VMPN–Software Tool for Performance Modeling of Dynamic Modifiable Structure Systems with Timed Membrane Petri Nets

Emilian GUTULEAC, Iurie TURCANU, Emilia GUTULEAC
Computer Science Department, Technical University of Moldova
Bd. Stefan cel Mare nr. 168, MD-2004 Chisinau, Republic of Moldova
e gutuleac@mail.utm.md

Abstract—In order to capture the compartmentalization and the behavior of membrane systems for performance modeling of parallel and distributed computing, we present the Descriptive Timed Membrane Petri Nets (TMPN) that can modify, in runtime, their own structure by rewriting some of their descriptive expression components. Furthermore, this descriptive approach facilitates the understanding of complex models and their component-based construction as well as the application of modern computer engineering concepts. A Visual Membrane Petri Nets (VHPN) software tool that offers a graphical user interface has been developed and described for visual simulation of TMPN models.

Index Terms—membrane systems, performance modeling, Petri nets, rewriting, reconfigurable, software tool

I. INTRODUCTION

Recent technological achievements require advances beyond the existing computational models in order to be used effectively. Thus, the software architectures provide a high-level system description in terms of a collection of computational components and connectors. Dynamic software architectures are those that change their structure and enact the modifications during the systems execution.

Also, in mobile ad-hoc networks and sensor networks, it is critical to accurately represent dynamic behavior in a precise and rigorous style. The dynamic behavior includes not only state change, but also change in structure [2, 11, 12].

Pragmatic aspects of current and future reconfigurable software architectures, computer systems, mobile ad-hoc networks and sensor networks, will be modeled so that realistic estimations of efficiency can be given for algorithms in these new settings.

However, the main technical challenge is to find a way to represent the structural changes of networks behavior, consistent with the conventional representation of dynamic state change, synchronization, concurrency, non-determinism. The representation must be mathematically based and explicitly represent timed structural behavior when run-time self-modifying systems are considered.

To do this requires the use of rigorous methods for specification, design, verification, performance evaluation which have been developed, including finite state machines, Petri nets [3, 5, 6] and P systems [9, 7], but they have focused on the representation of state change.

In previous studies [4, 5, 6] using descriptive expressions (DE), we have introduced the dynamic marked-controlled rewriting generalized Petri net (RGPN) and timed membrane Petri nets (TMPN) for performance modeling of parallel and distributed computing where a structure change and moving of membrane is described as transition rewriting rules. With RGPN and TMPN, we can easily and directly model concurrent and distributed systems that change their structure dynamically.

Traditional Petri net simulation environments [10], however, are not an adequate tool for our purposes for analysis of TMPN models, as they make use of the fixed static structure of nets.

In this paper, we present the Visual Membrane Petri Nets (VMPN) tool for the verification and performance modeling and of marked-controlled RGPN and TMPN that can in run-time modify their own structure by rewriting some of their components.

The paper is organized as follows. In Section 2 we present the model definition of timed RGPN. In Section 3 we consider the P systems and model definition of timed TMPN. Subsequently in Section 4 we considered the VMPN tool. Conclusions are drawn in Section 5.

II. DYNAMIC REWRITING TIMED PETRI NETS

In this section we introduce the model of descriptive dynamic net rewriting PN system. Let $X \rho Y$ be a binary relation. The domain of $\rho$ is the $\text{Dom}(\rho) = \rho X$ and the codomain of $\rho$ is the $\text{Cod}(\rho) = \rho Y$.

Definition 1. A descriptive dynamic marked-controlled rewriting generalized Petri net (RGPN) system is 6-tuple:

$\text{RN} = < \Gamma , \ R , \ \phi , G_\phi , G , \ M >$, where $\Gamma = < P, T, \ Pre, \ Post, \ Test, \ Inh, \ G, \ Pri, \ K_\rho >$ is a generalized Petri net structure [??];

$R = \{ r_1, ..., r_k \}$ is a finite set of rewriting rules about the runtime structural modification of net so that $P \cap T \cap R = \emptyset$.

In the graphical representation, the rewriting rule is drawn as two embedded empty rectangles. We let $E = T \cup R$ denote the set of events of the net;
\( \phi : E \rightarrow \{ T, R \} \) is the function that indicates for every rewriting rule the type of event can occur; 
\( G_r : R \times IN^{| \rho |}_r \rightarrow \{ True, False \} \) is the transition rule guard function associated with \( r \in R \), and 
\( G_j : R \times IN^{| \rho |}_j \rightarrow \{ True, False \} \) is the rewriting rule guard function defined for each rule \( r \in R \), respectively.

For \( \forall r \in R \), the \( g_r(M) \in G_r \) and \( g_j(M) \in G_j \) will be evaluated in each marking and if they are evaluated to \( True \), the rewriting rule \( r \) is enabled, otherwise it is disabled. Default value of \( g_r(M) \in G_r \) is \( True \) and for \( g_j(M) \in G_j \) is \( False \).

Let \( A = < Pre, Post, Test, Inh > \) be a set of arcs that belong to net \( \Gamma \) and \( R N =< R \Gamma, M > \) and 
\( R \Gamma =< \Gamma, R, \phi, G_r, G_j > \) be represented by the descriptive expression \( DE_{\Gamma R} \) and \( DE_{\Gamma N} \), respectively [5, 6]. A dynamic rewriting structure modifying the rule \( r \in R \) of \( R N \) is a map \( r : DE_{\Gamma R} \rightarrow DE_{\Gamma N} \), where the codomain of the rewriting operator, \( > \), is a fixed descriptive expression \( DE_{\Gamma R} \) of a subnet \( R N \) of current net \( R N \), where \( R N \) is a net with \( P \subseteq P \) and \( E \subseteq E \) and set of arcs \( A \subseteq A \) and the domain of \( > \) is a descriptive expression \( DE_{\Gamma N} \) of a new \( R N \) subnet with \( P \subseteq P \) and \( E \subseteq E \) and set of arcs \( A \). The rewriting operator, \( > \), represents a binary operation which produces a structure change in \( DE_{\Gamma N} \) of the net \( R N \) by replacing (rewriting) the fixed current \( DE_{\Gamma R} \) of the subnet \( R N \) ( \( DE_{\Gamma R} \) and \( R N \) are dissolved) by the new \( DE_{\Gamma N} \) of subnet \( R N \) that now belongs to the new modified resulting \( DE_{\Gamma N} \) of the net \( R N =< R \Gamma, M > \) with \( P' = (P \setminus P_L) \cup P_w \) and \( E' = (E \setminus E_j) \cup E_w \), 
\( A' = (A \setminus A_L) + A_w \), where the meaning of \( \setminus \) is operation of removing (adding) \( R N \) from \( R N \) to net \( R N \). In this new net \( R N' \), obtained by the execution of the enabled rewriting rule \( r \), the places and events with the same attributes which belong to \( R N' \) are fused. By default, the rewriting rules \( r : DE_{\Gamma R} \rightarrow DE_{\Gamma N} \) or \( r : DE_{\Gamma N} \rightarrow DE_{\Gamma R} \) describe the rewriting rule which maintains \( R N =< R \Gamma, M > \) and \( R N =< R \Gamma, M > \) of a state of \( R N \) net is the pair (\( R \Gamma, M \)), where \( R \Gamma \) is the configuration of the net together with a current marking \( M \). Also, the pair (\( R \Gamma, M \)) with \( P_0 \subseteq P \), \( E_0 \subseteq E \) and marking \( M_0 \) is called the initial state of the net.

Enabling and Firing of Events. The enabling of events depends on the marking of all places. We say that a transition \( t_j \) of the event \( e_j \) is enabled in current marking \( M \) if the following enabling condition \( ec(t_j, M) \) is verified:

\[
ec(t_j, M) = (\bigwedge_{\forall p \in \mathcal{P}_r} (m_i \geq \text{Pre}(p_i, t_j)) \land (m_k < \text{Inh}(p_k, t_j)) \land (m_j \geq \text{Test}(p_j, t_j)) \land ((K_{p_{n}} - m_n) \geq \text{Post}(p_n, t_j) \land g(t, M)).
\]

The rewriting rule \( r_j \in R \) is enabled in current marking \( M \) if the following enabling condition \( ec(p, r_j) \) is verified:

\[
ec(p, r_j) = (\bigwedge_{\forall p \in \mathcal{P}_r} (m_i \geq \text{Pre}(p_i, r_j)) \land (m_k < \text{Inh}(p_k, r_j)) \land (m_j \geq \text{Test}(p_j, r_j)) \land ((K_{p_{n}} - m_n) \geq \text{Post}(p_n, r_j) \land g_r(r_j, M)).
\]
Let we consider the RN1 net, given by the following descriptive expression:

\[
DE_{R1} = p_1 \mid r_1, \quad p_2 \lor DE'_{R1}.
\]

\[
DE'_{R1} = (p_2 \cdot p_5) \mid r_2, \quad p_3 \mid r_3, \quad \phi(p_3).
\]

\[
M_0 = (5p_1,1p_5), \quad r_i : DE_{R1} > DE'_{R1}, \quad g_r(r_1,M) = (m_1 = 3) \land (m_5 = 0).
\]

Also, for the rewriting rule \( r_j \) is required to identify if RN1 net belongs to the RT*. The firing of enabled events or rewriting rules modify the current marking and/or modify the structure and the current marking of RN1 in RN2 given by:

\[
DE_{R2} = p_1 \mid r_1, \quad p_2 \lor DE'_{R2}.
\]

\[
DE'_{R2} = (p_2 \cdot p_6) \mid r_3, p_4 \mid r_4, p_5 \lor DE''_{R2},
\]

\[
DE''_{R2} = p_3 \mid r_5, p_6 \mid r_6, \phi(p_6).
\]

\[
M_1 = (5p_1,3p_2,1p_3), \quad g_r(r_2,M) = (m_1 = 4) \land (m_3 = 1).
\]

The translation of expression (1) \( DE_{R1} \) in RN1 and expression (2) \( DE_{R2} \) in RN2 is shown in figure 1a and figure 1b, respectively.

Systems are described in timed PN (TPN) as interactions of components that can perform a set of activities associated with events. An event is \( e = (\alpha, \theta) \), where \( \alpha \in E \) is the type of an activity (action name), and \( \theta \) is the firing delay.

**Definition 2.** A timed RGPN system is a pair \( RTN = \langle RN, \theta \rangle \), where:

\[
RN = \langle \Gamma, R, \phi, G_r, G_r, M \rangle, \quad \Gamma = < P, T, Pre, Post, Test, Inh, G, Pri, K_p, l > \quad \text{with set of events}
\]

\[
E = E_0 \cup E_e, \quad \text{which can be partitioned into a set} \ E_0 \ \text{of immediate events and a set} \ E_e \ \text{of timed events, so that}
\]

\[
E_0 \cap E_e = \emptyset, \quad Pri(E_0) > Pri(E_e). \quad \text{The immediate event}
\]

\[
is \text{drawn as a thin bar and the timed event is drawn as a black rectangle for transitions and two embedded empty rectangles for rewriting rules;}
\]

\[
\theta : E \times IN^{|P|} \rightarrow IR, \quad \text{is the weight function that maps events onto real numbers IR} \quad \text{delays or weight speeds. It can be dependent of the marking. The delays}
\]

\[
\theta(e_j,M) = d_j(M) \quad \text{define the duration of timed events.}
\]

If several timed events \( e_j \in E(M) \) are concurrently enabled in current marking for the \( e_j \in p^+ = \{ \forall e_j \in E : Pri(e_j) > 0 \} \), either in competition or independently, we assume that a race condition exists between them. The evolution of the model will determine whether the other timed events have been aborted or simply interrupted by the resulting change. The \( \theta(e_j,M) = w_j(M) \) is the weight speed \( w_j \) immediate events \( e_j \in E_0 \). If several enabled immediate events are scheduled to fire at the same time in vanishing marking \( M \) with the weight speeds, then the probability of the enabled immediate event \( e_j \) to fire is:

\[
q_j(M) = w_j(M) / \sum_{e_j \in E(M) \land \theta(e_j,M)} w_j(M).
\]

where \( E(M) \) is the set of enabled events in \( M \). An immediate event has a zero firing time.}

**III. P SYSTEMS AND TIMED MEMBRANE PETRI NETS**

Here we give a brief review of P systems and its relation with TMPN. A full guide for P systems can be referred to [4]. The main components of P systems are membrane structures consisting of hierarchically embedded membranes in the outermost skin membrane. Each membrane encloses a region containing a multisets of objects and possibly other membranes.

In general, a basic evolution-communication P system with active membranes (of degree \( n \geq 0 \)) is \( \Gamma P = (O, H, \mu, \Omega, (\rho, \pi)) \), where: \( O \) is the alphabet of objects; \( H \) is a finite set of labels for membranes; \( \mu \) is a membrane structure consisting of \( n \) membranes labeled with elements \( h = 0,1,..., n-1 \) in \( H \); \( \Omega \) is the configuration, that is mapping from membrane \( h \) of \( \Gamma P \) (nodes in \( \mu \) ) to multisets of objects \( \omega_h \in \Omega \). \( \omega_h = (\omega_{h,i}), \quad i = 0,1,...,|\Omega| \) present in the corresponding region of membrane \( h \), then the system is created; \( \rho \) and \( \pi \) is respectively the set of rules \( \rho_h \in \rho \) and its priorities \( \pi_h = (\pi_{h,i}), \quad j = 0,1,..., k \). The active membranes can be of two forms of rules \( \rho_{h,i} \); a) the object rules (OR), i.e., evolving and communication rules concerning the objects; b) the membranes rules (MR), i.e., the rewriting rules about the structural modification of membranes.

The structure of a membrane is: \( [h\omega, [i_1],...,[i_j], i_h] \), and each object rule \( \rho_{h,i} \in \rho \) has a form:
\( \rho_{h,j} : \omega \rightarrow \omega_{\text{in}}, \omega_{\text{out}}, \omega_{\text{m}} \times \{ \delta, \neg \delta \} \), here \( l \) belongs to the label set of its sub-membranes, and \( \omega_{\text{in}}, \omega_{\text{out}}, \) and \( \omega_{\text{m}} \) denote the multisets of objects which will be kept in this membrane, send out of this membrane, and send into its sub-membrane labeled by \( l \), respectively. The \( \delta \) denotes the dissolving rule.

Here we present the \( \text{TMPN} \) for encoding of P systems mentioned above into descriptive dynamic rewriting TPN as a RTN. The basis for \( \text{TMPN} \) is a membrane RTN that is a DE net structure comprising: places and its capacity, transitions and its priority, and guard functions, weighed directed arcs from places to transitions and vice-versa, weighed inhibitory and test arcs.

Consider the P system \( \Gamma P \). The encoding of \( \Gamma P \) into \( \text{TMPN} \) is decomposed into two separate steps. First, for every membrane \([ h ] \) we associate: to each object \( \omega_{h,j} \in \omega_h \) one place \([ h ] \mu_{h,j} \in \omega_h \) with the initial marking \( \mu^0_{h,j} \), and to each rule \( \rho_{h,j} \in \rho \) one event \([ e_{h,j} ] \) that acts on this membrane. Second, for every membrane \([ h ] \) we define the \( \text{DE}_h \) of RTN\( h \) that corresponds to the initial configuration of the P system \( \Gamma P \) as \([ h ] \text{DE}_h \).

Let \( u, v \) and \( u', v' \) be a multisets of objects.

**Definition 3.** The \( \text{TMPN} \) of degree \( n \geq 0 \), is a construct
\[
(\text{DM}) = \bigcup_{h=0}^{n} [ h ] \text{DE}_h \]
where:

- The \( \rho_{h,j} : [ h ] [ k ] u \rightarrow v \) evolving object rule with multisets of objects \( u, v \) which will be kept in \([ h ] \) is encoded as:
\[
[ h ] [ k ] P_{h,u} [ k ] P_{h,v} [ k ] \nu = [ h ] \rho_{h,j} ;
\]

- The antiport rule, that realizes a synchronization of object \( c \) with the objects of the type \( \rho_{h,k} : \mu \rightarrow [ h ] [ k ] \mu \rightarrow [ h ] \nu \rightarrow [ h ] [ k, u'] [ k, v ] \)
\[
\mu = [ h ] \rho_{h,k} ;
\]

Also, the \( \rho_{h,k} : [ h ] [ k, v ] \rightarrow [ h ] [ k, u'] [ k, v ] \) antiport rule that moves objects from inside to outside a membrane, or vice-versa is encoded as:
\[
[ h ] [ k ] P_{h,u} [ k ] P_{h,v} [ k ] \nu = [ h ] \rho_{h,k} ;
\]

Because the configuration means both a membrane structure and the associated multisets of objects, we need the rewriting rules for processing membranes and multisets of objects as: \( \text{MR} = \{ \text{mr}_L, \text{mr}_R, \text{mr}_M, \text{mr}_S, \text{mr}_T, \text{mr}_E \} \).

The above membrane rewriting rules (realized by the rewriting events in \( \text{DE}_h \)) are defined as:

- **mr**: Change rewriting rule, that changes, in run-time, the current structure and the multisets of objects of membrane \( h \), encoded by descriptive expression \( \text{DE}_h \) and its marking \( M_h \) in a new structure \( \text{DE}_h' \) with new marking \( M_h' \):
\[
[ h ] \text{DE}_h \rightarrow [ h ] \text{DE}_h' \rightarrow [ h ] \text{DE}_h'' \rightarrow [ h ] \text{DE}_h''' \rightarrow [ h ] \text{DE}_h'''' ;
\]

- **mr**: Dissolve rewriting rule says that the membrane \( h' \) is dissolved and the objects as \( M_h \) and sub-membranes of membrane \( h' \) belong now to its parent membrane \( h \). The skin membrane cannot be dissolved:
\[
[ h ] \text{DE}_h \rightarrow [ h ] \text{DE}_h'' ' = [ h ] \text{DE}_h'' ;
\]

- **mr**: Create rewriting rule, says that the new membrane \( h' \) with \( \text{DE}_h'' \) and \( M_h' \) is created in membrane \( h \), the rest remain in the parent membrane \( h'' \):
\[
[ h ] \text{DE}_h \rightarrow [ h ] \text{DE}_h' \rightarrow [ h ] \text{DE}_h'' ;
\]

- **mr**: Divide rewriting rule says that the objects and sub-membranes are reproduced and added into membrane \( h' \) and into \( h'' \), respectively:
\[
[ h ] \text{DE}_h \rightarrow [ h ] \text{DE}_h' \rightarrow [ h ] \text{DE}_h'' ;
\]

- **mr**: Separate rewriting rule is the counterpart of the Merge rewriting rule and is done by a:
\[
[ h ] \text{DE}_h \rightarrow [ h ] \text{DE}_h''' ;
\]

- **mr**: Move rewriting rule where a membrane \( h'' \) can be moved out or moved into a membrane \( h' \) as a whole is done by a:
\[
[ h ] \text{DE}_h \rightarrow [ h ] \text{DE}_h'' ;
\]

Thus, using the \( \text{TMPN} \) facilitates a compact and flexible specification and verification of parallel computing models.

In order to describe the details of this approach, we present a simple but illustrative example of encoding \( \Gamma P \) into \( \text{TMPN} \).

Consider the following P system \( \Gamma P_1 \) of degree 3:
\[
\mu = [0, b, \{ a, \{ 2 \}, c \}], \quad O = \{ a, b, c, d \},
\]
\[
\omega_0 = \{ b \}, \omega_1 = \{ a \}, \rho_0 = \{ \rho_{0,1} : c \rightarrow dd \delta \},
\]
\[
\rho_{0,2} : b \rightarrow b_{in}, \quad \pi_0 = \{ \pi_{0,1} > \pi_{0,2} \},
\]
\[
\rho_1 = \{ \rho_{1,1} : a \rightarrow b_{in}, \rho_{1,2} : b \rightarrow a_{out} \}, \quad \pi_1 = \emptyset, \omega_2 = \emptyset, \pi_2 = \emptyset.
\]

The mapping solution for the initial configuration of \( \Gamma P_1 \) described by (3) is given by the \( \text{TMPN} \), where every object
can be represented as a place labeled as the name of objects: 
\[ l(p_{0,1})=l(p_{1,1})=a, \quad l(p_{0,2})=l(p_{1,2})=b, \]
\[ l(p_{0,3})=c, \quad l(p_{2,1})=l(p_{0,4})=d. \]

The number of tokens in this place denotes the number of occurrences of this object. Every object-rule can be represented by an event type transition. For example, in membrane 0, the rule \( p_{0,1} : c \to dd_{in2} \), can be described by a transition \( t_{0,1} \). Because two copies of object \( d \) are send to membrane 2, the weight of the arc \( (t_{0,1}, p_{2,1}) \) is 2, which denotes that whenever the rule \( p_{0,1} \) is performed, one copy of object \( c \) will be removed in membrane 0 and two copies of object \( d \) will be sent to membrane 2.

Up to now, all objects and rules of \( \Gamma P1 \) are encoded in DM1-net as following:

\[
DM^1 = \left[ \begin{array}{c} D E_0 \\ D E_1 \\ D E_2 \\ D E_3 \end{array} \right] = \left[ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} \right],
\]

\[
DE_0 = p_{0,1} \lor p_{0,2} \lor p_{0,3} \lor p_{0,4}[2],
\]

\[
DE_1 = p_{1,1} \lor p_{1,2} \lor p_{1,3} \lor p_{1,4}[2],
\]

\[
DE_2 = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

\[
DE_3 = p_{3,1} \lor p_{3,2} \lor p_{3,3} \lor p_{3,4},
\]

\[
DE_4 = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

\[
DE_5 = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

\[
DE_6 = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

The dissolving rule \( mr_1 = \delta \) is represented in DM1-net by the following \( mr_1 \) rule:

\[
r_{1,1} : \quad DM^1 > DM^1', \quad DM'^1 = \left[ \begin{array}{c} 0 \\ 0 \\ 0 \end{array} \right],
\]

\[
DE_0' = p_{0,1} \lor p_{0,2} \lor p_{0,3} \lor p_{0,4}[2],
\]

\[
DE_1' = p_{1,1} \lor p_{1,2} \lor p_{1,3} \lor p_{1,4}[2],
\]

\[
DE_2' = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

\[
DE_3' = p_{3,1} \lor p_{3,2} \lor p_{3,3} \lor p_{3,4},
\]

\[
DE_4' = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

\[
DE_5' = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

\[
DE_6' = p_{2,1} \lor p_{2,2} \lor p_{2,3} \lor p_{2,4}[2],
\]

\[
g(r_{1,1}, M^{DE_1}) = (M^{DE_1} = (p_{0,3}, 2p_{1,2}, 1p_{2,1})),
\]

where \( M^{DE_1} \) is the current marking of \( DM^1 \).

The translation of P system \( \Gamma P1 \) into DM1-net described by \( 4 \) and DM1-net described by \( 5 \) is shown in figure 2 and figure 3, respectively.

The reachability graph of DM1-net in the listing form is:

\[
M_0^{DE_1} = (1p_{0,3}, 1p_{1,1}) \cup U_1 > M_1^{DE_1};
\]

\[
M_1^{DE_1} = (1p_{0,3}, 2p_{1,2}, 1p_{2,1}) \cup U_2 > M_2^{DE_1};
\]

\[
M_2^{DE_1} = (1p_{0,3}, 2p_{0,2}, 2p_{0,4}) \cup U_3 > M_3^{DE_1};
\]

\[
M_3^{DE_1} = (1p_{0,3}, 3p_{0,2}, 1p_{0,4}) \cup U_4 > M_4^{DE_1};
\]

\[
M_4^{DE_1} = (1p_{0,3}, 4p_{2,0}) \cup U_5 > M_5^{DE_1};
\]

\[
U_2 = \{t_{0,1}, t_{1,1}, t_{2,1}\}; \quad U_3 = U_4 = \{t_{0,3}\};
\]

\[
U_5 = \{t_{0,3}, t_{1,3}, t_{2,3}\}; \quad U_3 = U_4 = \{t_{0,3}\};
\]

IV. VMPN TOOL OVERVIEW

VMPN is a software tool for construction, editing, visual simulation and analysis of untimed and timed RGPNs and TMPN models. Developing Environment: Microsoft Visual Studio 6. This tool consists of a Graphical User Interface (GUI), an animation modeling and a visualization component (see Fig. 3 and Fig. 4).

Figure 2. Translation of \( DE^1 \) into DM1-net for \( \Gamma P1 \).

Figure 3. Translation of \( DE^1 \) into DM1-net for \( \Gamma P1 \).

Figure 4. Graphical User Interface.
has only few scrollbars and dialog boxes.

Figure 5. Formula Editor of VMPN.

Main toolbar of GUI contains all the necessary tools for designing, editing and analyzing the topology of the VMPN, scale tool, Formula Editor (figure 5) and help button: a) Arrow – the purpose of this tool is to select and to move objects, to move selections and the changing object’s attributes; b) Designing are used to design the RGPN or TMPN; c) Editing – are represented by standard copy, paste and cut buttons and grid button – showing and hiding the grid; d) History Graph – invokes the new process thread to start qualitative analysis.

Figure 6. Simulation form of VMPN.

**Object Inspector** provides easy and fast access to every RGPN or TMPN objects property. That while changing property of the selected object, VMPN tool automatically changes the values of all the items which depend on this value and changes the general net topology of the project. User can observe all the changes immediately by means of the hierarchical tree of Object Inspector. It is very useful in case the user changes the topology of the network modeled.

**File browser.** To make easier work with files, the File Browser was included in VMPN application. This feature allows us to browse all the filenames of the projects in the Sample Directory, which makes switching between the projects more convenient. The VMPN tool interface was made up as a user-friendly interface by means of standard GUI objects, so the user will easily find all necessary attributes to use. Also, this tool provides easy access to the most frequently used tools by providing a toolbar consisting of *ImageButtons* which can be depressed to enable functions and a collection of editing features including *Cut, Copy, Paste, and Delete* to make the design process easier. The view of the design can be altered to zoom-in and zoom-out on portions of the design.

By providing a time parameter for each transition, we are able to simulate the net performance. A timed RGPN and TMPN models are simulated in two modes: auto and step by step. In the auto-mode, transitions are fired as long as they are enabled. In step-mode, transitions are fired according to the step chosen by the user. In all cases, token flow due to firing can be visualized on the screen.

Simulation is started from the separate form of a figure 6. The description of the form:

**Start**- start of a network on simulation; **Stop** - stops of simulation, can be continued by button **Start; Time** - modeling time; the **Toddler** - sets speed of simulation; **Close** - to close a Window. The panel for creation of diagrams as a chosen parameter is displayed in the diagram. For a choice of parameter all over again it is necessary to choose an object, then his parameter and press the button **Add graph.** For displaying a part of the form with diagrams press "Show ". If the diagrams are not necessary to be present it is possible to hide them, pressing "Hide ". If the diagram is used for inserting into a report or in another program, it can be done with the help of the buttons "Save to file" - preservation of the diagram in a file of a format name.bmp or copy to clipboard - to copy in the buffer of an exchange. The Simulator supports all types of objects realized in the program. The step of time simulation gets out automatically.

However, it may occur situations when only immediate transitions (rewriting rules) are enabled. When a transition fires, tokens will be symbolically moved along the input arcs from the input places to the transition and along the output arcs from the transition to the output places, according to the various arc multiplicities. Viewing the simulation visualization in this manner significantly enhances the user’s understanding of the nets dynamic behavior, and is very useful in presentations to non-specialists and of course for functional verifying (debugging) of the model behavior.

V. CONCLUSIONS

We have developed a VMPN tool to analyze, simulate, and verify computing process systems with timed GRPN and TMPN, that can modify, in run-time, their own structure by rewriting some rules of their subnet components.

The integration within the same graphical user interface of the facilities for the model construction and 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 GRPN and/or TMPN models.

The animation of the TMPN models, 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.

As further work, we will develop and integrate hybrid
TMPN models in VMPN tool and it will be possible to editing, visual simulation, behavioral and performance analysis.

REFERENCES


