e \in E$ with the lower priority. By default, the *Pri*(E_0)>*Pri*(E_τ); $G^E : E \times IN_+^{|P|} \to \{True, False\}$ is the set of *guard function* associated with all event $e \in E$ and $G^R : R \times IN_+^{|P|} \to \{True, False\}$ is the set of *guard function* associated with all *rewriting rule* $r \in R$; $K^p : P \times IN_+^{|P|} \to IN_+ \cup \{\infty\}$ is the capacity bound of each place $p_i \in P$, which can contain an *integer* or real number of *tokens* [5]. By default, K_i^P it is $+\infty$; M_0 is the initial marking; $\tilde{\Lambda} : E_\tau \times IN_+^{|P|} \to IR^+$ is the function that determines the firing rate $0 < \tilde{\lambda}(e, M) < +\infty$ of timed event $e \in E_\tau$, that is *enabled* by current marking M; $\omega : E_0 \times IN_+^{|P|} \to IR^+$ is the weight function $0 \le \omega(e, M) < +\infty$ which determines the firing probability $q(t, M)$ of immediate event $e \in E_0$, therein describes a probabilistic selector; $V : T^C \times IN_+^{|P^A|} \to IR^+$ is the marking dependent fluid rate function of T^C. . If

$u_j \in T^C$ is enabled in *tangible* marking M it fires with rate $V_j(M)$, that continuously changes the fluid level of continuous place $b \in P^C$; $\tilde{\rho}: P \cup E \to IR^+$ is the function that determines the *reward rates* (real numbers) assigned to each current marking M and to each firing event $e \in E$; Lib_R is the set of $R\Gamma_v$, $v = 1, 2, \cdots, n_v$ *subnets* and parameters pattern class library involved in structural reconfiguration of the current $R\Gamma$ configuration by firing of an enabled rewriting rule $r \in R$; $\phi: Lib_{ENN} \to T^{At}$ is a mapping to assign a ENN to each *adaptive transition*, where Lib_{ENN} is the set of all ENNs;

Fig. 3 summarizes the graphical representation *HSRN* primitive elements of $R\Gamma$.

Enabling and firing rules of events $e \in E$ and continuous transitions *by current marking* M in $R\Gamma$ are the same as for RSRN and HSRNM [5, 7].

To develop and present more compact $R\Gamma$ models we will use the approaches presented in [5, 7, 11]. We present here only the most important features of Matrix Transi-tion Net (MTN) [11] wthere *matrices* are used instead of places (transitions) and other attributes as in PNs and SRNs. MTD models are more expressive and compact than PNs and capture a greater amount of information. Removal (placement) of tokens in MTN is a simple sub-traction (addition) of the input (output) matrix from the input (output) function matrix. The result should be the updated input matrix (output) after transition firing, i.e. this is the next marking of that matrix.

Graphically, a matrix attributes of $R\Gamma$ models will be presented in a way that its will contain in square brackets the matrix name [7]. So, for example, a direct arc matrix cardinality, denoted by  , can take values that are contained in a specified matrix **A**.

An adaptive transition, called **At** - transition, is a transition associated with a respective learned ENN.

Fig. 4 shows an **At** – transition, At_j, that corresponds to the ENN in Fig. 2. The inputs of an At - transition are presented by Matrix C-place $[\mathbf{B}_j]_{n \times 1}$ that determine the respective inputs of the ENN, while the marking of the output D – place p_k represent the order number of the k^*-th class selected by the ENN function .

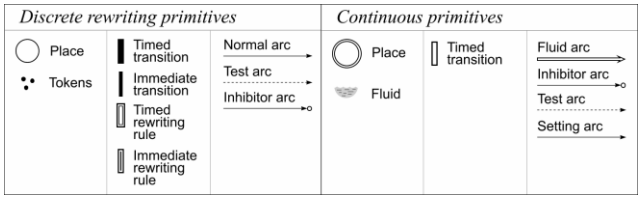

Figure 3. The graphical primitive of the *HSRN*

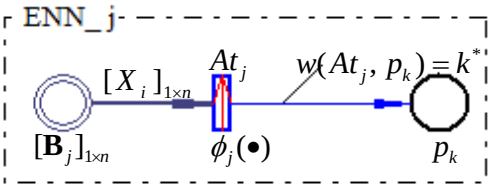

Figure 4. An **At**-transition representing the ENN in figure 2

If the outputs of two or more **At**-transitions can eventually access the same place, they operate independently. This structure can be used to model two components that separately and independently complete adaptation. But since their outputs eventually reach the same D-transition, they are dependent on each other. This structure is used to model that several components are combined together to complete the adaptation. **At** - transitions have lower priorities than C-transitions and D-transitions in a conflict.

V. PERFORMABILITY MODELING CASE STUDY

In this section, we will illustrate the application of the ExNRPN to performability modeling of a particular DES, for exemple, steam turbine generator [12], manufacturing system [6]. The concept of performability emerged from a need to assess a system's ability to perform when performance degrades as a consequence of faults.

In Fig. 5 is presented the FMSRN1 model, denoted $R\Gamma 1$, which describes the behavior, on-line ENN fault diagnosis [6, 10] and recovery of a particular ADES. *The meanings of places and transitions in* $R\Gamma 1$ *model*:

- **Places.** $p1$ - initiation of the maintenance period; $p2$ - operation of the system during the diagnosis; $p3$ - reliable operation of the system; $p4$ - setting the generation of the current values of the environment pa-

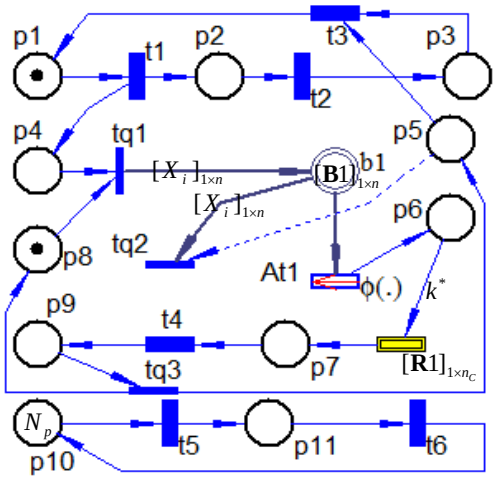

Figure 5. $R\Gamma 1$ performability model of steam turbine generator

rameters; $p5, p9$ - the end of the diagnosed fault recovery; $p6$ - indicator of the k^* type of diagnosed fault; $p7$ - initiating the recovery of the system in reliable state; $p8$ - ENN is ready to diagnose the system; $p10$ - indicate the total number $M_0(p10) = N_p$ of instances environment space parameters; $p11$ - control place that generates the current environment instance.

- **Timed transitions.** $t1$ - system maintenance time period; $t2$ - the duration of the system's operation during the diagnostics; $t3$ - reliable system operation time; $t4$ - system recovery time; $t5$ - generation time of a new environment parameter instance; $t6$ - the current instance time of the environment.
- **Immediate transitions.** $tq1$ - generation of the environment current instance; $tq2$ - elimination of the markings in matrix C-place $b1$; $tq3$ - reseting of ENN.

There are N_p environmental instances X_i that are represented in a matrix $\mathbf{X}$. Each instance is an line - vector $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$ and it can be randomly chosen from the matrix $\mathbf{X}$ using the control place $p11$, such that $i = M(p11)$ specify the index row of $\mathbf{X}$.

The line-vector $\mathbf{R}1 = (r_1, r_2, \ldots, r_{n_C})$ describes all the rewriting rules for on-line reconfiguration of $R\Gamma 1$ model. With each rewriting rule $r_j \in \mathbf{R}1$ is associated the respective type of diagnosed fault F_j, i.e. F_j involves enabling an firing of r_j. Thus, the selection of r_j to be enabled and fired depends on the current marking of the *control place* $p6$, i.e. $j = M(p6) > 0$ specify the index of r_j.

The guard functions of rewriting rule r_j in $R\Gamma 1$ are:

$g^E(r_j) := (M(p6 > 0);\ g^R(r_j) := "True",\ j = 1, 2, \ldots, n_c.$

Generation of environment parameters: if transition $tq1$ is enabled, its firing generates a line-vector quantity, written as $X_i = (x_{i,1}, x_{i,1}, \ldots, x_{i,1})$, where each component is a positive real number. The reason is that if the system captures the change of the environment, that is reflected in the line-vector cardinalite of directed arc $(tq1, b1)$. The weights can be adjusted during the training process of a ENN1, which adds the adaptation ability to the $R\Gamma 1$.

A dynamic reconfiguration of $R\Gamma 1$ by the firing of enabled $r_j \in R$ is a map $r_j : R\Gamma_L \triangleright R\Gamma_j^w$, The operator $\triangleright$ represents a binary rewriting operation which produces a *structural change* in $R\Gamma 1$ by *replacing* the current subnet $R\Gamma_L \subseteq R\Gamma 1$ ($R\Gamma_L$ is dissolved) and a new $R\Gamma_j^w \in Lib_R$ subnet is *added* and belongs to the new modified resulting underlying net $R\Gamma 1' = (R\Gamma 1 \setminus RN_L) \cup R\Gamma_j^w$, where the meaning of \ (and $\cup$) is operation of removing (adding) RN_L from ($R\Gamma_j^w$ to) $R\Gamma 1$. As exemple, in our case $R\Gamma_L := t_4$. For more detail to using of the $\triangleright$ see [5].

Performability analysis of $R\Gamma 1$ model can be performed following the approach described in [5, 7].

In a future work, we will focus on developing a visual simulator software system incorporate into VRPN ToolBox [13] with a friendly interface for checking behavioral properties and performability analysis of $R\Gamma$ models that involve other kinds of law time distributions fuzzy parameters of transition and firing rewriting rules.

REFERENCES

[1] C. Krupitzer et al., "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing* 17, pp. 184–206, 2015.

[2] O. Kephart, D. M. Chess, "The vision of autonomic computing," *Computer*, 36.1 pp. 41–50, 2003.

[3] D. Weyns, M. Iftikhar, D. Iglesia, T. Ahmad, "A survey of formal methods in self-adaptive systems," *Proceedings C3S2E*, Montreal, QC, Canada, pp. 67–79, 2012.

[4] G. Chiola, M. Ajmone- Marsan, G. Balbo, G. Conte, "Generalized stochastic Petri nets: A definition at the net level and its implications," *IEEE Transactions on Software Engineering*, 19 (2), pp. 89-107, 1993.

[5] V. Moraru, E. Guţuleac, S. Zaporojan, "Uncertainty modelling of dynamically reconfigurable systems based on rewriting stochastic reward nets with z-fuzzy parameters," *Computer Science Journal of Moldova*, Vol.29, No.3(87), pp. 388-406, 2021.

[6] Z. Ding, Y. Zhou, M. Zhou, Modeling Self-Adaptive Software Systems with Learning Petri Nets," *IEEE Transactions on Systems, MAN, and Cybernetics: Systems*, Vol. 46, No. 4, pp. 483-498, 2016.

[7] E. Guţuleac, S. Zaporojan, I. Gîrleanu, V. Cărbune, "Hybrid stochastic Petri nets with matrix attributes for modelling of discrete-continuous process," *Meridian Ingineresc*, No. 2, pp. 34–40, 2016.

[8] M.H. Wang, C P. Hung, "Extension neural network and its applications," *NeuralNetw. Off. J. Int. Neural Netw. Soc.* **16** (5–6), pp. 779–784, 2003.

[9] W. Cai, "The extension set and incompatibility problem," *J. Science Exploration* 3(1), pp. 81–93, 1983.

[10] M. H. Wang,"Extension Neural Network-Type 2 and Its Applications," IEEE Transactions on Neural Networks, Vol. 16, No. 6, pp. 1352-1361, 2005.

[11] A. S. Staines, F. Neri, "A Matrix Transition Oriented Net for Modeling Distributed Complex Computer and Communication Systems," *WSEAS Transact. on Systems*, Vol. 13, pp. 12-22, 2014.

[12] M. H. Wang, "Application of extension theory to vibration fault diagnosis of generator sets," *IEE Proc.-Gener. Transm. Distrib.*, Vol. 151, No. 4, pp. 503-508, 2004.

[13] Iu. Ţurcanu, E. Guţuleac, Em. E. Guţuleac, "VRPN-software tool for the modeling of marked-controlled rewriting generalized Petri nets," *Proc. of the 5-th International Conference on ICMCS- 2007*, Vol. 1, pp. 239-245, 2007.