9th International Conference on *DEVELOPMENT AND APPLICATION SYSTEMS, Suceava, Romania, May 22-24, 2008*

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

Emilian GUȚULEAC, Iurie ȚURCANU, Emilia GUȚULEAC
Computer Science Department, Technical University of Moldova
Bd. Stefan cel Mare nr. 168, MD-2004 Chisinau, Republic of Moldova
egutuleac@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 run-time, 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 ρ is the $Dom(r) = rY$ and the *codomain* of r is the $Cod(\rho) = Xr$.

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

$RN =< G,\ R, f, G_{tr}\ G_r,\ M >$, where G = < *P*, *T*, *Pre*, *Post*, *Test*, *Inh*, *G*, *Pri*, K_p> 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 $P \cap T \cap R = \varnothing$.

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;

$f: E \rightarrow \{T, R\}$ is the function that indicates for every rewriting rule the type of event can occur; $G_{tr}: R \times IN_{+}^{|P|} \rightarrow \{True, False\}$ is the *transition rule guard function* associated with $r \in R$, and $G_r: R \times IN_{+}^{|P|} \rightarrow \{True,\ False\}$ is the *rewriting rule guard function* defined for each rule of $r \in R$, respectively. For $\forall r \in R$, the $g_{tr}(M) \in G_{tr}$ and $g_r(M) \in G_r$ will be evaluated in each marking and if they are evaluated to *True*, the rewriting rule *r* is *enabled*, otherwise it is *disabled*. Default value of $g_{tr}(M) \in G_{tr}$ is *True* and for $g_r(M) \in G_r$ is *False*).

Let *A*= < *Pre*, *Post*, *Test*, *Inh* > be a set of arcs that belong to net *G* and $RN = < RG,\ M >$ and $RG = < G,\ R, f, G_{tr}\ G_r >$ be represented by the descriptive expression $DE_{R\Gamma}$ and DE_{RN}, respectively [5, 6]. A dynamic rewriting structure modifying the rule $r \in R$ of *RN* is a map $r: DE_L > DE_W$, where the *codomain* of the rewriting operator, $>$, is a fixed descriptive expression DE_L of a subnet RN_L of current net *RN*, where $RN_L \subseteq RN$ with $P_L \subseteq P$, $E_L \subseteq E$ and set of arcs $A_L \subseteq A$ and the *domain* of $>$ is a descriptive expression DE_W of a new RN_w subnet with $P_w \subseteq P, E_w \subseteq E$ and set of arcs A_W. The rewriting operator, $>$, represents a binary operation which produces a *structure change* in DE_{RN} of the *RN* net by replacing (rewriting) the fixed current DE_L of the subnet RN_L (DE_L and RN_L are dissolved) by the new DE_W of subnet RN_W that now belongs to the new modified resulting $DE_{RN'}$ of the net $RN' = (RN \setminus RN_L) \cup RN_W$ with $P' = (P \setminus P_L) \cup P_W$ and $E' = (E \setminus E_L) \cup E_W$, $A' = (A - A_L) + A_W$ where the meaning of \ ($\cup$) is operation of removing (adding) RN_L from (RN_W to) net *RN*. In this new net RN', obtained by the execution of the enabled rewriting rule $r \in R$, the places and events with the same attributes which belong to RN' are fused. By default, the rewriting rules $r: DE_L > \varnothing$ or $r: \varnothing > DE_W$ describe the rewriting rule which maintains $RN' = (RN \setminus RN_L)$ or $RN' = (RN \cup RN_W)$. A state of a *RN* net is the pair (RG, M), where *RG* is the configuration of the net together with a current marking *M*. Also, the pair (RG_0, M_0) with $P_0 \subseteq P, E_0 \subseteq E$ and marking M_0 is called the initial state of the net. n

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:

$$ec(t_j, M) = (\underset{\forall p_i \in {}^{\bullet}t_j}{\wedge}(m_i \geq Pre(p_i, t_j))\ \&\ \underset{\forall p_k \in {}^{\mathbf{o}}t_j}{\wedge}(m_k < Inh(p_k, t_j))\ \&\ \underset{\forall p_l \in {}^{*}t_j}{\wedge}(m_l \geq Test(p_l, t_j))\ \&\ \underset{\forall p_n \in t_j^{\bullet}}{\wedge}((K_{p_n} - m_n) \geq Post(p_n, t_j))\ \&\ g(t_j, M))$$

. The rewriting rule $r_j \in R$ is enabled in current marking *M* if the following enabling condition $ec_{tr}(r_j, M)$ is verified:

$$ec_{tr}(r_j, M) = (\underset{\forall p_i \in {}^{\bullet}r_j}{\wedge}(m_i \geq Pre(p_i, r_j))\ \&\ \underset{\forall p_k \in {}^{\mathbf{o}}r_j}{\wedge}(m_k < Inh(p_k, r_j))\ \&\ \underset{\forall p_l \in {}^{*}r_j}{\wedge}(m_l \geq Test(p_l, r_j))\ \&\ \underset{\forall p_n \in r_j^{\bullet}}{\wedge}((K_{p_n} - m_n) \geq Post(p_n, r_j))\ \&\ g_{tr}(r_j, M)) .$$

Let the *T*(*M*) and *R*(*M*), $T(M) \cap R(M) = \varnothing$, be the set of enabled transitions and rewriting rules in current marking *M*, respectively. Let the $E(M) = T(M) \cup R(M)$, be the set of enabled events in a current marking *M*. The event $e_j \in E(M)$ fires if no other event $e_k \in E(M)$ with higher priority has been enabled. Hence, for e_j event *if* $((f_j = t_j) \vee (f_j = r_j) \wedge (g_{tr}(r_j, M) = False))$ *then* the firing of the transition $t_j \in T(M)$ or of the rewriting rule $r_j \in R(M)$ changes only the current marking:

$(RG,\ M) \xrightarrow{e_j} (RG,\ M') \Leftrightarrow (RG = RG$ and the $M[e_j > M'$ in RG)).

Also, for the every event $e_j \in E$, *if* $((f_j = r_j) \wedge (g_r(r_j, M) = True))$ *then* the event e_j occurs at firing of the rewriting rule r_j and it changes the configuration and marking of the current net, so that:

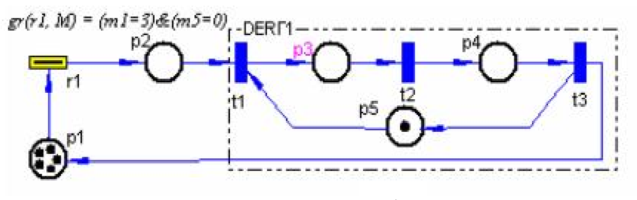
$(R\Gamma,\ M) \xrightarrow{r_j} (R\Gamma',\ M')$, $M[r_j > M'$).

The accessible state graph of a net $RN = < RG, M >$ is the labeled directed graph whose nodes are states and whose arcs, which are labeled with events or rewriting rules of *RN*, are of two kinds:

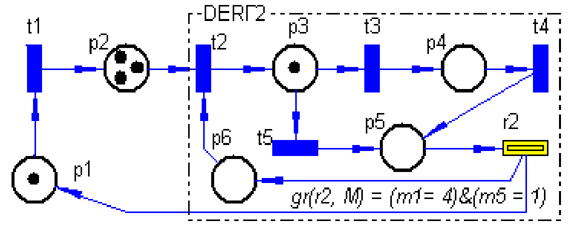
a) *firing* of an enabled event $e_j \in E(M)$: arcs from state (RG, M) to state (RG, M') labeled with event e_j, so that this event can fire in the configuration *RG* at marking *M* and leads to a new marking:

$M': (RG, M) \xrightarrow{e_j} (RG', M') \Leftrightarrow (RG = RG'$ and $[M[e_j > M'$ in RG);

b) *change configuration*: arcs from state (RG, M) to state(RG', RM') labeled with the rewriting rule $r_j \in R$, $r_j: (RG_L, M_L) > (RG_W, M_W)$ which represent the change configuration of current *RN* net: $(RG,\ M) \xrightarrow{r_j} (RG',\ M')$ with $M[r_j > M'$.



a)



b)

Figure 1. Translation: a) DE_{RG1} in *RN*1 and b) DE_{RG2} in *RN*2.

Let we consider the *RN*1 net, given by the following descriptive expression:

$$DE_{RG1} = p_1 |_{r_1} \; p_2 \vee DE'_{RG1},$$

$$DE'_{RG1} = (p_2 \cdot p_5) |_{t_1} \; p_3 |_{t_2} \; p_4 |_{t3} \; (p_1 \Diamond p_5), \qquad (1)$$

$$M_0 = (5p_1, 1p_5), \; r_1 : DE_{RG1} > DE_{RG2},$$

$$g_r(r_1, M) = (m_1 = 3) \& (m_5 = 0).$$

Also, for the rewriting rule r_j is required to identify if RN_L net belongs to the RG . The firing of enabled events or rewriting rules modify the current marking and/or modify the structure and the current marking of *RN*1 in *RN*2 given by:

$$DE_{RG2} = p_1 |_{t_1} \; p_2 \vee DE'_{RG2},$$

$$DE'_{RG2} = (p_2 \cdot p_6) |_{t_2} \; p_3 |_{t_3} \; p_4 |_{t_4} \; p_5 \vee DE''_{RG2},$$

$$DE''_{RG2} = p_3 |_{t_5} \; p_5 |_{r_2} \; (p_1 \Diamond p_6), \qquad (2)$$

$$r_2 = r_1^{-1} : DE_{RG2} > DE_{RG1}, \; M = (1p_1, 3p_2, 1p_3),$$

$$g_r(r_2, M) = (m_1 = 4) \& (m_5 = 1).$$

The translation of expression (1) DE_{RG1} in *RN*1 and expression (2) DE_{RG2} in *RN*2 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 $e = (a, q)$, where $a \in E$ is the type of an activity (action name), and q is the firing delay.

Definition 2. A timed *RGPN* system is a pair *RTN* = < *RN*, q >, where:

$RN = < G, \; R, f, G_{tr} \; G_r, \; M >$, G = < *P*, *T*, *Pre*, *Post*, *Test*, *Inh*, *G*, *Pri*, K_p, *l* > with set of events $E = E_0 \cup E_t$ which can be partitioned into a set E_0 of *immediate* events and a set E_t of *timed* events, so that $E_0 \cap E_t = \varnothing$, $Pri(E_0) > Pri(E_t)$. The immediate event is 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;

$q : E \times IN_+^{|P|} \to IR_+$ is the weight function that maps events onto real numbers IR_+ (delays or weight speeds). It can be dependent of the marking. The delays $q(e_k, M) = d_k(M)$ define the duration of timed events.

If several timed events $e_j \in E(M)$ are concurrently enabled in current marking for the $e_j \in {}^{\bullet}p_i = \{\forall e_j \in E : \mathrm{Pr}\,e(p_i, e_j) > 0\}$, either in competition or independently, we assume that a *race competition* 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 ıte change. The $q(e_j, M) = w_j(M)$ is the weight speed oı 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:

$$q_j(M) = w(e_j, M) / \sum_{e_l \in (E(M) \& {}^{\bullet}p_i)} w(e_l, M),$$

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

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 $GP = (O, H, m, W, (r, p))$, where: *O* is the alphabet of objects; *H* is a finite set of labels for membranes; m is a membrane structure consisting of *n* membranes labeled with elements *h* = 0,1,…, *n*-1 in *H*; W is the configuration, that is mapping from membrane *h* of GP (nodes in m) to multisets of objects $w_h \in W$, $w_h = \{w_{h,i}\}$, *i*=0,1,…,$|O|$ present in the corresponding region of membrane *h*, then the system is created; r and p is respectively the set of rules $r_h = \{r_{h,j}\}$ and its priorities $p_h = \{p_{h,j}\}$, *j*=0,1,…, *k*. The active membranes can be of two forms of rules $r_{h,j}$: 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: $[_h w, [_i]_i, ..., [_l]_l]_h$, and each object rule $r_{h,j} \in r$ has a form:

$r_{h,j}: w \to w_{here} w_{out} w_{in_l} \times \{d, \neg d\}$, here *l* belongs to the label set of its sub- membranes, and w_{here}, w_{out}, and w_{in_l} 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 d denotes the dissolving rule.

Here we present the *TMPN* for encoding of P systems mentioned above into descriptive dynamic rewriting TPN as a *RTN*. The basis for *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 GP. The encoding of GP into *TMPN* is decomposed into two separate steps. First, for every membrane $[_h\ ]_h$ we associate: to each object $w_{h,i} \in w_h$ one place $[_h\ m_i^0 p_{h,i}\ ]_h$ with the initial marking $m_{h,i}^0$, and to each rule $r_{h,j} \in r$ one event $[_h\ e_{h,j}\ ]_h$, that acts on this membrane. Second, for every membrane $[_h\ ]_h$ we define the DE_h of RTN_h that corresponds to the initial configuration of the P system GP as $[_h\ DE_h\ ]_h$.

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

Definition 3. The *TMPN* of degree $n \geq 0$, is a construct $DM = \vee_{h=0}^{n-1} [_h\ DE_h^0\ ]_h$, where:

- The $r_{h',j}: [_h\ [_{h'}\ u \to v\ ]_{h'}\ ]_h$ *evolving* object rule with multiset of objects u, v which will be kept in $[_{h'}\ ]_{h'}$ is encoded as:

$[_h\ [_{h'}\ p_{h,u} |_{t_{h,j}}\ p_{h,v}\ ]_{h'}\ ]_h$;

- The *antiport* rule, that realizes a synchronization of object c with the objects of the type $r_{h',i}: [_h u [_{h'}\ v\ ]_{h'}\ ]_h \to [_h v' [_{h'}\ u'\ ]_{h'}\ ]_h$, is encoded as:

$[_h\ [_{h'}\ (p_{h',u} \cdot p_{h',v} \cdot \tilde{p}_{h',c}) |_{t_{h',i}}\ (p_{h',u'} \Diamond p_{h',v'})\ ]_{h'}\ ]_h$.

Also, the $r_{h',k}: [_h u\ [_{h'}\ ]_{h'}\ ]_h \to [_h\ [_{h'}\ u'\ ]_{h'}\ ]_h$ *symport* rule that moves objects from inside to outside a membrane, or vice-versa is encoded as:

$[_h\ [_{h'}\ (p_{h',u} \cdot \tilde{p}_{h',c}) |_{t_{h',k}}\ p_{h',u'}\ ]_{h'}\ ]_h$.

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: *MR* = {*mr*$_0$, *mr*$_1$, *mr*$_2$, *mr*$_3$, *mr*$_4$, *mr*$_5$, *mr*$_6$}.

The above membrane rewriting rules (realized by the rewriting events in *DE's*) are defined as:

- *mr*$_0$: *Change* rewriting rule, that changes, in run-time, the current structure and the multisets of objects of membrane *h*, encoded by descriptive expression $DE_{h'}$ and its marking $M_{h'}$ in a new structure $DE'_{h'}$ with new marking $M'_{h'}$:

$[_h\ [_{h'}\ DE_{h'}\ ]_{h'}\ ]_h > [_h\ [_{h'}\ DE'_{h'})\ ]_{h'}\ ]_h$;

- *mr*$_1$: *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*:

$[_h DE_h\ [_{h'}\ DE_{h'}\ ]_{h'}\ ]_h > [_h\ \ DE'_h\ \ ]_h$,

$M'_h = M_h + M_{h'}$;

- *mr*$_2$: *Create* rewriting rule, says that the new membrane h' with $DE''_{h'}$ and $M''_{h'}$ is created in membrane *h*, the rest remain in the parent membrane *h*:

$[_h\ \ DE_h\ \ ]_h > [_h DE'_h\ \ [_{h'}\ DE''_{h'}\ ]_{h'}\ ]_h$,

$M_h = M'_h + M''_{h'}$;

- *mr*$_3$: *Divide* rewriting rule says that the objects and sub-membranes are reproduced and added into membrane h' and into h'', respectively:

$[_h\ \ DE_h\ \ ]_h > [_h\ [_{h'}\ DE_h\ ]_{h'}\ \ [_{h''}\ DE_h\ \ ]_{h''}\ ]_h$;

- *mr*$_4$: *Merge* rewriting rule says that the objects of membrane h' and h'' are added to a new membrane h is:

$[_h [_{h'}\ DE'_{h'}\ ]_{h'}\ [_{h''}\ DE''_{h''}\ \ ]_{h''}]_h > [_h\ DE'_{h'} \vee DE''_{h''}\ \ ]_h$

with new marking $M'_{h'} + M''_{h''} = M_h$;

- *mr*$_5$: *Separate* rewriting rule is the counterpart of the *Merge* rewriting rule and is done by a:

$[_h\ DE'_{h'} \vee DE''_{h''}\ ]_h > [_h [_{h'}\ DE'_{h'}]_{h'}\ [_{h''} DE''_{h''}\ ]_{h''}]_h$

with meaning that the content of membrane h is split into two membranes, with labels h' and h'', and the new marking is $M_h = M'_h + M''_{h'}$;

- *mr*$_6$: *Move* rewriting rule where a membrane h'' can be moved out or moved into a membrane h' as a whole is done by a:

$[_h\ [_{h'} DE_{h'}\ [_{h''} DE_{h''}\ ]_{h''}]_{h'}]_h > [_h$
$[_{h'} DE_{h'}\ ]_{h'}\ [_{h''}\ DE_{h''}\ ]_{h''}]_h$

or $[_h [_{h'}\ DE_{h'}\ ]_{h'} [_{h''} DE_{h''}\ ]_{h''}]_h > [_h [_{h'}$
$DE_{h'}\ [_{h''}\ DE_{h''}\ ]_{h''}]_{h'}]_h$

with their marking, respectively. **n**

Thus, using the *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 GP into *TMPN*.

Consider the following P system $GP1$ of degree 3:

$$m = [_0\ b,\ [_1\ a,\ [_2\ ]_2,\ ]_1,\ ]_0,\ O = \{a, b, c, d\},$$
$$w_0 = \{b\}, w_1 = \{a\},\ r_0 = \{r_{0,1}: c \to dd_{in2},$$
$$r_{0,2}: b \to b_{in1}\},\ p_0 = \{p_{0,1} > p_{0,2}\}, \qquad (3)$$

$$r_1 = \{r_{1,1}: a \to b_{here} c_{out} d_{in2},\ r_{1,2}: b \to a_{out} d\},$$
$$r_{1,2}: b \to a_{out} d\},\ p_1 = \varnothing, w_2 = \varnothing,\ p_2 = \varnothing.$$

The maping solution for the initial configuration of $GP1$ described by (3) is given by the *TMPN*, where every object can be represented as a place labeled as the name of objects:

$$l(p_{0,1})=l(p_{1,1})=a\ ,\ l(p_{0,2})=l(p_{1,2})=b\ ,$$
$$l(p_{0,3})=c\ ,\ l(p_{2,1})=l(p_{0,4})=d\ .$$

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 $r_{0,1}: c \rightarrow dd_{in2}$, 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 $r_{0,1}$ 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 $GP1$ are encoded in *DM1-net* as following:

$$DM1=[_0\ DE_0\ [_1\ DE_1\ [_2\ DE_2\ ]_2\ ]_1\ ]_0\ ,$$
$$DE_0 = p_{0,1} \vee 1p_{0,2}\ |_{t_{0,2}}\ p_{2,1} \vee p_{0,3}\ |_{t_{0,1}}\ p_{2,1}[2],$$
$$DE_1 = 1p_{1,1}\ |_{t_{1,1}}\ (p_{0,3} \Diamond p_{1,2}\ |_{r_{1,1}}\ p_{0,1} \Diamond p_{2,1})\ , \qquad (4)$$
$$DE_2 = p_{2,1}\ |_{t_{2,1}}\ p_{1,2}\ ,\ \Pr i(t_{0,1})\ ) > \Pr i(t_{0,2})\ ,$$
$$DE1 = DE_0 \vee DE_1 \vee DE_2\ .$$

The dissolving rule $mr_1 = d$ is represented in *DM1-net* by the following mr_1 rule:

$$r_{1,1}:\ DM\ 1 > DM\,'1\ ,\ DM\,'1=[_0\ DE_0'\ ]_0\ ,$$
$$DE_0' = 1p_{0,1} \vee p_{0,3}\ |_{t_{0,1}}\ 2p_{0,4}[2]\ |_{t_{0,2}}\ 2p_{0,2}\ , \qquad (5)$$
$$gr(r_{1,1}, M^{DE1}) = (M^{DE1} = (p_{0,3}, 2p_{1,2}, 1p_{2,1}))\ ,$$

where M^{DE1} is the current marking of $DM\ 1$.

The translation of P system $GP1$ into *DM1-net* described by (4) and *DM'1-net* described by (5) is shown in figure 2 and figure 3, respectively.

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

$$M_0^{DE1} = (1p_{0,2}, 1p_{1,1})\ [U_1 > M_1^{DE1};$$
$$M_1^{DE1} = (1p_{0,3},\ 2p_{1,2}, 1p_{2,1})\ [U_2 > M_2^{DE'1};$$
$$M_2^{DE'1} = (1p_{0,1},\ 2p_{0,2}, 2p_{0,4})\ [U_3 > M_3^{DE'1};$$
$$M_3^{DE'1} = (1p_{0,1},\ 3p_{0,2}, 1p_{0,4})\ [U_4 > M_4^{DE'1};$$
$$M_4^{DE'1} = (1p_{0,1},\ 4p_{0,2}\ )\,[\ ,\ \text{where } U_2 = \{t_{0,1}, r_{1,1}, t_{2,1}\}$$
$$U_2 = \{t_{0,1}, r_{1,1}, t_{2,1}\}\ ;\ U_3 = U_4 = \{t_{0,3}\}\ .$$

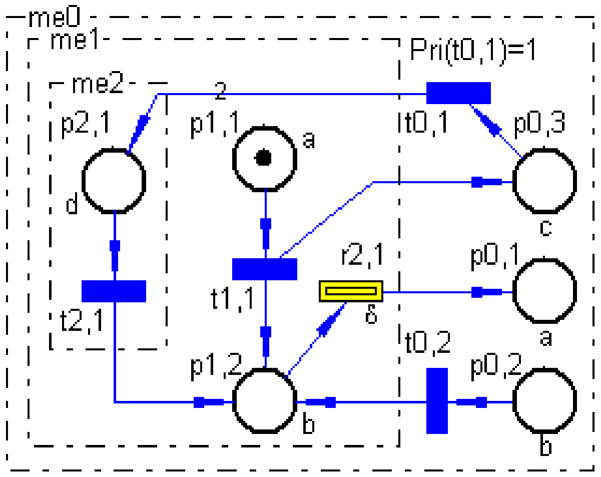


Figure 2. Translation of $DE1$ into *DM1-net* for $GP1$.

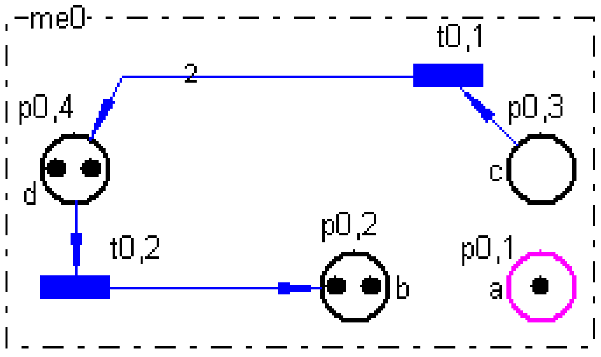


Figure 3. Translation of $DE'1$ into $DM'1$*-net* for $GP1$.

IV. VMPN TOOL OVERVIEW

VMPN is a software tool for construction, editing, visual simulation and analysis of untimed and timed *RGPN*s 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).

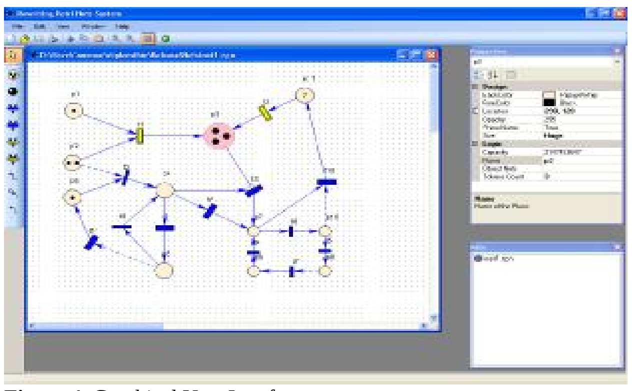


Figure 4. Graphical User Interface.

This editor allows us to create, save and load Petri nets according to the last standard PNML (Petri Net Markup Language [6]) for Petri nets. The GUI (figure 4) is designed with multi-document capability, allowing several nets to be edited simultaneously. This allows the user to study the behavior of the net by observing the token game. The GUI has only few scrollbars and dialog boxes.

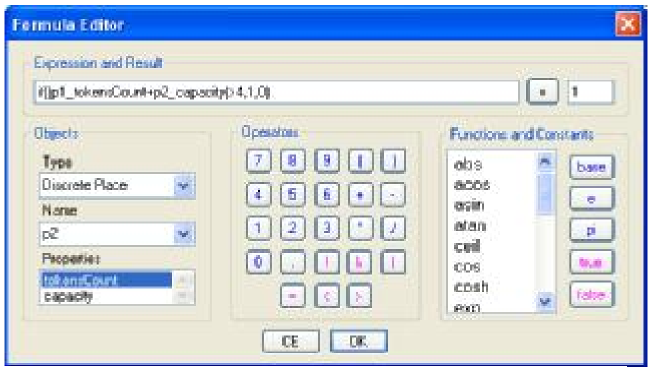


Figure 5. Formula Editor of *VMPN*.

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.

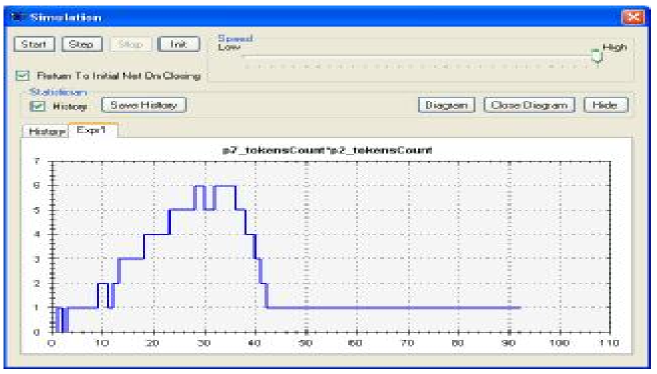


Figure 6. Simulation form of *VMPN*.

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

[1] Calzarossa, M., and Marie, R. Tools for Performance Evaluation. Performance Evaluation 33, 1998, pp. 1-3.

[2] Cook, S., Harrison, R., and Wernik, P. A simulation model of self-organising evolvability in software systems. In: Proceedings of the 1-st IEEE International Workshop on Software Evolvability, Hungary, 2005, pp. 17-22.

[3] Compton, K., and Hauck, S., Reconfigurable Computing: a Survey of Systems and Software. ACM Computing Surveys (CSUR), vol. 34, No. 2, 2002, pp. 171-210.

[4] Guțuleac, E., Descriptive Timed Membrane Petri Nets for Modelling of Parallel Computing. International Journal of Computers, Communications & Control, No: 3, Vol. I, Oradea, România, 2006, pp. 33-39.

[5] Guțuleac, E., Descriptive self-reconfigurable generalized stochastic Petri nets for performance modeling of computer systems, Buletinul Institutului Politehnic din Iaşi, Tomul LI (LV), Fasc. 1-4, Automatica şi Calculatoare, România, 2005, pp. 121-136.

[6] Guțuleac, E., Mocanu M., Descriptive Dynamic Rewriting GSPN-based Performance Modeling of Computer Systems, In Proc. of the 15th Intern. Conf. CSCS15, 25-27 May 2005, Bucureşti, România, 2005, pp. 656-661.

[7] Kleijn J., Koutny M., Rozenberg G., Towards a Petri Net Semantics for Membrane Systems. In Proceedings of the WMC6 2005, July 18-21, Wien, Austria, 2005, pp. 439-459.

[8] Home Page of PNML, Available: http://www.informatik.hu-berlin.de/top/pnml/

[9] Păun, Gh., Membrane Computing. An Introduction, Natural computing Series. ed. G. Rozenberg, Th. Back, A.E. EibenJ.N. Kok, H.P. Spaink, Leiden Center for Natural Computing, Springer–Verlag, Berlin, 2002, p. 420.

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

[11] Sekanina, L., Evolvable computing by means of evolvable components. Kluwer Academic Publishers, Natural Computing vol 3, 2004, pp. 323–355.

[12] Shrivastava, S., and Wheater, S., Architectural Support for Dynamic Reconfiguration of Large Scale Distributed Applications. In: Proceedings of the 4th International Conference on Configurable Distributed Systems, May 1998, pp. 10–17.

[13] Calin CIUFUDEAN, George MAHALU, Modelling Artificial Social Systems with Petri Nets, Advances in Electrical and Computer Engineering, Suceava, Romania, ISSN 1582-7445, No 2/2002, volume 2 (9), pp. 10-14.