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

Emilian GUȚULEAC, Iurie TÜRCANU, Ion BALMUŞ, Victor CHEIBAȘ, Alexei CORDUNENU
Computer Science Department, Technical University of Moldova
Bul. Stefan cel Mare nr. 168, MD-2012 Chisinau, Republic of Moldova
e-gutuleac@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 Rewriting, software tool, visual simulation

Generalized Differential Petri Nets (RGDN) a class of Petri nets, that accept 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.

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 \( \mathcal{H} = \langle P, T, Pre, Post, Test, Inh, K_f, K_s, G, Pre, \text{IR}, W, V \rangle \), where:

- \( P \) is the finite set of places partitioned into a set of discrete places \( P_d \), and a set of continuous places \( P_c \).
- \( P = P_d \cup P_c \), \( P \cap P_c = \emptyset \). 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, \( T \cap P = \emptyset \), that is partitioned into a set \( T_d \) of discrete transitions and a set \( T_c \) of continuous transitions, that \( T = T_d \cup T_c \), \( T_d \cap T_c = \emptyset \).
- \( Pre, Test \) and \( Inh : P \times T \rightarrow \text{Bag}(P) \) respectively are a forward flow, test and inhibition functions. \( \text{Bag}(P) \) is a discrete or continuous multiset over \( P \). The backward flow function in the multiset of \( P \) is a \( \text{Post} : T \times P \rightarrow \text{Bag}(P) \), where define the set of arcs \( A \) and it describes the marking-dependent cardinality of arcs connecting transitions with places and vice-versa.
- \( 0 \leq K^{min}_p < +\infty < K^{max}_p \) which can contain an integer number of tokens, respectively. By default \( K^{min}_p = 0 \) and \( K^{max}_p \) being infinite value;
- The \( K_s : P_c \rightarrow \text{IR} \) is the function-capacity of continuous places and for each \( p_i \in P_c \) describes the fluid lower bound \( x^{min}_i \) and upper bounds \( x^{max}_i \) of fluid on each continuous place, that \(-\infty < x^{min}_i < x^{max}_i < +\infty \). This \( x^{max}_i \) by default it is \( \infty \), and bound has no effect when it is set to infinity. Continuous place has an implicit lower bound at level is \( 0 \).
- The \( \text{IR}_c \) and \( \text{IR} \) are the sets of discrete natural and real numbers, respectively.
- \( G : T \times \text{Bag}(P) \rightarrow \{ \text{True}, \text{False} \} \) 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);
- $\text{Pri}: T_D \rightarrow \mathbb{IN}$ 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
is enabled in
Fig. 1 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.

\[ q(t_k, M) = w(t_k, M)/ \sum_{j=1}^{N} w(t_j, M); \]

\[ V : T_c \times Bag(P) \rightarrow IR \text{ is the marking dependent fluid} \]
\[ \text{rate function of timed continuous transitions } T_c. \text{ These rates} \]
\[ \text{appear as labels next to the continuous timed transitions. If} \]
\[ t_i \in T_c \text{ is enabled in tangibile marking } M \text{ it fires with rate} \]
\[ V(M), \text{that continuously change the fluid level of continuous} \]
\[ \text{place } P_C. \]

III. DYNAMIC REWRITING GDPN SYSTEMS

Let $X \rho Y$ be a binary relation. The domain of $\rho$ is the
$\text{Dom}(\rho) = \rho Y$ and the codomain of $\rho$ is the $\text{Cod}(\rho) = X \rho$. Also, let $\text{Pre} = \text{Post, Test, Inh} >$ be a set of arcs
belonging to net $\mathcal{M} = < P, T, \text{Pre, Post, Test, Inh, } K_p, K_{\text{in}}, G, \text{Pri}>$.

A dynamic rewriting GDPN system (RGDN) is a
$\text{RH} = < H^\mathcal{M}, R, \phi, G, G_0, G_1, M >$, where:

\[ q(t_k, M) = w(t_k, M)/ \sum_{j=1}^{N} w(t_j, M); \]

\[ V : T_c \times Bag(P) \rightarrow IR \text{ is the marking dependent fluid} \]
\[ \text{rate function of timed continuous transitions } T_c. \text{ These rates} \]
\[ \text{appear as labels next to the continuous timed transitions. If} \]
\[ t_i \in T_c \text{ is enabled in tangible marking } M \text{ it fires with rate} \]
\[ V(M), \text{that continuously change the fluid level of continuous} \]
\[ \text{place } P_C. \]
\[ N = \langle H', \theta, W, V > \] 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 \[ P \cap T \cap R = \emptyset \]. In the graphical representation, the DR rule is drawn as two embedded empty rectangles;

- \[ \phi : E \rightarrow \{ T_D, R \} \] 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;

- \[ G_e : R \times Bag(P) \rightarrow \{ true, false \} \] is the transition rule guard function associated with \( r \in R \), and \[ G_e : R \times Bag(P) \rightarrow \{ true, false \} \] is the rewriting rule guard function defined for each rule of \( r \in R \), respectively. For \( \forall r \in R \), the function \( g_e(M) \in G_e \) and \( g_e(M) \in G_e \) 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 \( g_e(M) \in G_e \) is true and for \( g_e(M) \in G_e \) is false.

Let \( R' = H', R, \phi, G_e, G_r > \) and \( RN = \langle R', M > \) . A dynamic rewriting structure modifying rule \( r \in R \) of \( RN \) is a map \( r : RN \rightarrow RN \), where the codomain of the rewriting operator is a fixed subnet \( RN_L \) of current net \( RN \), where the domain of the rewriting operator 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 \( RN' = (RN \setminus RN_L) \cup RN_w \), where the meaning of \( \setminus \) and \( \cup \) is operation of removing (adding) \( RN_L \) from \( RN \) 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.

![Discrete rewriting primitives](image)

**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 the \( g_e(r_j, M) \) are 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 \( g_e(r_j, M) \) are verified.

Let \( T_D(M) \) and \( R(M), \) \( T_D(M) \cap R(M) = \emptyset, \) 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 \( e_i \in E(M) \) with higher priority is enabled. Hence, for the each event \( e_j \) if \( ((\phi_j = t_j) \lor (\phi_j = r_j) \land (g_e(r_j, M) = False)) \) 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) \rightarrow_{e_j} (R\Gamma', s') \iff (R\Gamma = R\Gamma' \land M[e_j > M'] \)

Also, for event \( e_j \) if \( ((\phi_j = r_j) \land (g_e(r_j, M) = True)) \) 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:

\[ (R\Gamma, s) \rightarrow_{r_j} (R\Gamma', s') \iff M[r_j > M'] \]

The state graph of an \( RN = \langle R', M > \) net is the labeled directed graph whose nodes are the states and whose arcs which are labeled with events or rewriting rules of \( RN \):

- **Change event:** arcs from state \( (R\Gamma, s) \) to state \( (R\Gamma', s') \) labeled with rewriting rule \( r_j \in R \), so that \( r_j : (R\Gamma, M) \triangleright (R\Gamma', M) \) which represent the change configuration of current \( RN \) net: \( (R\Gamma, s) \rightarrow_{r_j} (R\Gamma', s') \) with \( M[r_j > M'] \)

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

IV. VRDN SOFTWARE ENVIRONMENT

The VRDN is a software environment for construction, editing, visual simulation and performance analysis of RGDNs 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.](image)

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.](image)

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.](image)

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.](image)

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.](image)

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.

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 9. Simulation control form.](image)

![Figure 10. Initial rewriting net RPDR1.](image)

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 \( r_1: \text{RPDR1} \rightarrow \text{RPDR2} \) for this particular case at the next simulation step we may have the result presented in the Fig.11.

![Figure 11. Rewriting net RPDR2, as a result of applying rewriting rule.](image)

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.](image)

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 chart the evolution of a specified parameter or an expression. This includes both 2 dimensional and 3 dimensional charts.

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