VRDN: A Software Environment for Visual Simulation of Rewriting Generalized Timed Differential Petri Nets Models

Emilian GUŢULEAC, Iurie ŢURCANU, Ion BALMUŞ, Victor CHEIBAŞ, Alexei CORDUNENU
Computer Science Department, Technical University of Moldova
Bul. Stefan cel Mare nr. 168, MD-2012 Chisinau, Republic of Moldova
egutuleac@mail.utm.md, Iurie.Turcanu@endava.com, balmus@mail.utm.md

***Abstract* — This paper presents the Rewriting Generalized Differential Petri Nets (RGDN) a class of Petri nets, that accept the negative-continuous place capacity, negative real values for discrete and continuous place marking and negative marked-dependent arc cardinalities. For the purpose of visual simulation and analysis of RGDN a Visual Rewriting Differential Petri Nets (VRDN) software environment has been elaborated and developed. It offers an intuitive graphical user interface for designing various elements of nets as well as their efficient simulation, thus making it usable for research and academic activities**.

***Index Terms* — hybrid systems, modeling, Petri nets, rewriting, software tool, visual simulation**

I. INTRODUCTION

Discrete-continuous modelling is concerned with the description, analysis and performance evaluation the dynamic behaviour of hybrid systems [3, 4, 6, 16, 17]. Among the most popular formalisms that are used, there are Timed Hybrid Petri nets (*HPN*) [1] models in which some places may hold a discrete number of tokens while others contain a continuous quantity represented by real numbers.

A number of different extensions of *HPN* have been proposed [2, 6, 10, 17]. In [7, 9] we have introduced the Generalized Differential Petri Nets (*GDPN*), which are suitable to represent the behaviour of hybrid systems in a common model. The features of *GDPN* accept negative real values for continuous place marking, place capacity and negative token-dependent arc cardinalities that permit to generalize the concept of *HPN*.

To our knowledge, existing modeling methods do not support dynamic reconfiguration modelling and simulation of hybrid systems.

In [7] there is an introduction to *rewriting GDPN* (*RGDN*) that can dynamically modify their own structures, and namely some of their components, using rewriting rules, thus supporting structural dynamic changes within modelled systems [12, 16].

Although their structures are simple, analysis of *HPN* is a very complicated thing. There are several research groups all around the world working on analysis of Petri nets, and certainly there is a whole bunch of modelling software packages [3, 13, 14]. Most of the tools provide a graphical user interface for the convenient drawing and editing of the model. They usually include analysis components for one or several classes of models. But still there is not such an application that would satisfy everyone's specific needs, both commercial and educational, so we decided to start developing our own software for modeling *GDPN*, and called it the Visual Rewriting Differential Petri Nets (*VRDN*).

The paper is organized as follows. In Section 2 and in Section 3 we describe the model definition of *GDPN* and *RGDN*, respectively. Subsequently in Section 4 we considered the *VRDN* tool overview. Visual simulation facility of timed *RGDN* in *VRDN* is presented in Section 5. Conclusions are drawn in Section 6.

II. GENERALIZED DIFFERENTIAL PETRI NETS

The definition of *GDPN* with is derived from [1, 2, 10] and presented in [7].

A *GDPN* is a 14-tuple $H\Gamma = < P, T, Pre, Post, Test, Inh, K_p, K_b, G, Pri, M_0, \theta, W, V >$, where:

- P is the finite set of places partitioned into a set of discrete places P_D, and a set of continuous places P_C, $P=P_D \cup P_C$, $P_D \cap P_C = \varnothing$. The P_D may contain a natural number of tokens, while the marking of a P_C is a real number (fluid level).
- T is a finite set of transitions, $T \cap P = \varnothing$, that is partitioned into a set T_D of discrete transitions and a set T_C of continuous transitions, that $T = T_D \cup T_C$, $T_D \cap T_C = \varnothing$.
- *Pre*, *Test* and $Inh: P \times T \to Bag(P)$ respectively are a forward flow, test and inhibition functions. $Bag(P)$ is a discrete or continuous multiset over *P*. The backward flow function in the multisets of *P* is a $Post: T \times P \to Bag(P)$, where define the set of arcs *A* and it describes the marking-dependent cardinality of arcs connecting transitions with places and vice-versa.
- $0 \le K_{p_i}^{\min} < +\infty < K_{p_i}^{\max}$ which can contain an *integer* number of *tokens*, respectively. By default $K_{p_i}^{\min} = 0$ and $K_{p_i}^{\max}$ being infinite value;
- The $K_b : P_C \to IR$ is the function-capacity of continuous places and for each $p_i \in P_C$ describes the fluid lower bound x_i^{min} and upper bounds x_i^{max} of fluid on each continuous place, that $-\infty < x_i^{\min} < x_i^{\max} < +\infty$. This x_i^{max} by default it is ∞, and bound has no effect when it is set to infinity. Continuous place has an implicit lower bound at level is 0;

The IN_+ and IR are the sets of discrete natural and real numbers, respectively.

- $G: T \times Bag(P) \to \{True, False\}$ is the *guard function* defined for each transition. For $t \in T$ a guard function $g(t, M)$

will be evaluated in each marking *M*, and if it evaluates to *true*, the transition may be enabled, otherwise *t* is disabled (the default value is *true*);

- *Pri*: $T_D \rightarrow IN_+$ defines the priority functions for the firing of each transition. The enabling of a transition with higher priority disables all the lower priority transitions;
- The current marking (state) value of a net depends on the kind of place, and it is described by a pair of vector-columns $M = (\boldsymbol{m}, \boldsymbol{x})$, where $\boldsymbol{m}$: $P_D \rightarrow IN_+$ and $\boldsymbol{x}$: $P_C \rightarrow IR$ are the marking functions of respective type of places.

$\boldsymbol{m} = (m_i p_i,\ m_i \geq 0, \forall\ p_i \in P_D)$ with $m_i p_i$ describe the number $m_i = \boldsymbol{m}(p_i)$ of tokens in discrete place p_i, and it is represented by black dots.

$\boldsymbol{x} = (x_k b_k,\ x_k \geq x_k^{\min}, \forall b_k \in P_C)$ with $x_k b_k$ describe the fluid level $x_k = \boldsymbol{x}(b_k)$ in continuous place b_k and it is a real number, also allowed to take *negative* real value. The initial marking of net is $M_0 = (\boldsymbol{m}_0, \boldsymbol{x}_0)$. Vector-columns $\boldsymbol{m}_0$ and $\boldsymbol{x}_0$ give the initial marking of discrete places and of continuous places, respectively;

- The set of discrete transitions T_D is partitioned into $T_D = T_\tau \cup T_i$, $T_\tau \cap T_i = \varnothing$ so that: T_τ is a set of timed discrete transitions and T_i is a set of immediate discrete transitions. The $Pri(T_i) > Pri(T_\tau)$.

Let $T(M)$ denote the set of enabled transitions in current marking $M = (\boldsymbol{m}, \boldsymbol{x})$.

Figure 1 summarizes the graphical representation of all the *GDPN* primitives.

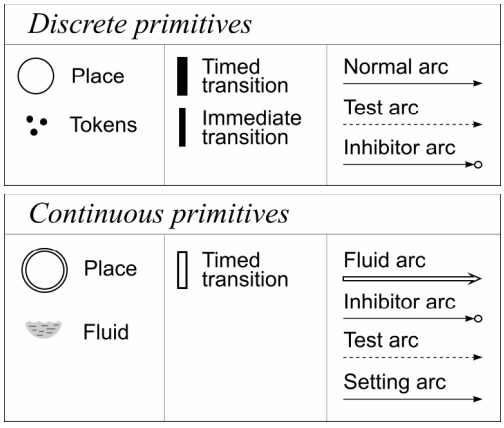

Figure 1. All the primitive of the *GDPN* .

A timed discrete transition $t \in T_\tau$ is drawn as a black rectangle and has a firing delay $\theta : T_\tau \times Bag(P) \rightarrow IR_+$ is associated to it, and this is can be marking dependent. Thus, a timed transitions $t \in T_\tau(M)$ is enabled in current tangible marking M, it fires after delay $\theta(t, M)$. Note once again, we do allow the firing delay to be dependent on fluid levels;

- $W : T_i \times Bag(P) \rightarrow IR_+$ is the weight function of immediate discrete transitions $t_j \in T_i$, and this type of transition is drawn with a black thin bar and has a zero constant firing time. If several enabled immediate transitions $t_j \in T_i(M)$ are scheduled to fire at the same time in *vanishing* marking *M*, the transitions t_k with the respective weights w_k fire with probability:

$$q(t_k, M) = w(t_k, M) / \sum_{t_j \in T_i(M)} w(t_j, M);$$

- $V : T_C \times Bag(P) \rightarrow IR_+$ is the marking dependent fluid rate function of timed continuous transitions T_c. These rates appear as labels next to the continuous timed transitions. If $t_i \in T_c$ is enabled in *tangible* marking *M* it fires with rate $V_i(M)$, that continuously change the fluid level of continuous place P_C.

Figure 2 summarizes the all possible ways of placing arcs in a *GDPN* net for discrete transition and continuous transition with the discrete places and continuous places, respectively.

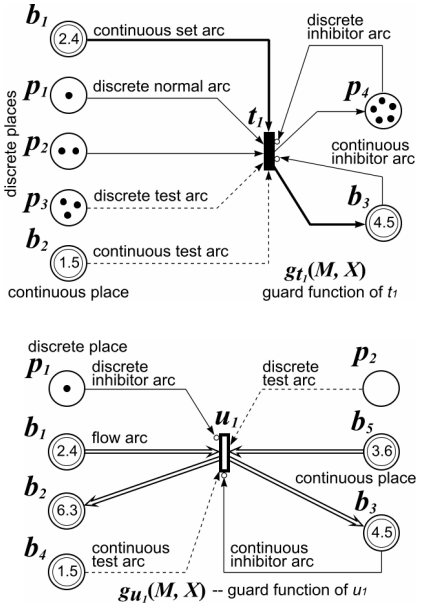

Figure 2. All the possible ways of placing arcs in a *GDPN*.

In the *GDPN*, the potential rate $\beta_i(M)$ of change of fluid level in place $p_i \in P_C$ in marking current *M* is given by:

$$\beta_i(M) = \sum_{t_k \in T(M)} [V_{k,i}(M) - V_{i,k}(M)],$$

where for any given $t_k \in T_C$, $V_{i,k}(M)$ is an input fluid rate of fluid place $p_i \in P_C$ and $V_{i,k}(M)$ is an output fluid rate of this place. We allow the firing rates and the enabling functions of the timed discrete transitions, the firing speeds and enabling functions of the timed continuous transitions, and arc cardinalities to be dependent on the current state of the *GDPN*, as defined by the marking $M(\tau)$.

The enabling and firing transitions of *GDPN* is described in [9, 10].

In the following, we describe the *dynamic rewriting RGDN systems* introduced in [7].

III. DYNAMIC REWRITING GDPN SYSTEMS

Let $X \rho Y$ be a binary relation. The *domain* of ρ is the $Dom(\rho) = \rho Y$ and the *codomain* of ρ is the $Cod(\rho) = X\rho$. Also, let $A = < Pre, Post, Test, Inh >$ be a set of arcs belonging to net $H\Gamma = < P, T, Pre, Post, Test, Inh, K_p, K_b, G, Pri >$.

A *dynamic rewriting GDPN* system (*RGDN*) is a $RH = < H\Gamma, R, \phi, G_{tr}, G_r, M >$, where:

- $N = < H\Gamma, \theta, W, V >$ and $R = \{r_1, ..., r_k\}$ is a finite set of discrete rewriting rules (DR) about the run-time structural modification of a net, so that $P \cap T \cap R = \varnothing$. In the graphical representation, the DR rule is drawn as two embedded empty rectangles;
- $\phi : E \to \{T_D, R\}$ is a function which indicates for every rewriting rule the type of event that can occur and $E = T_D \cup R$ denote the set of *events* of the net;
- $G_{tr} : R \times Bag(P) \to \{true, false\}$ is the *transition rule guard function* associated with $r \in R$, and $G_r : R \times Bag(P) \to \{true, false\}$ is the *rewriting rule guard function* defined for each rule of $r \in R$, respectively. For $\forall r \in R$, the function $g_{tr}(M) \in G_{tr}$ and $g_r(M) \in G_r$ will be evaluated in each marking and if they are evaluated to *true*, then the rewriting rule *r* may be *enabled*, otherwise it is *disabled*. In current marking *M* the default value of $g_{tr}(M) \in G_{tr}$ is *true* and for $g_r(M) \in G_r$ is *false,*

Let $R\Gamma = < H\Gamma, R, \phi, G_{tr}\ G_r >$ and $RN = < R\Gamma, M >$. A dynamic rewriting structure modifying rule $r \in R$ of *RN* is a map $r : RN_L \triangleright RN_W$, where the *codomain* of the $\triangleright$ *rewriting operator* is a fixed subnet RN_L of current net *RN*, where the *domain* of the $\triangleright$ is a new RN_W subnet. The rewriting operator $\triangleright$ represents the binary operation which produces a *structure change* in the net *RN* by replacing (rewriting) the fixed current subnet RN_L (RN_L are dissolved) with the new subnet RN_W, now belonging to the new modified resulting net $RN' = (R\,N \setminus RN_L) \cup RN_W$ where the meaning of \ (and $\cup$) is operation of removing (adding) RN_L from (RN_W to) the net *RN*. In this new net RN', obtained by firing of enabled rewriting rule $r \in R$, the places and events with the same attributes which belong to RN' are fused.

A state configuration of a net *RN* is a pair $(R\Gamma, s)$, where $R\Gamma$ is the current structure of net together with a current state $s = (M,\ \beta(M))$. The $(R\Gamma_0, s_0)$ is the initial configuration of a *RN*.

Figure 3 summarizes the graphical representation of all the *RGDN* primitives.

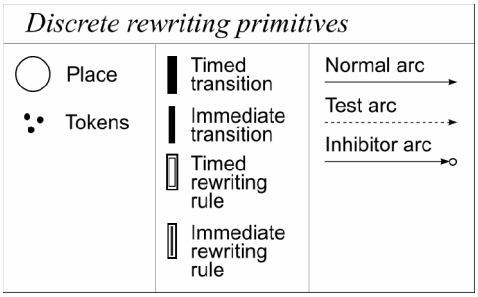

Figure 3. All the discrete rewriting primitive of the *RGDN.*

The enabling of events depends on the marking of all places. We say that a transition $t_j \in T_D$ of the event e_j is enabled in current marking *M* if the enabling condition $ec_d(e_j, M)$ and is verified. This is described in [6].

The discrete rewriting rule $r_j \in R$, that changes the structure of *RN*, is enabled in current marking *M* if the $ec_d(e_j, M)$ and the $g_{tr}(r_j, M)$ are verified.

Let $T_D(M)$ and $R(M)$, $T_D(M) \cap R(M) = \varnothing$, be the sets of enabled discrete transitions and enabled rewriting rule in current marking *M*, respectively. We denote the set of enabled events in a current marking *M* by the $E(M) = T_D(M) \cup R(M)$.

The event $e_j \in E(M)$ fires if no other event $e_k \in E(M)$ with higher priority is enabled. Hence, for the each event e_j ***if*** $((\phi_j = t_j) \vee (\phi_j = r_j) \wedge (g_{tr}(r_j, M) = False))$ ***then*** the firing of transition $t_j \in T_D(M)$ or rewriting rule $r_j \in R(M)$ changes only the current marking:

$(R\Gamma, s) \xrightarrow{e_j} (R\Gamma, s') \Leftrightarrow (R\Gamma = R\Gamma$ and in $R\Gamma$ the $M[e_j > M')$. Also, for event e_j ***if*** $((\phi_j = r_j) \wedge (g_r(r_j, M) = True))$ ***then*** the event e_j occurs to firing of rewriting rule r_j and it changes the configuration and marking of the current net in the following way:

$(R\Gamma, s) \xrightarrow{r_j} (R\Gamma', s')$, $M[r_j > M'$.

The state graph of a $RN = < R\Gamma, M >$ net is the labeled directed graph whose nodes are the states and whose arcs which are labeled with events or rewriting rules of RN :

a) *firing* of an enabled event $e_j \in E(M)$ determines an arc from the state $(R\Gamma, s)$ to the state $(R\Gamma, s')$ which is labeled with event e_j when this event can fire in the net configuration $R\Gamma$ at marking *M* and leads to a new state:

s': $(R\Gamma, s) \xrightarrow{e_j} (R\Gamma', s') \Leftrightarrow$

$(R\Gamma = R\Gamma'$ and $M[e_j > M'$ in $R\Gamma)$;

b) *change configuration*: arcs from state $(R\Gamma, s)$ to state $(R\Gamma', s')$ labeled with the rewriting rule $r_j \in R$, so that $r_j : (R\Gamma_L, M_L) \triangleright (R\Gamma_W, M_W)$ which represent the change configuration of current *RN* net: $(R\Gamma, s) \xrightarrow{r_j} (R\Gamma', s')$ with $M[r_j > M'$.

In the following, we describe the *VRDN* environment for visual simulation of *GDPN* models.

IV. VRDN SOFTWARE ENVIRONEMENT

The *VRDN* is a software environment for construction, editing, visual simulation and performance analysis of *RGDN*s models.

It is a desktop application developed on .Net platform in C# using Microsoft Visual Studio.

The architecture of the *VRDN* tool is based on MVC pattern (Fig. 4) [12].

The general idea of this approach is to have a model which represents the single point of truth. The state of this model is reflected by a series of views registered in the model and updated when an event is raised within the model. The model is controlled by a series of controllers, and as a result it changes its state. Finally the user is

provided with a set of views, in which he or she could see the evolution of the model and a set of controller which allows him or her to interact with the model.

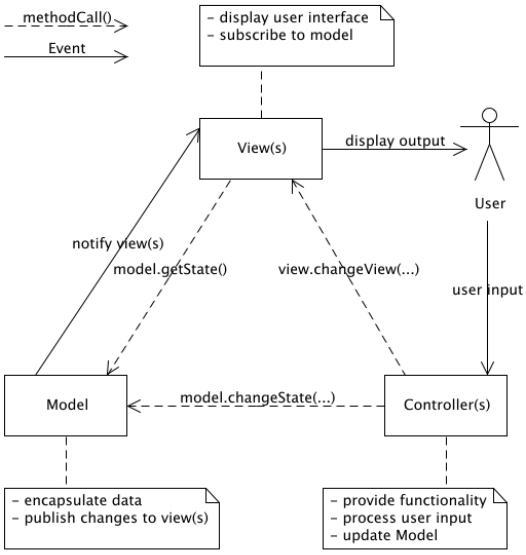

Figure 4. The architecture of the *VRDN* tool.

This tool consists of a Graphical User Interface (GUI), an animation modeling and a visualization component (see Fig. 5). The *VRDN* allows us to create, save and load *RGDN* nets in compliance to the last XML-based standard for Petri nets, namely *PNML* (*Petri Net Markup Language*) [15].

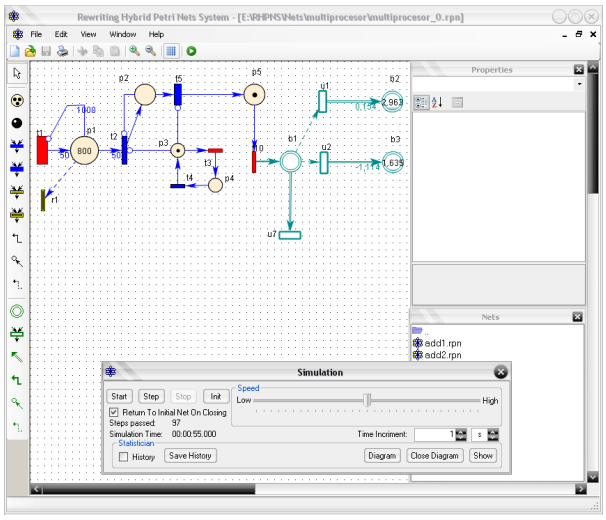

Figure 5. Graphical User Interface of *VRDN*.

The main features of the interface of the *VRDN* environment are presented below.

Firstly, since the GUI is designed with multi-document capability, it allows for several nets to be edited simultaneously. This allows the user to study the behavior of the net by observing the token animation.

Secondly, the interface is composed in a very intuitive way and includes a menu bar, specialized tool bars, context menus and keyboard shortcuts (Fig. 6). Also, the *VRDN* tool has printing and exporting capabilities, thus the results of the net definitions could be exported in most of the common graphical formats.

Constructing *RGDN* models consists of drawing actual net elements (places, transitions, arcs) according to the rules on a working area, named drawing board. The board contains a grid which facilitates a better elements alignment. For a better readability arcs could contain several segments with moveable handler.

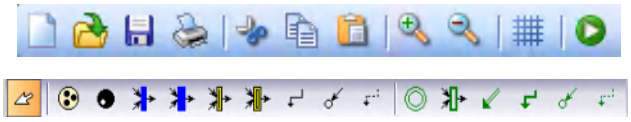

Figure 6. General actions and *VRDN* elements toolbars.

Also, for a better alignment and readability all objects placed on the drawing area are moveable by simply pointing to them and dragging them to the new desired position. Often it is handy to operate with multiple net elements at once. For this purposes *VRDN* offers multiple selection and grouping features. When working with groups, the transformation applied to the group is subsequently applied to all members of the group.

A great addition to the GUI of the *VRDN* tool is the Properties Inspector. It is used to adjust properties of the selected elements of the net. It contains several areas, including an area which explains the meaning of the property being edited. Editing of complex properties, such as Guard Function, implies opening of specialized editors such as Formula Editor (Fig. 7).

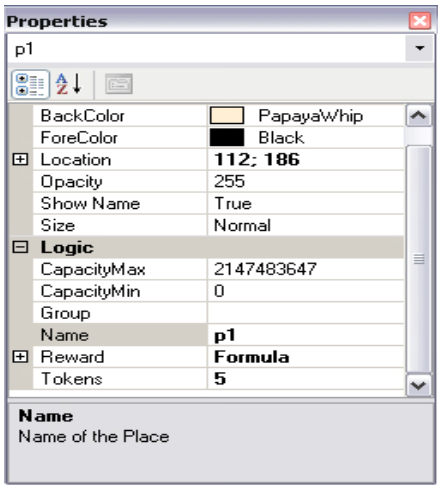

Figure 7. Properties Inspector.

For design of auto-modifying nets, *VRDN* has Formula Editor – an instrument to set values of properties which are marking dependant (Fig. 8).

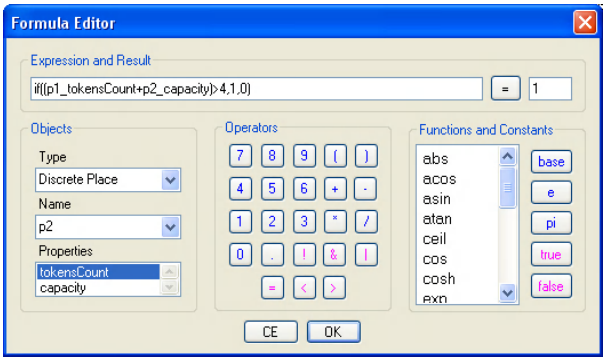

Figure 8. Formula Editor.

Formula Editor allows building complex arithmetic and logical expressions based on various properties of objects in the net. These objects as well as their types and properties are listed in the left part of the editor. Central and right part of the editor's window allows building of arithmetic and logic expressions from a set of operators, functions and constants.

Final expression is displayed at the top of the windows, which also allows instant evaluation of the expression. In case the evaluation fails due to an error, the application displays relevant error information.

VRDN has convenient navigation mechanisms, including zooming and panning for efficient working with large nets.

V. VISUAL SIMULATION OF TIMED RGDN

The simulation is the primary goal of the modeling *RGDN* nets with *VRDN*. At the high level it is done in three steps. It starts with constructing the model on the screen using the whole set of features provided by the *VRDN* graphical user interface. Then the model is validated to be syntactically and semantically correct. At the third step the proper simulation is performed and statistical data is collected during the simulation cycle. This data is later analyzed and interpreted, thus allowing predicting the behavior of the actual modeled system.

Simulation feature of the *VRDN* tool consist of a simulation form (Fig. 9), which contains all necessary controls to debug nets in both automatic and step-by-step modes.

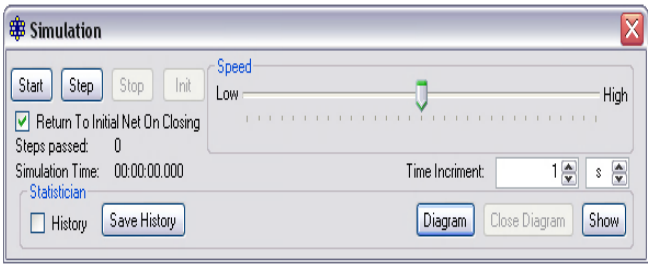

Figure 9. Simulation control form.

Simulation begins when the Start button is clicked. During the simulation process, all the results are reflected in the drawing board.

An interesting case is the simulation of *VRDN* with rewriting transitions. All rewriting rules point to two nets: 1) the old one which will be replaced, and 2) the new one, which will replace all elements with the same identity. *VRDN* looks into the working folder of the net, finds relevant nets at run-time and loads them according to the rules.

Below is an example of an effect of such a rule. Suppose we have to develop the rewriting net presented in Fig. 10.

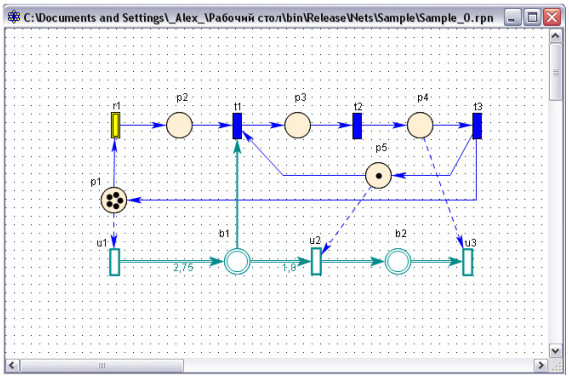

Figure 10. Initial rewriting net *RPDR*1.

The rewriting net should have a rewriting rule with a corresponding guard function. Along with guard function we have to set at least one of sub-nets. For the case when the rule defines only the old sub-net, then the resulting net will degrade. When only new sub-net is specified, then the resulting net will evolve. In the case when both sub-nets are specified, the new sub-net replaces the old sub-net. Both these sub-nets are defined in the net working folder and are loaded dynamically at run-time.

The newly loaded rewriting net could also contain one or many rewriting rules. In this case the process will continue until the new rewriting rule is fired and so on and so forth.

Thus, after firing the rewriting rule $r1: RPDR1 \triangleright RPDR2$ for this particular case at the next simulation step we may have the result presented in the Fig.11.

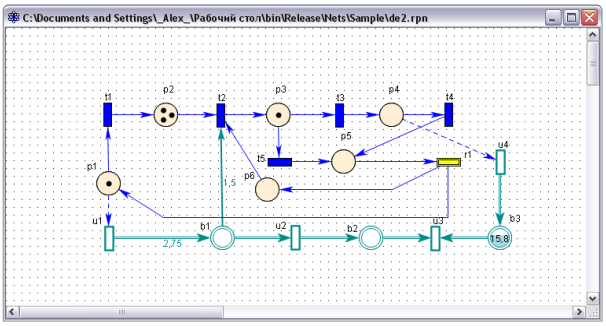

Figure 11. Rewriting net *RPDR*2, as a result of applying rewriting rule.

During simulation it is possible to collect the intermediate information, which can later be used for further analysis.

The simulation history (Fig.12) contains the results of the simulation process at each individual step and namely contains total simulation time, current marking, enabled and fired immediate and timed transitions as well as elapsed time for firing of timed transitions.

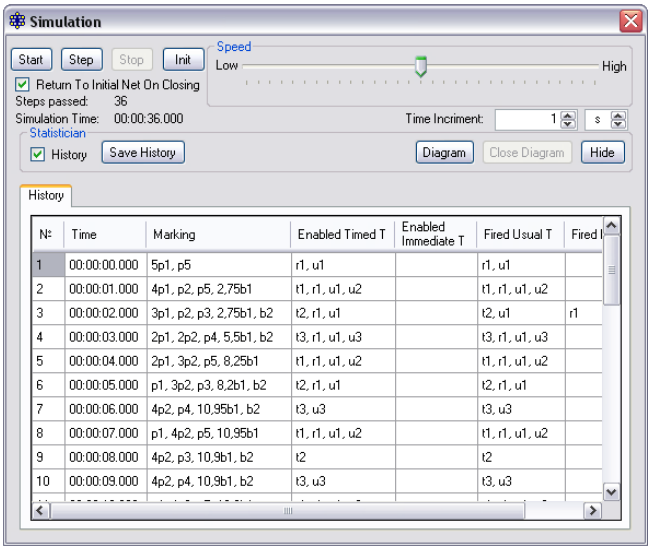

Figure 12. Simulation history.

VRDN offers good diagramming features used for visual analysis of the simulation process at run-time. It makes it possible to dynamically reflect in a chart or a series of chard the evolution of a specified parameter or an expression. This includes both 2 dimensional and 3 dimensional charts.

An example of such a char, obtaining at run-time during simulation is presented in Fig.13.

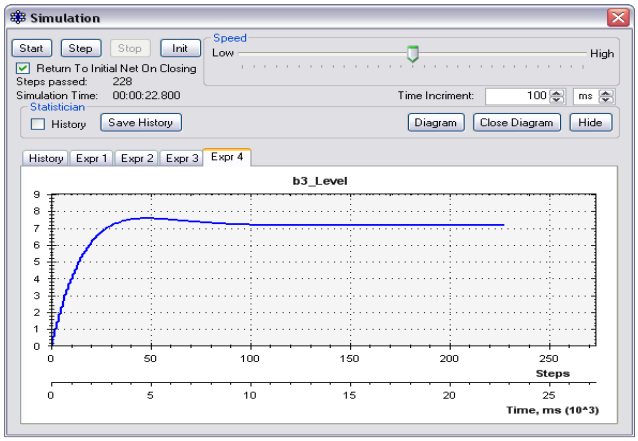

Figure 13. A sample of a simulation chart.

VI. CONCLUSION

In this paper we have discussed the possibility of timed *GDPN* models to study the performance characteristics of hybrid systems. When dealing with real cases, it often happens that the *HPN* models tend to become very complex. In these cases *GDPN* models can be used as formal specification of system and automatically translated into detailed simulation programs in order to estimate performance result.

The construction and the analysis of complex timed *GDPN* models can only be done with the help of powerful automatic tools such as the user-friendly software *VRDN* tool whose performance modeling facilities and software architecture have been described in this paper.

The integration within the same interface of the graphical facilities for the model construction, of the structural analysis algorithms for model validation, and of the control panels for performing the visual and interactive simulation, emphasizes the importance of including in a simulation experiment both validation and evaluation aspects of timed *GDPN* models .

The animation of the model, performed only when desired by the user, appears to be a powerful tool that complements the structural results for the debugging and tuning of the models used for the performance analysis.

A special interest in using VRDN could be its powerful simulation mechanisms for rewriting nets. This mechanism allows for dynamic specification and loading of replaced and replacing sub-nets.

Already *VRDN* is used in educational process; it gives students a grasp of timed hybrid Petri net principles, introducing them into world for performance evaluation of computer systems and communication networks.

As next steps in this research, our team is going to thoroughly analyze other classes and extensions of PN, such as colored timed hybrid PN and membrane colored PN.

REFERENCES

[1] A. Alla, H. David, "Continuous and hybrid Petri nets," Journal of Systems Circuits and Computers, 8(1), pp. 159-188, 1998.

[2] Demongodin, N.T. Koussoulas, "Differential Petri Nets: Representing continuous systems in a discrete-event world," IEEE Transaction on Automatic Control, Vol. 43, No. 4, 1998.

[3] M. Calzarossa, R. Marie, "Tools for Performance Evaluation," Performance Evaluation, no. 33, pp.1-3, 1998.

[4] C. Ciufudean, A. B. Larionescu, "Estimation of the Performances of The Discrete Events Systems," Advances in Electrical and Computer Engineering, no. 2, pp. 30-34, 2003.

[5] K. Compton, S. Hauck, "Reconfigurable Computing: a Survey of Systems and Software," ACM Computing Surveys (CSUR), vol. 34, no. 2, pp. 171-210, 1998.

[6] R. German, M. Gribaudo, G. Horvath, M. Telek, "Stationary analysis of FSPNs with mutually dependent discrete and continuous parts," Proceedings of 10th Int. Workshop on Petri Nets and Performance Models (PNPM'03), Urbana-Champaign, USA, September 2003, IEEE Comp. Soc. Press., p. 30-39, 2003.

[7] E. Guţuleac, M. Mocanu, I. Ţurcanu, "Dynamic Rewriting of Differential Petri Nets for Modeling of Hybrid Systems," In Proc. of the 2-nd International Conference on Intelligent Computer Communication and Processing, 1-2 September, Cluj-Napoca, România, pp. 105-112, 2006.

[8] E. Guţuleac, "Descriptive Timed Membrane Petri Nets for Modeling of Parallel Computing," International Journal of Computers, Communications & Control, Agora University, Oradea, România, no. 3, Vol. I, pp. 33-39, 2006.

[9] E. Guţuleac, "Descriptive compositional HSPN modeling of computer systems," Annals of the University of Craiova, vol. 3 (30), no.2, pp.82-87, 2006.

[10] E. Guţuleac, "Descriptive compositional HSPN based discrete - continuous modeling of distributed systems," Scientific Annals of the State University of Moldova, CEP USM, Chisinau, pp. 182-187, 2005.

[11] K. Hoffmann, H. Ehrig, T. Mossakowski, "High-Level Nets with Nets and Rules as Tokens," Proceedings of ICATPN'05, volume 3536 of LNCS, Springer, p. 268–288, 2007.

[12] M. Llorens, J. Oliver, "Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Nets," IEEE Transactions on Computers, vol. 53, no. 9, pp. 1147-1158, 2004.

[13] C. Lefter, M.H. Matcovschi, O. Pastravanu, "Computer-Aided Analysis and Design of Discrete-Event Systems with Petri Net Toolbox for MATLAB," În: Yagawa, Atanasiu G. and C. Brătianu (Eds.), Perform. Based Engineering for 21st Century, Ed. Cermi, p. 222-227, 2004.

[14] *http://ru.wikipedia.org/wiki/.NET_Framework.*

[15] Petri nets world - Petri nets tools database. *http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html*.

[16] T. Streichert, D. Koch, C. Haubelt, J. Teich, "Modeling and Design of Fault-Tolerant and Self-Adaptive Reconfigurable Networked Embedded Systems," EURASIP Journal on Embedded Systems, Volume 3, Hindawi Publishing Corporation, p. 1-15, 2006.

[17] B. Tuffin, D.S. Chen, K.S. Trivedi, "Comparison of Hybrid Systems and Fluid Stochastic Petri Nets," Discrete Event Dynamic Systems, 11(1&2), p. 77–96, 2001.