Technical Report #9601

# MODELING USING TIMED PETRI NETS – MODEL DESCRIPTION AND REPRESENTATION

by

W.M. Zuberek

Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada A1B 3X5

August 1996

Department of Computer Science
Memorial University of Newfoundland
St. John's, Canada A1B 3X5
tel: (709) 737–8627
fax: (709) 737–2009



The Natural Sciences and Engineering Research Council of Canada
partially supported this research through Research Grant A8222.

# MODELING USING TIMED PETRI NETS – MODEL DESCRIPTION AND REPRESENTATION

**A b s t r a c t**

A collection of software tools, TPN–tools, for analysis of timed Petri nets, developed over years of extensions, modifications and redesigns, contains several tools for structural and reachability analysis of net models. As both structural and reachability analyses impose certain restrictions on the class of analyzed nets, a simulation tool, TPNsim, has recently been added to the collection. All these tools use the same (internal) representation of nets, so the integration of different tools is quite straightforward.

This report describes the specification of timed Petri net models used by TPN–tools. It discusses the structure of specifications (i.e., the 'input language'), provides several examples of net descriptions, and discusses the internal representation of net models used by the tools. Presented model descriptions are rather 'low–level', and it is expected that a more convenient graphical user interface will be developed (or adopted) at some later time.

**A c k n o w l e d g e m e n t**

Software tools for analysis of timed Petri net models were developed over many years, using direct and indirect contributions of many colleagues and students. All these contributions, rather difficult to trace and identify because of countless software modifications, redesigns and extensions, are appreciated and gratefully acknowledged.

## INTRODUCTION

Petri nets have been proposed as a simple and convenient formalism for modeling systems that exhibit parallel and concurrent activities [Ag79, Pe81, Re85, Mu89]. These activities are represented by the so called tokens which can move within a (static) graph–like structure of the net. More formally, a marked (place/transition) Petri net $\mathcal{M}$ contains two basic components, the structure $\mathcal{N}$ which is a bipartite directed graph (i.e., a graph with two types of nodes, called places and transitions), and an initial marking function $m_0$, which assigns nonnegative numbers of tokens to places of the net, $\mathcal{M} = (\mathcal{N}, m_0)$, $\mathcal{N} = (P, T, A)$, $m_0 : P \rightarrow \{0, 1, ...\}$. The directed arcs connect places with transitions and transitions with places, $A \subseteq T \times P \cup P \times T$. Place/transition Petri net models are also called 'condition/event' systems as places usually represent conditions (in the most general sense), while transitions – events. If a nonzero number of tokens is assigned to a place, the place is 'marked', which means that the condition represented by it is satisfied. If all places connected by directed arcs to a transition are marked, the transition is 'enabled' and can fire. Firing a transition is an instantaneous event which removes (simultaneously) a single token from each of the input places of the transition and adds a single token to each of the transition's output places. This creates a new marking function of a net, a new set of enabled transitions, and so on. The set of all possible marking functions which can be derived in such a way is called the set of reachable markings (or the forward marking class) of a net. This set can be finite or infinite; if it is finite, the net is bounded.

An important extension of the basic net model is addition of inhibitor arcs [AF73, Va82]. Inhibitor arcs (which connect places with transitions) provide a 'test if zero' condition which is nonexistent in the basic Petri net. Nets with inhibitor arcs are usually called inhibitor nets. In inhibitor nets, a transition is enabled only if all places connected to it by directed arcs are marked and all places connected by inhibitor arcs are empty (i.e., not marked). Formally, the set of inhibitor arcs $B$ is an additional element of the net structure, $\mathcal{N} = (P, T, A, B)$, and usually the same place cannot be connected by a directed and inhibitor arc with the same transition, so $A \cap B = \Phi$.

A place is shared if it is connected to more than one transition. A shared place is guarded if for every pair of transitions sharing it there exists another place which is connected by a directed arc to one of these two transitions and by an inhibitor arc to the other transition; consequently, if a place is guarded, at most one of the transitions sharing it can be enabled by any marking function.

If all shared places of a net are guarded, the net is conflict–free, otherwise the net contains conflicts. The simplest case of conflicts is known as a free–choice (or generalized free–choice) structure; a shared place is (generalized) free–choice if all transitions sharing it are connected with the same places by directed as well as inhibitor arcs (i.e., all sharing transitions have identical input and inhibitor sets). An inhibitor net is free–choice if all shared places are either guarded or free–choice. The transitions sharing a free–choice place constitute a free–choice class of transitions. For each marking function, and each free–choice class of transitions, either all transitions in a class are enabled or none of them is. It is assumed that the selection of transitions for firing within each free–choice class is a random process which can be described by 'choice probabilities' assigned to (free–choice) transitions. Moreover, it is usually assumed that the random variables describing choice probabilities in

different free–choice classes are independent.

Another popular extension of the basic model is to allow multiple arcs between places and transitions. A transition is enabled in such nets only if the number of tokens is at least equal to the number of directed arcs between a place and a transition. Formally this extension can be described by a 'weight function' $w$ which maps the set of directed arcs $A$ into the set of positive numbers, $\mathcal{N} = (P, T, A, B, w)$, $w : A \rightarrow \{1, 2, ...\}$. It should be observed that inhibitor arcs could be included in the same description as arcs with the weight equal to zero, but it appears that separating the set of inhibitor arcs from the set of directed arcs is a more convenient approach.

In (ordinary) nets the tokens are indistinguishable, so their distribution can conveniently be described by a marking function which maps the set of places into the set on nonnegative integer numbers. In colored Petri nets [Je87], tokens have attributes called colors. Token colors can be quite complex, for example, they can describe the values of (simple or structured) variables or the contents of message packets. Token colors can be modified by (firing) transitions and also a transition can have several different occurrences (or variants) of firing.

The basic idea of colored nets is to 'fold' an ordinary Petri net. The original set of places is partitioned into a set of disjoint classes, and each class is replaced by a single place with token colors indicating which of the original places the tokens belong to. Similarly, the original set of transitions is partitioned into a set of disjoint classes, and each class is replaced by a single transition with occurrences indicating which of the original transitions the firing corresponds to.

Any partition of places and transitions will result in a colored net. One of the extreme partitions will combine all original places into one place, and all original transitions into one transition; this will create a very simple net (one place and one transition only) but with quite complicated rules describing the use of colors. The other extreme partition will create one–element classes of places and transitions, so the colored net will be isomorphic to the original net, with only one color. To be useful in practice, colored nets must constitute a reasonable balance between these two extreme cases.

In order to study performance aspects of Petri net models, the duration of activities must also be taken into account and included into model specifications. Several types of Petri nets 'with time' have been proposed by assigning 'firing times' to the transitions or places of a net. In timed nets, firing times are associates with transitions, and transition firings are 'real–time' events, i.e., tokens are removed from input places at the beginning of the firing period, and they are deposited to the output places at the end of this period (sometimes this is also called a 'three–phase' firing mechanism as opposed to 'one–phase' instantaneous firings of transitions). All firings of enabled transitions are initiated in the same instants of time in which the transitions become enabled (although some enabled transition cannot initiate their firing; for example, all transitions in a free–choice class can be enabled, but only one can fire). If, during the firing period of a transition, the transition becomes enabled again, a new, independent firing can be initiated, which will 'overlap' with the other firing(s). There is no limit on the number of simultaneous firings of the same transition (sometimes this is called 'infinite firing semantics'). Similarly, if a transition is enabled 'several times' (i.e., it remains enabled after initiating a firing), it may start several independent firings in the same time instant.

In timed nets, the initiated firings continue until their terminations. Sometimes, however, an initiated firing should be discontinued, as in the case of modeling processes with preemptions; if a lower–priority job is executing on a processor, and a higher–priority job needs the same processor for its execution, the execution of the lower–priority job must be suspended, and the processor allocated to the higher–priority job to allow its execution. The preempted job can continue only when the higher–priority job is finished (and no other higher–priority job is waiting). Another extension to the basic model in needed to 'interrupt' firing transitions; a special type of inhibitor arcs, called interrupt arcs, can be used for this purpose. If, during the firing period of a transition, any place connected with this transition by an interrupt arc (i.e., the place is called an interrupting place) receives a token, the firing discontinues, and the tokens removed from the transition's input places at the beginning of firing, are returned to these places (if there are several firings of the transition, the least recent one is discontinued; if there are several interrupting tokens, the corresponding number of least recent firings is discontinued). Moreover, a marked interrupting place disables transition's firings in the same way as inhibitor arcs do. Formally, the set of interrupt arcs, $C$, is added to the structure of the net as a subset of the set of inhibitor arcs, so $\mathcal{N} = (P, T, A, B, C, w)$, $C \subseteq B$. It should be noted that an effect similar to an 'interruption' of a firing transition can be obtained by using a more complicated net with an inhibitor arc (Example 1 illustrates the idea), so interrupt arcs are not a necessary extension; it is rather a convenient addition that simplifies the modeling process.

In timed nets, the firing times of some transitions may be equal to zero, which means that the firings are instantaneous; all such transitions are called immediate (while the other are called timed). Since the immediate transitions have no tangible effects on the (timed) behavior of the model, it is convenient to 'split' the set of transitions into two parts, the set of immediate and the set of timed transitions, and to fire first the (enabled) immediate transitions, and then (still in the same time instant), when no more immediate transitions are enabled, to start the firings of (enabled) timed transitions. It should be noted that such a convention effectively introduces the priority of immediate transitions over the timed ones, so the conflicts of immediate and timed transitions should be avoided. Also, the free–choice classes of transitions must be 'uniform', i.e., all transitions in each free–choice class must be either immediate or timed.

The firing times of transitions can be either deterministic or stochastic (i.e., described by some probability distribution function); in the first case, the corresponding timed nets are referred to as D–nets, in the second, for the (negative) exponential distribution of firing times, the nets are referred to as M–nets (Markovian nets). In both cases, the concepts of state and state transitions have been formally defined and used in the derivation of different performance characteristics of the model [Zu91].

Analysis of net models can be based on their behavior (i.e., the set of reachable states) or on the structure of the net; the former is called reachability analysis while the latter – structural analysis. Invariant analysis seems to be the most popular example of the structural approach. Structural methods eliminate the derivation of the state space, so they avoid the 'state explosion' problem of reachability analysis, but they cannot provide as much information as the reachability approach does. Quite often, however, all the detailed results of reachability analysis are not really needed, and more synthetic performance measures, that can be provided by structural methods, are quite satisfactory.

Both reachability and structural analyses are based on quite detailed net characterizations. Consequently, only very simple models can be analyzed manually; for more realistic models, software tools for analysis of net models are needed. It is, therefore, not surprising that many different tools have been developed for analysis of a variety of net types [Fe93]. A collection of software tools developed for analysis timed Petri net models, TPN–tools, uses the same internal representation of models and a common 'language' for the description of modeling nets. This report describes the structure of model specifications (i.e., the 'input language'), provides several examples of net descriptions, and discusses the internal representation of net models.

## TPN DESCRIPTIONS

Net descriptions are 'transition oriented', i.e., nets are specified as collections of transitions, and each transition contains all parameters associated with it.

The syntax of model descriptions, in the BNF notation, is as follows:

```
<model-descr> ::= <color-list> <net-class> <net-descr> <imarking>
<color-list> ::= <colors> | <empty>
<net-class> ::= <class> | <empty>
<net-descr> ::= <net-header> ( <transitions> )
<net-header> ::= Mnet | Dnet | net
<transitions> ::= <transition> | <transitions> ; <transition>
<transition> ::= <t-header> = <input-output-list>
               | <t-header> <occurrence-list>
<t-header> ::= <t-indent> <type> <time> <prob>
<occurrence-list> ::= <occurrence> | <occurrence-list> , <occurrence>
<occurrence> ::= { <o-header> = <input-output-list> }
<o-header> ::= <o-name> <type> <time> <prob>
<t-ident> ::= # <integer> | # <name>
<o-name> ::= <name> | <empty>
<type> ::= :D | :M | :X | <empty>
<time> ::= * <rational> | <empty>
<prob> ::= , <rational> | , <integer> / <integer> | , <ref> | <empty>
<rational> ::= <integer> | <integer> . <integer>
<ref> ::= [ <place_id> ] | [ <place_id> : <color> ]
<input-output-list> ::= <input-list> | <input-list> / <output-list>
<input-list> ::= <arc> | <input-list> , <arc>
<output-list> ::= <arc> | <output-list> , <arc>
<arc> ::= <place-id> | <place-id> - <color> | <place-id> : <weight> <color>
<place-id> ::= <integer> | <name>
<weight> ::= <integer>
<color> ::= <name> | <empty>
<name> ::= <letter> | <name> <letter> | <name> <digit> | <name> _
```

The type of the net can be indicated in the net header or in the `class` directive:

```
<class> ::= class = D ; | class = M ;
```

The type of a transition or an occurrence (M–timed, D–timed) can also be indicated by the `type` elements; such a specification overrides the net type. The specification `X` indicates the type opposite to the one indicated for the net.

For occurrences without `type`, `time`, or `prob` elements, the values of `type`, `time` and `prob` specified for the transition are used. Transitions and occurrences with empty `time` elements denote immediate transitions and occurrences, i.e., transitions and occurrences with `time` equal to 0.

Probability element `prob` specifies the free-choice probabilities of occurrences or relative frequencies of conflicting occurrences. Empty element `prob` is equivalent to probability equal to 1. The form `<integer>/<integer>` is provided as a convenient way of specifying fractional values.

Marking–dependent relative frequencies are indicated by place/color references `ref` of the `prob` element. During conflict resolution, the number of (colored) tokens in the place indicated by `ref` is used as the relative frequency of transition/occurrence firings. Usually `ref` is one of the transition/occurrence's input places.

Arcs without weight are equivalent to arcs with weight equal to 1. Inhibitor arcs are specified as arcs with weight equal to 0, and interrupt arcs are indicated by the "`-`" symbol following the place identifier.

All colors used in net descriptions must be declared in the list of colors. This list must precede the net description:

```
<colors> ::= color ( <color-list> ) ;
<color-list> ::= <color> | <color-list> , <color>
<color> ::= <name>
```

The initial marking function is specified as a list of marked places:

```
<imarking> ::= mark ( <marking-list> ) ;
<marking-list> ::= <marked-place> | <marking-list> , <marked-place>
<marked-place> ::= <place> | <place> : <count> <color>
<count> ::= <integer>
<color> ::= <name> | <empty>
```

Marked places without the `count` element are equivalent to places with the value of `count` equal to 1.

**Example 1.** A Petri net model of a simple protocol with a timeout mechanism is shown in Fig.1.

The token in $p_1$ represents a message to be sent from a 'sender' ($p_1$) to a 'receiver' ($p_3$) and confirmed by an acknowledgement sent back to the sender. The message is sent by firing $t_1$, after which a single token is deposited in $p_2$ (the message) and in $p_5$ (the timeout). Enabled $t_2$ and $t_6$ can start their firings concurrently; firing time of $t_2$ represents the 'communication delay' of sending a message, and that of $t_6$, the timeout time. When the firing of $t_2$ is finished, a token is deposited in $p_3$, the receiver. $p_3$ is a free-choice place, so $t_3$ and $t_4$ are enabled simultaneously, but only one of them can fire; the random choice is characterized by 'choice probabilities' assigned to $t_3$ and $t_4$. $t_3$ represents (in a simplified

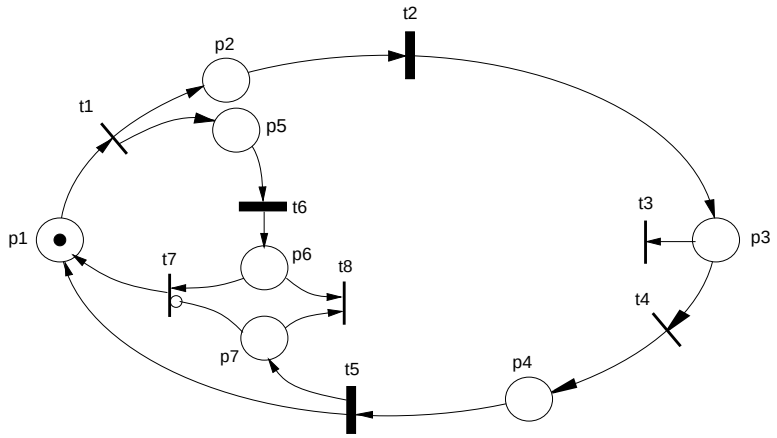


Fig.1. A simple protocol with a timeout.

way) the loss or distortion of the message or its acknowledgement; it $t_3$ is selected for firing (according to its free–choice probability), the token is removed from $p_3$ as well as from the model ($t_3$ is a 'token sink'). In such a case the timeout transition $t_6$ finishes its firing with no token in $p_7$, so the inhibitor arc $(p_7, t_7)$ enables $t_7$, and its firing regenerates the lost token in $p_1$, so the message can be 'retransmitted'. If the message is received correctly, $t_4$ is selected for firing rather than $t_3$, and after another 'communication delay' (modeled by $t_5$) tokens are deposited in $p_7$ and $p_1$ (so another message can be sent to the receiver). The token in $p_7$ waits until the timeout transition $t_6$ finishes its firing, and then removes the timeout token by firing $t_8$ ($t_7$ is disabled in this case by the inhibitor arc).

**Note:** The transition $t_4$ may seem redundant in this model but in fact it is required due to the restriction that all free–choice classes of transitions must be uniform, i.e., each free–choice class must contain either immediate or timed transitions, but not both.

All immediate transitions (i.e., transitions with zero firing time) are represented by (thin) bars, while timed transitions are represented by (black) rectangles. The firing times of timed transitions are selected in such a way that the timeout time ($t_6$) is greater than the sum of the delays of sending a message ($t_2$) and an acknowledgement ($t_5$). In the following description, the immediate transitions are indicated by the default zero firing times:

```
Dnet(#1=1/2,5;
     #2*5=2/3;
     #3,1/10=3;
     #4,9/10=3/4;
     #5*5=4/1,7;
     #6*15=5/6;
     #7=6,7:0/1;
     #8=6,7)
mark(1);
```

The timeout mechanism shown in Fig.1 can be simplified by using an interrupt arc, as shown in Fig.2 (interrupt arcs are indicated by black dots instead of the arrowheads). After

firing $t_1$, a token deposited in $p_5$ immediately starts the firing of the 'timeout' transition $t_6$ (the interrupt arc $(p_6, t_6)$ inhibits new firings only when $p_6$ is marked). If the token reaching $p_3$ is 'lost' (by firing $t_3$), the timeout transition finishes its firing and regenerates the lost token in $t_1$, so another cycle of sending a message can begin. If the message is received correctly (i.e., $t_4$ is chosen for firing), the acknowledgement token enables and fires $t_5$ and, after a 'communication delay', deposits tokens in $p_1$ and in $p_6$. A token in $p_6$ discontinues the firing of $t_6$, so the timeout token removed from $p_5$ at the beginning of $t_6$'s firing is returned to $p_5$, and subsequently removed by firing the immediate transition $t_7$.

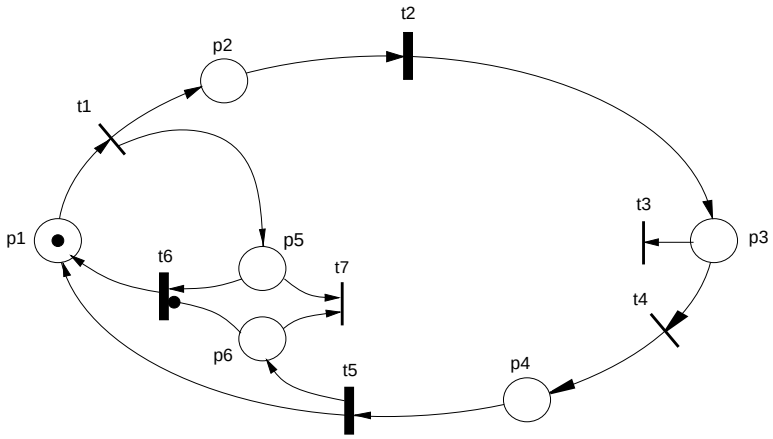


Fig.2. A timeout model with an interrupt arc.

```
Dnet(#1=1/2,5;
     #2*5=2/3;
     #3,1/10=3;
     #4,9/10=3/4;
     #5*5=4/1,6;
     #6*15=5,6-/1;
     #7=5,6)
mark(1);
```

**Example 2.** A colored Petri net model of 'five dining philosophers' is shown in Fig.3. The philosophers are represented by token colors A, B, C, D and E, while forks by colors m, n, o, p and q.

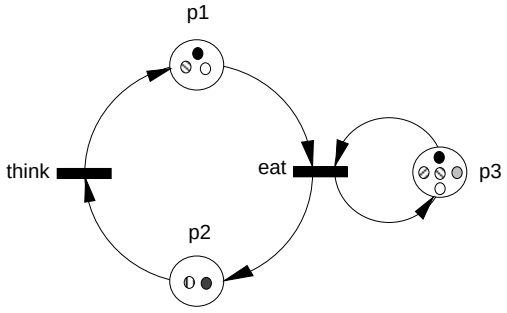


Fig.3. A colored net model of 'dining philosophers'.

The net shown in Fig.3 outlines a model of a single philosopher, so the use of colored tokens is essential for this model. The use of colors and the conflicts created by sharing forks can be described by the following 'connectivity matrix', in which rows correspond to token colors assigned to places, and columns – to transition occurrences; the elements "`+1`" represent arcs from transition occurrences to places while elements "`-1`", arcs form places to transition occurrences (the values of the elements are the weights assigned to arcs, in this case they are equal to `1`). The elements of the matrix are sets rather than expressions, so the 'loops' (e.g. place $p_3$ and transition *eat* in Fig.3) can also be represented:

| | *think* | | | | | *eat* | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | U | V | W | X | Y | U | V | W | X | Y |
| $p_1$:A | `+1` | | | | | `-1` | | | | |
| B | | `+1` | | | | | `-1` | | | |
| C | | | `+1` | | | | | `-1` | | |
| D | | | | `+1` | | | | | `-1` | |
| E | | | | | `+1` | | | | | `-1` |
| $p_2$:A | `-1` | | | | | `+1` | | | | |
| B | | `-1` | | | | | `+1` | | | |
| C | | | `-1` | | | | | `+1` | | |
| D | | | | `-1` | | | | | `+1` | |
| E | | | | | `-1` | | | | | `+1` |
| $p_3$:m | | | | | | `-1,+1` | `-1,+1` | | | |
| n | | | | | | | `-1,+1` | `-1,+1` | | |
| o | | | | | | | | `-1,+1` | `-1,+1` | |
| p | | | | | | | | | `-1,+1` | `-1,+1` |
| q | | | | | | `-1,+1` | | | | `-1,+1` |

For example, the last column describes the occurrence Y of transition *eat*. The occurrence is enabled (elements "`-1`") by a single token of color E in place $p_1$ (philosopher E), one token of color p and one of color q in place $p_3$ (E's 'right' and 'left' forks). Firing this occurrence removes these tokens from $p_1$ and $p_3$ and when the eating is finished (the firing time of this occurrence), one token of color E is deposited in $p_2$ (philosopher E is going to think), and single tokens of color p and q are returned to $p_3$ (the two forks are returned). It should be observed that the name of the occurrence, Y, is rather irrelevant; since this occurrence models the behavior of philosopher E, it could be named E as well.

The following model description is a rather straightforward transcription of this connectivity matrix (column by column); it is assumed that the thinking times as well as eating times of all philosophers are exponentially distributed (M–timed model) with the average values equal to 5 and 2, respectively:

```
color(A,B,C,D,E,m,n,o,p,q);
Mnet(#think*5{U=2:1A/1:1A},
      {V=2:1B/1:1B},
      {W=2:1C/1:1C},
      {X=2:1D/1:1D},
      {Y=2:1E/1:1E};
```

```
#eat*2{U,1=1:1A,3:1q,3:1m/2:1A,3:1q,3:1m},
      {V,1=1:1B,3:1n,3:1m/2:1B,3:1n,3:1m},
      {W,1=1:1C,3:1n,3:1o/2:1C,3:1n,3:1o},
      {X,1=1:1D,3:1o,3:1p/2:1D,3:1o,3:1p},
      {Y,1=1:1E,3:1q,3:1p/2:1E,3:1q,3:1p})
mark(1:1A,1:1B,1:1C,2:1D,2:1E,3:1m,3:1n,3:1o,3:1p,3:1q);
```

**Example 3.** The net shown in Fig.4 is a simple illustration of 'marking–dependent' conflict resolutions. The net represents an interactive system executing two classes of jobs, say class–A and class–B jobs, with random selection of jobs from a common pool of jobs waiting for execution. No priorities and no queueing is assumed, so the probability of selecting a class–A job is determined by the ratio of the number of waiting class–A jobs to the total number of waiting jobs.

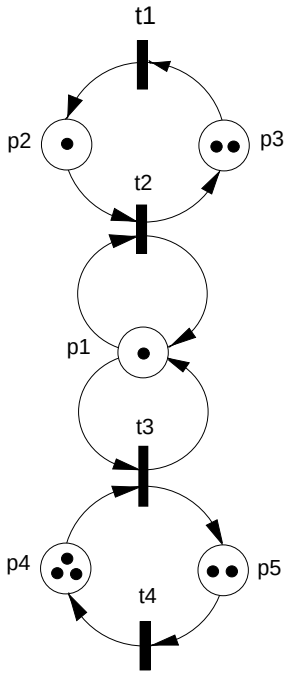


Fig.4. A model of a processor executing two classes of jobs.

In the model, $p_1$ represents the (idle) processor. Execution of class–A jobs is represented by $t_2$, and class-B jobs – by $t_3$. $t_1$ models the 'thinking time' for class–A jobs, and $t_4$ – the same for class–B jobs. $p_2$ is the pool of waiting class–A jobs, $p_4$ – the pool of waiting class–B jobs. $t_2$ and $t_3$ are in conflict because of sharing $p_1$, and the relative frequencies of $t_2$ and $t_3$ firings can be determined by the numbers of tokens in $p_2$ and $p_4$, respectively.

The description of the model is as follows:

```
Mnet(#1*5=3/2;
     #2*3,[2]=1,2/1,3;
     #3*2,[4]=1,4/1,5;
     #4*8=5/4)
mark(1,2,3:2,4:3,5:2);
```

## PNN DESCRIPTIONS

Another approach to net description was proposed in Petri Net Newsletter [BVV88]; although it can be used for colored nets as well, the presented version is for non-colored nets only (for color nets, the attributes should be extended to cover token colors and transition occurrences):

```
<net-descr> ::= pnnet <net-header>;
                <place-decl>
                <trans-decl>
                <arc-decl>
                <marking-decl>
                end;
<net-header> ::= <name> | <name> : <net-type>
<net-type> ::= M | D
<place-decl> ::= places <pdecl-list> ;
<pdecl-list> ::= <pdecl> | <pdecl-list> ; <pdecl>
<pdecl> ::= <p-list> <place-attr>
<p-list> ::= <p-ident> | <p-list> , <p-ident>
<p-ident> ::= <name> | <integer>
<place-attr> ::= <empty>
<trans-decl> ::= transitions <tdecl-list> ;
<tdecl-list> ::= <tdecl> | <tdecl-list> ; <tdecl>
<tdecl> ::= <t-list> <trans-attr>
<t-list> ::= <t-ident> | <t-list> , <t-ident>
<t-ident> ::= <name> | # <integer>
<trans-attr> ::= <empty> | : <time> | : <prob> | : <time> <prob>
<time> ::= <number>
<prob> ::= ( <number> ) | [ <p-ident> ]
<arc-decl> ::= arcs <adecl-list>
<adecl-list> ::= <adecl> | <adecl-list> ; <adecl>
<adecl> ::= <arc-list> <arc-attr>
<arc-list> ::= <arcs> | <arc-list> , <arcs>
<arcs> ::= ( <p-list> ; <t-list> ) | ( <t-list> ; <p-list> )
<arc-attr> ::= <empty> | <integer> | -
<marking-decl> ::= marking <mark-list>
<mark-list> ::= <mark> | <mark-list> ; <mark>
<mark> ::= <p-list> | <p-list> : <count>
<count> ::= <integer>
```

The attributes (place attributes, transition attributes, arc attributes) are associated with the corresponding lists of places, transitions or arcs rather than a single place, transition or arc; elements of these lists are separated by commas. Lists with attributes are declarations, which are separated by semicolons. Declarations are grouped in sections which begin with a keyword (`places`, `transitions`, `arcs`).

Attributes of places are not used in this version of PNN descriptions.

The firing times (attribute `time`) and free–choice probabilities (attribute `prob`) can thus be associated with lists of transitions. Similarly the weights of arcs.

Arcs are described by a pair of lists, a list of places and a list of transitions. If both lists contain more than one element, all possible combinations of list elements are specified.

**Example.** The description of the net shown in Fig.1 can be as follows:

```
pnnet timeout:D;
places 1,2,3,4,5,6,7;
transitions #1,#7,#8;
     #2:5.0;
     #3:(0.1);
     #4:(0.9);
     #5:5.0;
     #6:15.0;
arcs (1;#1),(#1;2,5);
     (2;#2),(#2;3);
     (3;#3);
     (3;#4),(#4;4);
     (4;#5),(#5;1,7);
     (5;#6),(#6;6);
     (6;#7);(7;#7):0;(#7;1);
     (6,7;#8);
marking 1;
end;
```

and that of Fig.4, as follows:

```
pnnet conflict:M;
places 1,2,3,4,5;
transitions #1:5.0;
     #2:3.0[2];
     #3:2.0[4];
     #4:8.0;
arcs (3;#1),(#1;2);
     (1,2;#2),(#2;1,3);
     (1,4;#3),(#3;1,5);
     (5;#4),(#4;4);
marking 1,2; 3:2; 4:3; 5:2;
end;
```

## INTERNAL REPRESENTATION

Internally, a (marked) Petri net is represented by three linked lists:

- a list of place descriptors,
- a list of transition descriptors, and
- a list representation of the initial marking function.

Each place descriptor contains an identifier of the place and a pointer to a list of colors associated with the place:

```
struct p_list {
     p_list             *next;
     int                pid;      /* place identifier */
     c_list             *clist;   /* list of colors */
};
```

For places identified by names, the names are stored in a separate list (of names) and places are identified by unique (negative) numbers. The same mechanism is used for occurrence names, transition names, etc.

Each color descriptor contains a global color identifier (unique in the set of colored places), a (local) color number, a link to the "home" place and a list of occurrences which share this colored place (this list is used for analysis of free–choice and conflict classes); for simulation, there is also a counter of tokens entering the place, the current number of (colored) tokens and its reference time (i.e., the time of the most recent change of the number of tokens), and the total (cumulative) waiting time of all tokens of this color:

```
struct c_list {
     c_list             *next;
     p_list             *pnode;   /* "home" place */
     int                cid;      /* global color identifier */
     int                num;      /* local color number */
     int                value;    /* current marking (for simulation) */
     int                count;    /* token count (for simulation) */
     double             total;    /* total waiting time (for simulation) */
     double             rtime;    /* reference time (for simulation) */
     f_list             *flist;   /* list of output occurrences */
     f_list             *xlist;   /* list of interrupted occurrences */
};
```

The lists of occurrences `flist` and `xlist` are built of simple descriptors which contain links to the occurrence descriptors and the weights of the connecting arcs:

```
struct f_list {
     f_list             *next;
     o_list             *onode;   /* occurrence descriptor */
     int                val;      /* arc weight */
};
```

Each transition descriptor contains an identifier of the transition and a list of occurrences:

```
struct t_list {
     t_list             *next;
     int                tid;      /* transition identifier */
     o_list             *olist;   /* list of occurrences */
};
```

Each occurrence descriptor contains a global occurrence identifier (unique in the net), a (local) occurrence number, a link to the "home" transition descriptor, free-choice or conflict

class identifier, the type of occurrence (characterizing the timing information), the (average) firing time, the free-choice probability or the relative frequency of firings (for the resolution of conflicts), a link to a color in the place list for marking–dependent conflict resolutions, and two linked lists of arc descriptors, one for incoming arcs and the other for outgoing arcs; moreover, all conflicting occurrences are linked by the `conf` field; finally, for simulation, there is a count of the number of firings and the total (cumulative) firing time:

```
struct o_list {
     o_list             *next;
     o_list             *conf;    /* conflict link */
     t_list             *tnode;   /* "home" transition */
     int                oid;      /* global occurrence identifier */
     int                num;      /* local occurrence number */
     int                class;    /* free-choice class identifier */
     int                type;     /* firing type (D, M, X) */
     double             time;     /* average firing time */
     double             prob;     /* free-choice probability or frequency */
     c_list             *mark;    /* link for marking-dependent conflicts */
     int                count;    /* firing count (for simulation) */
     double             total;    /* cumulative firing time (simulation) */
     m_list             *input;   /* input list */
     m_list             *output;  /* output list */
};
```

Each arc descriptor contains a link to a (colored) place and the weight of the arc between a (colored) place and an occurrence (or an occurrence and a place):

```
struct m_list {
     m_list             *next;
     c_list             *cnode;   /* color descriptor */
     int                val;      /* arc weight */
};
```

For inhibitor arcs the value of `val` is equal to zero; for interrupt arcs it is negative.

The initial marking function is represented by a list of `m_list` descriptors in which `val` represents the number of tokens, and the color is indicated by the `cnode` link.

The list of transitions is ordered with respect to transition identifiers; the occurrence lists are ordered with respect to occurrence identifiers; each place list is ordered with respect to place identifiers; and each color list is ordered with respect to color identifiers (global and local identifiers are consistent with respect to ordering).

A global flag `Net_type` indicates whether the net is M-timed (by the value 'M') or D-timed (by the value 'D').

A fragment of internal representation of the model shown in Fig.3 is presented in Fig.5 where only occurrence W is shown in greater detail, and many links are ignored in order to simplify the illustration.

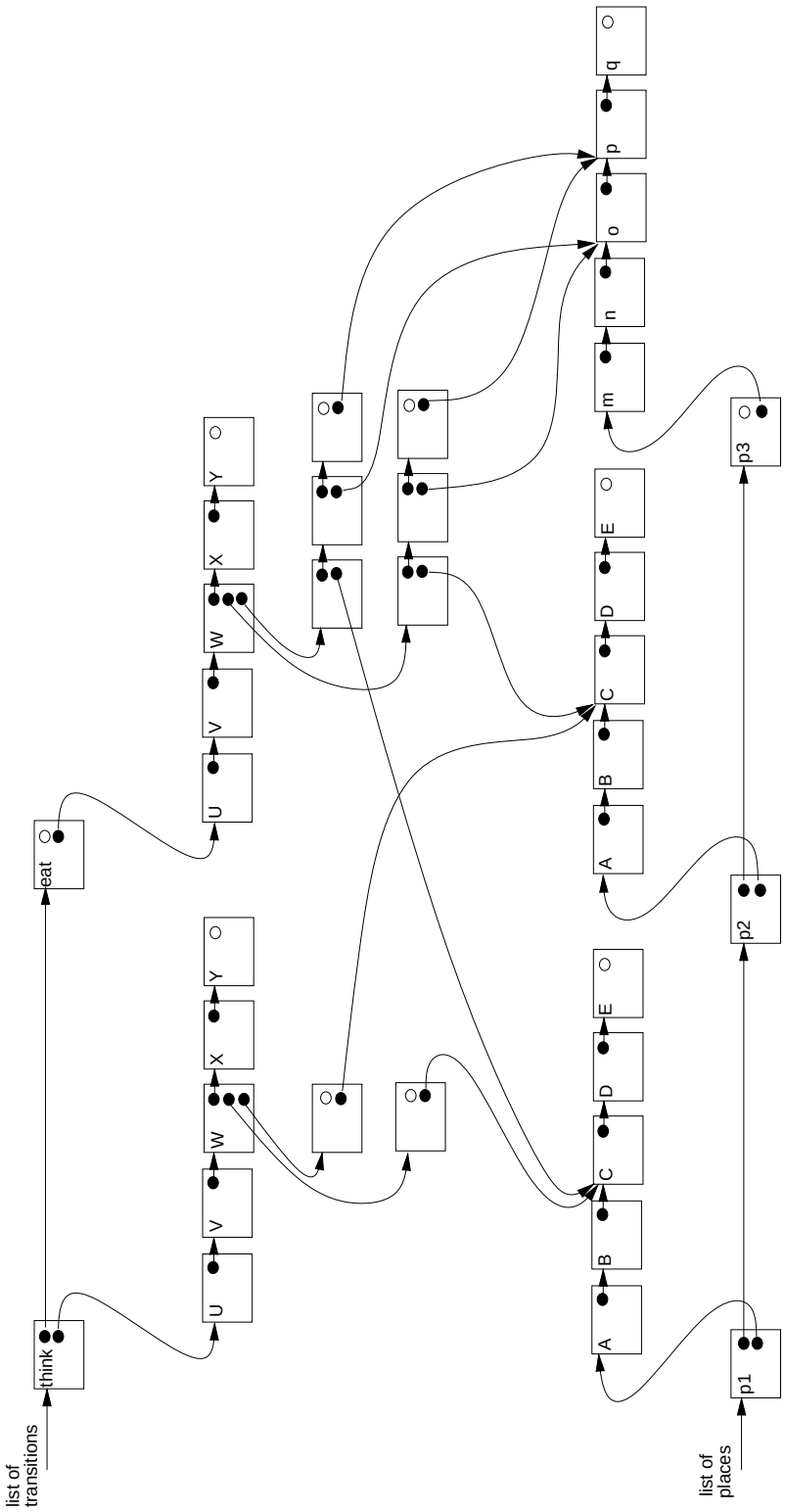


Fig.5. An outline of internal representation of the model shown in Fig.3 (only occurrence W is shown in greater detail and many links are ignored).

## CONCLUDING REMARKS

A simple language for textual description of timed colored Petri net models is presented. It provides compact and quite flexible descriptions of models. The descriptions are straightforward for processing as they are structured in a way similar to the internal representation of models. A (simplified) implementation of an alternative approach to specification of net models is also included. Implementations of other description methods can easily be added in the future.

Presented specification languages are rather 'low–level' which means that the model description is at a rather detailed level. It is anticipated that a more convenient graphical user interface, of the type used by GreatSPN [Ch92] or DSPNexpress [Li92], will be developed (or adopted) at some later time.

For colored net models, the current version of model specifications requires all occurrences to be described explicitly. A more convenient (and flexible) approach allows to use symbolic variables and implied rather than explicit occurrences. All such features can easily be added to the TPN–tools collection.

## References

[Ag79] Agerwala, T., "Putting Petri nets to work"; IEEE Computer Magazine, vol.12, no.12, pp.85-94, 1979.

[AF73] Agerwala, T., Flynn, M., "Comments on capabilities, limitations and 'correctness' of Petri nets"; Proc. of the First Annual Symp. on Computer Architecture, pp.81-86, 1973.

[BVV88] Berthelot, G., Vautherin, J., Vidal–Naquet, G., "A syntax for the description of Petri nets"; Petri Net Newsletter, no. 29, Gesellschaft fuer Informatik, pp.4-15, 1988.

[Ch92] Chiola, G., "GreatSPN 1.5 software architecture"; in: "Computer Performance Evaluation – Modeling Techniques and Tools", Balbo, G., Serazzi, G. (eds.), pp.121-136, Elsevier 1992.

[Fe93] Feldbrugge, F., "Petri net tool overview 1992"; in: "Advances in Petri Nets 1993" (Lecture Notes in Computer Science 674), Rozenberg, G., (ed.), pp.169-209, Springer Verlag 1993.

[Je87] Jensen, K., "Coloured Petri nets"; in: "Advanced Course on Petri Nets 1986" (Lecture Notes in Computer Science 254), G. Rozenberg (ed.), pp.248–299, Springer Verlag 1987.

[Li92] Lindemann, Ch., "DSPNexpress: a software package for the efficient solution of deterministic and stochastic Petri nets"; in: "Computer Performance Evaluation – Modeling Techniques and Tools", Balbo, G., Serazzi, G. (eds.), pp.9-20, Elsevier 1992.

[Mu89] Murata, T., "Petri nets: properties, analysis and applications"; Proceedings of IEEE, vol.77, no.4, pp.541–580, 1989.

[Pe81] Peterson, J.L., "Petri net theory and the modeling of systems", Prentice–Hall 1981.

[Re85] Reisig, W., "Petri nets - an introduction" (EATCS Monographs on Theoretical Computer Science 4); Springer Verlag 1985.

[Va82] Valk, R., "Test on zero in Petri nets"; in: "Applications and Theory of Petri Nets" (Informatik–Fachberichte 52), Girault, C., Reisig, W. (eds.), pp.193-197, Springer Verlag 1982.

[Zu91] Zuberek, W.M., "Timed Petri nets – definitions, properties and applications"; Microelectronics and Reliability (Special Issue on Petri Nets and Related Graph Models), vol.31, no.4, pp.627–644, 1991 (available through anonymous `ftp` at `ftp.cs.mun.ca` as `/pub/publications/91-MaR.ps.Z`).