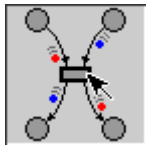
Ministry of Transport and Communications of Ukraine
Odessa National Academy of Telecommunication by the name of A.S. Popov

Department of Communication Networks



D.A. Zaitsev, T.R. Shmeleva

Simulating of Telecommunication Systems with CPN Tools

Students' book on the course
«Mathematical Modeling of Information Systems»
for teaching of masters in communications

APPROVED by
Council of faculty of
Information networks
Transaction № 5 of
16.11.2006 year

Odessa 2006

UDK 621.39, 004.7

Issue plan 2005/2006

Reviewers: Prof. V.A. Kudriashova,
k.t.n, docent I.A. Tregubova

Compilers: k.t.n., docent D.A. Zaitsev,
aspirant T.R. Shmeleva

The description of the simulation system CPN Tools features is presented. The system was developed in the University of Aarhus (Denmark) and is used for modeling of telecommunication systems and networks in the course «Mathematical Modeling of Information Systems». For case study an example of switched Ethernet model was chosen.

Affiliated on
meeting of
Communication Networks
Department
Transaction № 4 of
10.11.2006 year

Content

Introduction…… 5
1. Class of Petri Nets Implemented in CPN Tools…… 5
1.1. Petri Net Graph and CPN ML…….. 5
1.2. An Example Consideration…….. 7
2. Destination and Basic Functions of CPN Tools…… 10
2.1. Destination of CPN Tools…… 10
2.2. Basic Functions of CPN Tools……. 10
3. Organization of CPN Tools interface…… 11
3.1. Areas of Main Window…… 11
3.2. Working with Tools……. 12
3.3. Context-sensitive Menus…….. 13
3.4. Model's Structure……. 14
3.5. Organization of Help System…… 15
3.6. Reflecting Feedback of CPN Tools…….. 16
4. Tool Box of CPN Tools…… 18
4.1. Net Tools…….. 18
4.2. Create Tools…….. 19
4.3. Simulation Tools……... 21
4.4. Overview of Other Tools…….. 23
5. Basics of CPN ML…… 23
5.1. Simple Color Sets……. 24
5.2. Compound Color Sets…….. 25
5.3. Declaration of Variables and Constants…… 27
5.4. Functions……... 27
5.5. Random Functions…… 29
5.6. Multi-sets…….. 30
5.7. Timed Multi-sets……... 31
6. The Language of Models' Description…….. 31
6.1. Place Inscriptions…….. 32
6.2. Arc Inscriptions……. 32
6.3. Transition Inscriptions…….. 34
7. Peculiarities of Timed Nets in CPN Tools…… 36
8. Working with Nets' Fragments…….. 38
9. Fusion Places…… 40
10. Hierarchical Models' Construction…….. 42
10.1. Basics of Transition Substitution…… 42
10.2. Bottom-up Development……. 44
10.3. Top-down Development……. 44
11. Analyzing a CP-net…….. 45
11.1. Debugging of Models……. 45
11.2. State Space Analysis……... 46
11.3. Simulation of Net Behavior…… 48
11.4. Measuring Fragments……. 49

12. Additional Features of CPN Tools……... 51
12.1. Unions……. 51
12.2. Lists……. 52
Appendices: An Evaluation of Network Response Time using a Colored Petri Net Model of Switched LAN…… 54
A1. Switched LAN…….. 54
A2. Model of LAN…….. 54
A3. Model of Switch…… 56
A4. Models of Workstation and Server……... 57
A5. Model of Measuring Workstation……. 59
A6. Evaluation Technique……... 60
A7. Parameters of Model……. 60
References…….. 62

Introduction

CPN Tools is a special simulation system which uses the language of Petri nets for models' representation. The system was developed in University of Aarhus in Denmark and is distributed free of charge for noncommercial organizations via web site http://www.daimi.au.dk/CPNTools/. The level of service allows the classification of CPN Tools as an enterprise system. It was used in a lot of real-life projects especially in the area of telecommunications. Recently Nokia Corporation is applying CPN Tools for model-driven development of new generation of its mobile phones.

1. Class of Petri Nets Implemented in CPN Tools

CPN Tools proposes very powerful class of Petri nets for models' description. According to the standard classification such nets are named hierarchical timed colored Petri nets. It was proved that they are equivalent to Turing machine and constitutes a universal algorithmic system. So an arbitrary object can be specified using this class of nets.

Simplest concept of colored Petri net uses different types of tokens. The type of a token is specified by natural number and represented visually as a color: 1-red, 2-blue, 3-green, etc. The concept of a colored Petri net of CPN Tools is more complicated. Such nets are often called generalized colored nets because the type of token is described as an abstract data type like in programming languages. The term "colored" remains historically but it is very difficult to represent such "colors" visually now.

Timed Petri nets use the concept of model's time to represent the durations of actions in real-life objects. In spite of classical Petri net where the firing of a transition occurs instantly the firing of a transition in timed net is concerned with definite duration or timed delay. It allows the analysis of timed characteristics of real-life objects, for instance, response time as a characteristic of network's QoS.

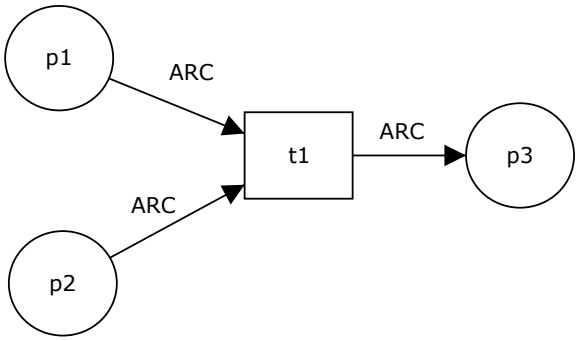
Hierarchical nets provide construction of complicated models. In such nets an element may be represented by another net. In CPN Tools a transition may be substituted by an additional net. So we have a nested construction: net inside net. The number of hierarchy levels has no principal limitations. Notice that, the idea is wide common for programming languages where procedures are used to maintain the complexity.

1.1. Petri Net Graph and CPN ML

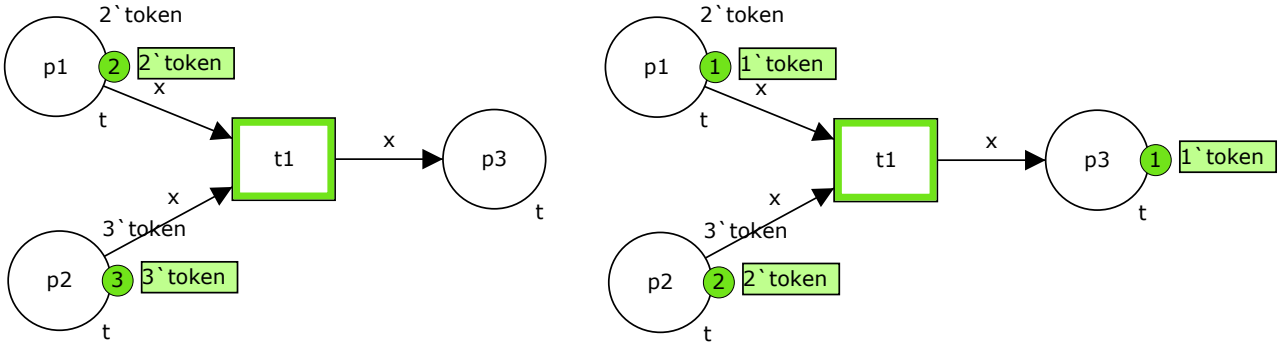
In CPN Tools the language of models' description constitutes a combination of Petri net graph and programming language CPN ML (Markup Language).

The graph of Petri net is a bipartite directed graph. It consists of vertices of two types: places drawn as circles or ovals and transitions drawn as bars:

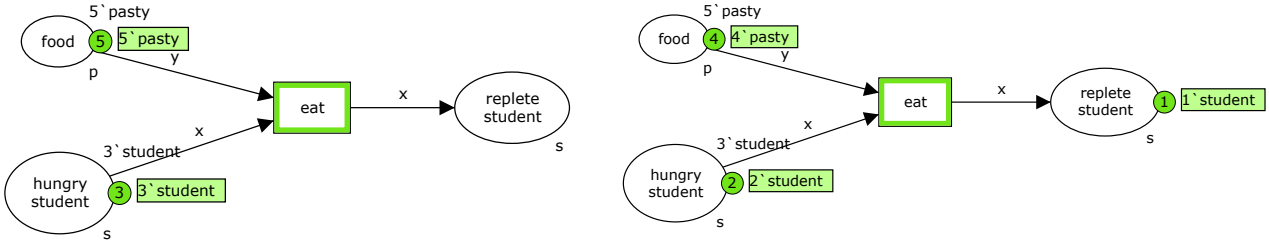
Arcs are used to connect places and transition:



In Petri nets a concept of a token is also considered. Token is a dynamic object that is located inside place and is moved as the result of transition firing:



In classical Petri nets all the tokens are elementary and the same. In colored Petri net types of tokens are distinguished. Let us consider an example of pasty consumed by students. We have two kinds of tokens: `student`, `pasty`. A hungry student becomes replete student after eating a pasty:



Really, in CPN Tools special programming language is included for description of attributes of net elements. This language provides declarations of color sets, variables, constants, functions, and procedures. In the above example such declarations were used:

```
colset s=unit with student;
colset p=unit with pasty;
```

```
var x:s;
var y:p;
```

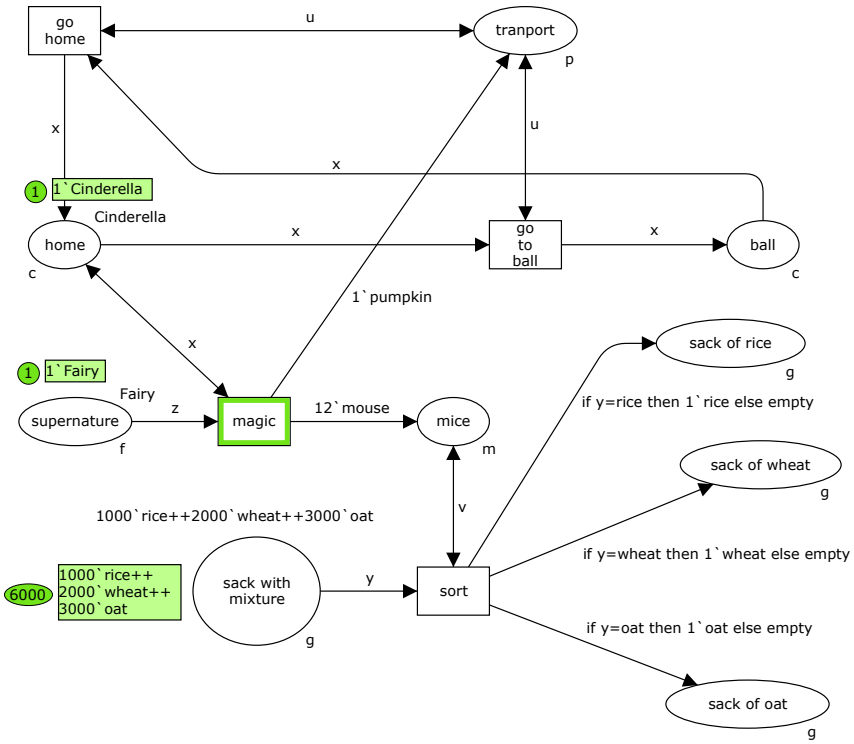
Two color sets were defined: `s` with the member `student` and `p` with the member `pasty`. Places "`hungry student`" and "`replete student`" are of type `s` with tokens `student`. Place `food` is of type `p` with tokens `pasty`. To fire the transition you have to have both: a student and a pasty. The variables `x` and `y` are used to extract tokens from places and to put new token into output place.

The example shows the way the different types of tokens may be processed. In models of telecommunication systems color sets may be more complicated and represent, for Ethernet instance, frames, records of switching table, etc.

In spite of classical Petri nets places, transitions and arcs have their attributes in colored Petri net. In the above example place have the name – "`hungry student`", color set – `t`, initial marking – `3`student` and current marking – `2`student` after consuming a pasty by one of them. Variable `x` allows the choice of an arbitrary student according to the color set of the variable; variable `y` allows the choice of an arbitrary pasty. The same student that was extracted by variable `x` from the place "`hungry student`" will be put in the place "`replete student`" because the output arc of the transition `eat` is inscribed with the same variable `x`.

1.2. An Example Consideration

Let us consider more complicated example for preliminary study of CPN Tools. It was taken from well-known fairytale about Cinderella. Stepmother has said Cinderella to separate grains of different kinds. In this example mice are separating grains while Cinderella is going to the ball.



In this colored Petri net the following declarations of color sets and variables are used:

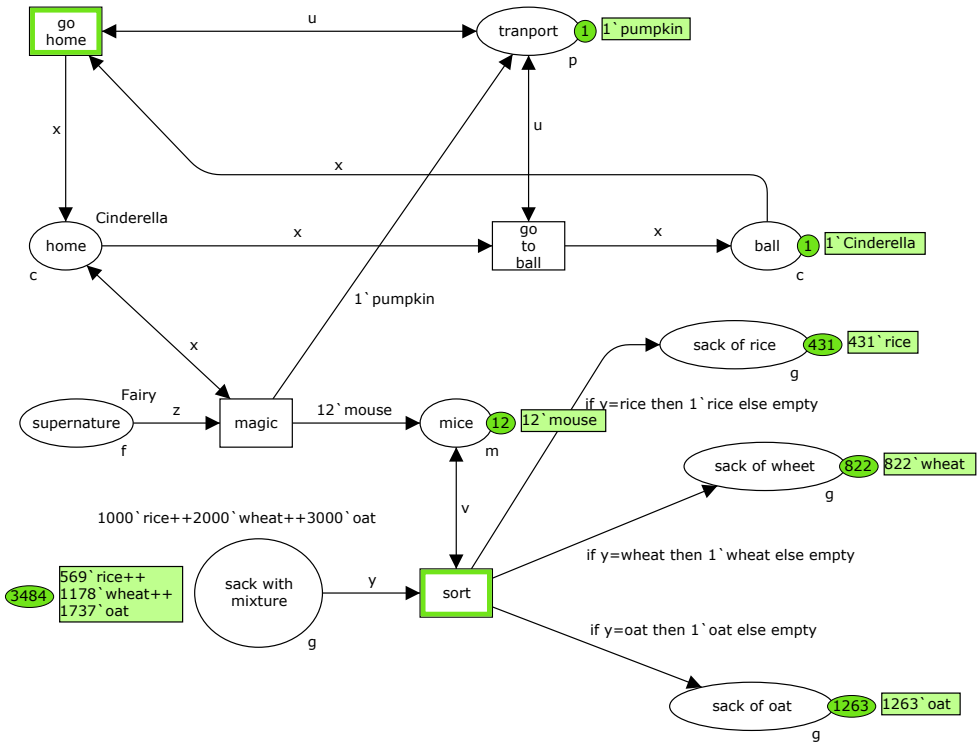
```
colset p=unit with pumpkin;
colset c=unit with Cinderella;
colset g=with rice | wheat | oat;
colset m=unit with mouse;
colset f=unit with Fairy;
var x: c;
var y: g;
var z: f;
var u: p;
var v: m;
```

In this example we have four color sets: `f` with tokens named `Fairy`, `c` with tokens named `Cinderella`, `p` with tokens named `pumpkin` and `g` with three possible kinds of tokens named `rice`, `wheat`, `oat`. In the initial marking the only permitted transition is `magic`; it is highlighted. Talking with Cinderella in the transition `magic`, Fairy creates 12 mice and 1 pumpkin and disappears. The traveling of Cinderella to the ball and sorting of grains are concurrent events and they may occur simultaneously in any order. Pumpkin is used as a resource for transitions "`go to ball`" and "`go home`" to carry Cinderella to the ball and backwards. Mice are used as a resource for transition `sort` to select grains of different kinds.

Let us consider the directions and inscriptions of arcs. In transition `magic Cinderella` is not changed as well as `pumpkin` in transitions "`go to ball`" and "`go home`" so bidirection arcs are used. The other arcs are unidirection. The arc directed from place to transition extracts a token of corresponding color set. The token is extracted according to the inscription of transition's input arc. In this example all the inscriptions are represented by variables of corresponding color set. For instance, input arc of the transition `sort` has the inscription `y`; `y` is a variable of color set `g`; so an arbitrary grain is extracted from the place "`sack with mixture`". More complicated inscriptions of transitions' input arcs will be studied further.

As for the output arcs of transitions, they create new tokens. New token may coincide with any token extracted from input place or may be created anew. For instance, in the transition "`go to ball`" token `Cinderella` is extracted from place `home` by variable `x` in the inscription of input arc and the same token `Cinderella` will be put to place `ball` according to the inscription `x` on the output arc. The transition `magic` is more complicated: `Fairy` disappears after firing of this transition because she is extracted by input arc with inscription `z` and variable `z` is not used in the inscriptions of output arcs; `Cinderella` in place `home` is not changed by transition `magic` because bidirection arc with inscription `x` is used, namely it only checks the presence of token in place `home`; 12 `mice` and 1

pumpkin are created, the corresponding constants are written in the inscriptions of output arcs. Now consider the model after 5000 steps of simulation:



Cinderella arrived to the ball, mice are struggling with their job, Fairy disappeared. The mice have selected 431 grains of rice, 822 grains of wheat and 1263 grains of oat. There are two permitted transitions in this marking: sort and "go home". Let us consider the way of grains' sorting in this model. The transition sort extracts one grain in the variable y but this grain is to be put only into one of places: "sack of rice", "sack of wheat", "sack of oat". The inscriptions of output arcs contain functions that select only the grain of the required kind otherwise the special token empty is chosen. Token empty means "nothing".

Close consideration of the above example reveals a lot of things that does not correspond the original fairytale. For instance, we do not consider time at all and the warning of Fairy about midnight. We recommend you to try this model and find all the lacks. After studying CPN Tools you may construct your own model completely fit to the original fairytale.

Let us notice that the above toy example gives grains of experience for developing models of telecommunication systems and networks. The job of mice looks like a function of network router. Real-life example of switched Ethernet model is put in Appendices. But to understand how it works a lot of knowledge collected in the following sections is required.

2. Destination and Basic Functions of CPN Tools

2.1. Destination of CPN Tools

CPN Tools is aimed to models' design and analysis. It is a vital system in the development of complicated objects in various fields of engineering. It is widely used for production and business management, planning and control of military operations, control of production systems and robots as well as vehicles and missiles. The complete list of real-life applications you will find at home-page of CPN Tools http://www.daimi.au.dk/CPNTools/. CPN Tools is implemented on both MS Windows and Unix platforms now and constitutes in essence a new generation of early used system Design-CPN.

As for telecommunications, CPN Tools is used for specification and verification of protocols, estimation of networks throughput and QoS, design of telecommunication devices and networks. Recently, Nokia Corporation is used CPN Tools in model-driven development of new generations of its mobile phones. This direction is the most fruitful for complicated devices' engineering. Early a model was used only for estimation of devices' or networks' characteristics in the process of their development. In model-driven development an initial simple model is sequentially transformed to the final specification of the system. The process of development constitutes the process of giving more and more details of the real-life system to the model until it becomes as minute as a technical specification required for its production or installation. The advantage of this approach is the possibility for analysis of a system on each stage of its design and estimation of its characteristics. Is it still fit to the requirements? It allows the design of systems that are very close to the optimal because for real-life complicated objects the formal solution of optimization task is quite hard and in the most cases is practically unfeasible.

As for colored timed hierarchical Petri nets of CPN Tools they are a universal algorithmic system so they allow the description of an arbitrary object. Moreover, the language of colored Petri nets is convenient for systems' specification especially for systems with complicated interaction between components. The concept of asynchronous events allows the way of description preserving the natural parallelism of systems' behavior. It is very convenient for further implementation on parallel processors or data-flow architectures of computers.

The most advantage of CPN Tools application is gained when special (hardware or software) processors of Petri nets are used. In this case the final specifications of a system in the form of colored Petri net may be put directly to such a processor. There are a few known types of hardware processors of Petri nets, for instance signal processors in controllers of company Klashka.

2.2. Basic Functions of CPN Tools

Basic functions of CPN Tools consist in:

- creation (editing) of model;

– analysis of models behavior via its simulation;
– creation and analysis of model's state space.

For the creation of models the special graphic editor of colored Petri nets is provided. The editor allows the drawing of Petri nets on the computer's screen and the inputting the attributes of nets' elements and additional declarations written in CPN ML language. Model may consist of a few pages. Pages are connected with each other to provide a hierarchical structure.

For simple enough models the generation of its complete state space (reachability graph) is possible. It is the best way, for instance, for verification of telecommunication protocols. CPN Tools provides creation of state space and automatic report on it where the conclusions about standard properties of Petri nets such as boundedness and liveness are presented. Moreover, special language on the base of CPN ML is provided for description of queries about nonstandard properties of the state space the user is interested in. Unfortunately, for complicated models the state space may be huge and its creation is unfeasible.

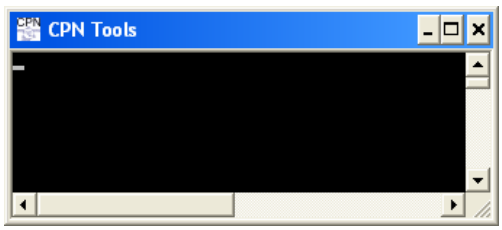
The only way for complicated models analysis is the simulation of its behavior. CPN Tools provides step-by-step simulation for debugging of model as well as automatic simulation of specified numbers of steps. Simulation on big time intervals is the way for statistical analysis of model's behavior. It is useful for estimation of networks' characteristics such as throughput and QoS.

3. Organization of CPN Tools interface

In CPN Tools a new concept of graphical interaction based on MS Open GL features is implemented. It allows the fast inputting and editing of models using tools from tool boxes and context-sensitive menus. The special facility of working with two mouse devices is provided. In this case left mouse is used for interaction with menu and for selection of tools from palettes while right mouse is used for drawing and editing Petri nets.

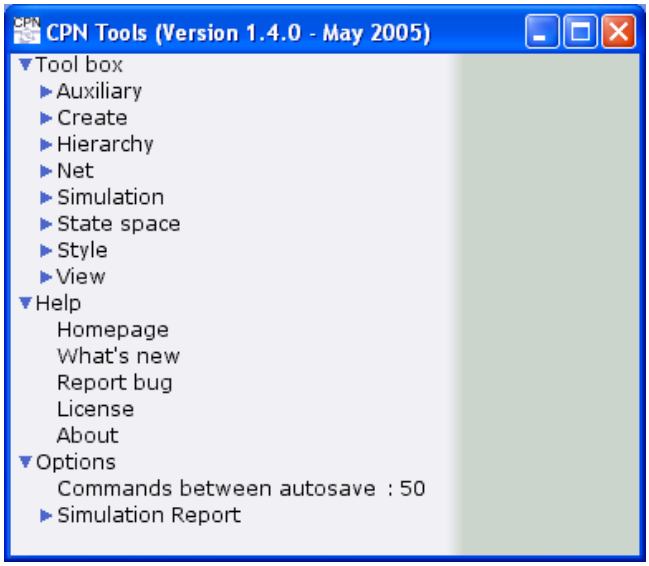
3.1. Areas of Main Window

After the launch two windows of CPN Tools appear on the screen. The first of them is black window; it is auxiliary and serves for output of messages when subprocessed are started:



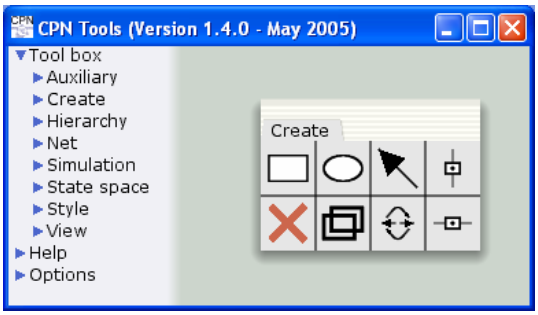
The second window is the main window of CPN Tools:

It contains two areas: workspace – grey and index – white. Index consists of Tools box, Help and Options; below them descriptions of nets are put; in the above example there is no loaded net. In workspace the pages of nets are visualized. There is graphical cursor in the main window for interaction with CPN Tools. Always in the system small triangle means an item that may be opened by clicking on it. For instance, we may open all the items in the index:



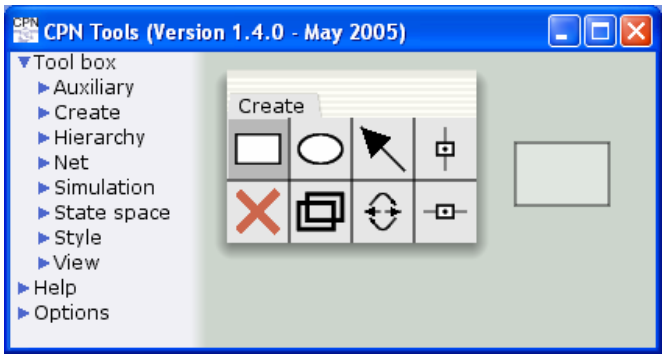
3.2. Working with Tools

The way to open a tool's palette is to drag it with mouse from the index to the workspace. Let us open `Create` tool:



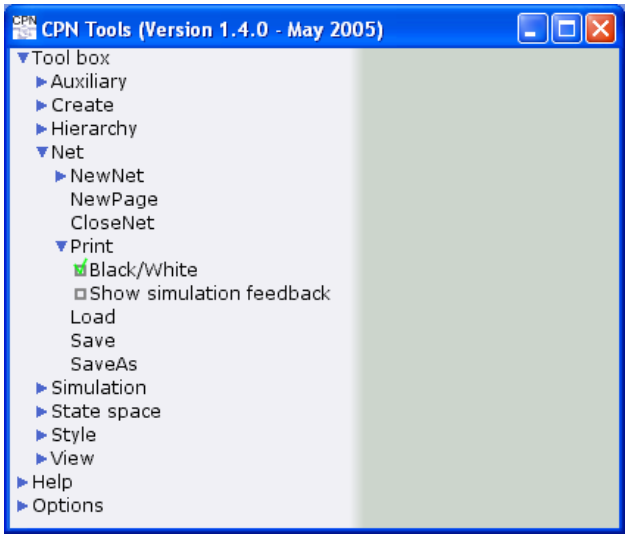
The tool's palette has appeared in the window.

To take an instrument from tool's palette we should click on it. Then cursor takes a shape of corresponding instrument. For instance, we choose `transition` tool from palette `Create`:



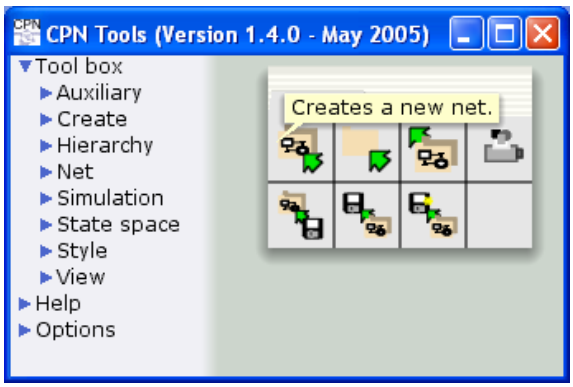
To abandon the tool we should drag it backwards on the palette and click on mouse or push `Esc` bottom on the keyboard.

Each tool has its special options that may be shown and changed by clicking on the corresponding rectangle in the index. For instance, in `Net` palette we may set printing option to print net in black and white:

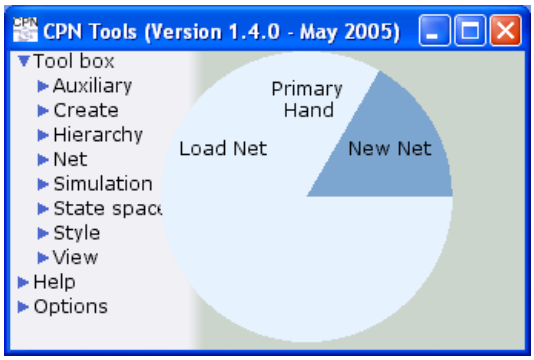


3.3. Context-sensitive Menus

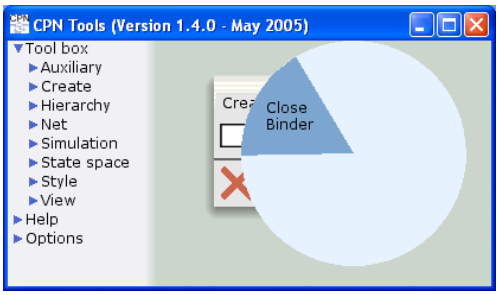
For convenient interaction CPN Tools provides a lot of context-sensitive menus appearing on the screen with pushing right button of mouse. Menus have the shape of a circle with named sectors. To keep menu on screen you should hold the button pushed moving mouse to choose required item. In the most cases items of context-sensitive menus duplicates tools in palettes. For instance we may create new net using instrument `new net` from `Net` palette:



This may be done also with context-sensitive menu:

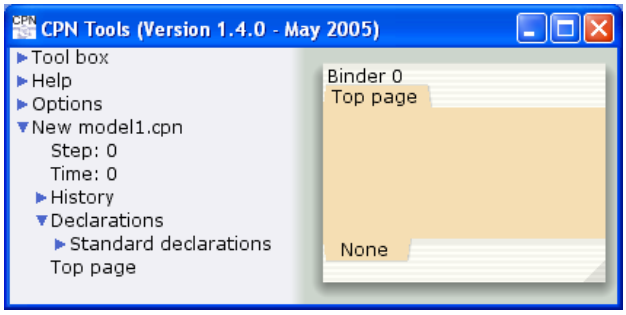


Context-sensitive menus make the interaction more natural and fast. You should only press right button of mouse on an object and choose required action. In this way you may close toolbox:



3.4. Model's Structure

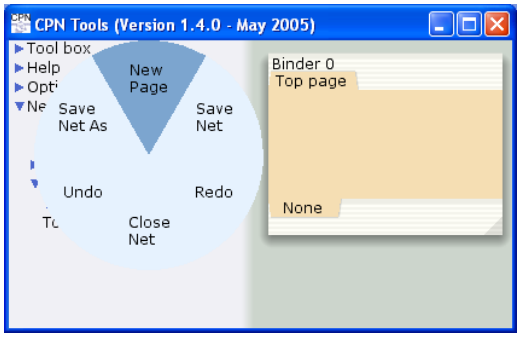
Models are named nets in CPN Tools. Their descriptions are situated in the index under standard items. Let us consider new net after its creation:



Each net in CPN Tools has:

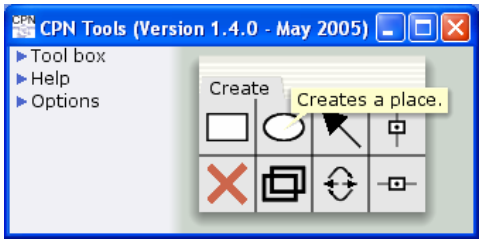
- Name – name of the correspondent file with type .cpn;
- Step - the number of steps that have executed in a simulation;
- Time - the current model time;
- History - the list of commands that have been performed on the net;
- Declarations - the declarations of color sets, functions, constant values;
- Pages – names of net's pages.

In the above window the name of net is model1.cpn, numbers of steps and time are equal to zero, net consists of the only page named "`Top page`". Let us notice that "`Top page`" has appeared in workspace; we can draw net inside it using tools. To open a page of a net we should drag it from index to workspace. For the creation of new declarations and new pages context-sensitive menus are used:

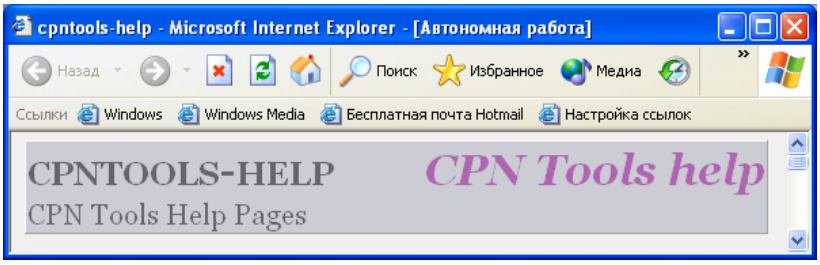


3.5. Organization of Help System

CPN Tools has three kinds of help: speech bubbles, offline help and online help. Speech bubbles appear on screen when you are keeping cursor on a corresponding item for a few seconds. It describes the pointed object:



Clicking on the help item in the index starts browser with hypertext help of CPN Tools:



It contains a lot of information on CPN Tools accomplished with nets' examples consideration. More peculiar details and up-to-date information is situated on CPN

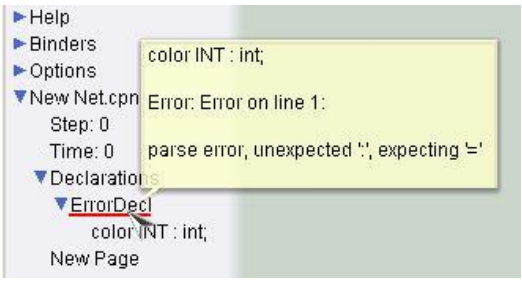
Tools home pages in Aarhus. In case of need, help system calls this information but it can be reached only in the case your computer is connected to Internet.

3.6. Reflecting Feedback of CPN Tools

CPN Tools provides graphical feedback that reflects current state of system. There are such kinds of graphical feedback as:

- Speech bubbles;
- Status bubbles;
- Auras;
- Changing cursor icon.

A **speech bubble** is a yellow rectangle that provides context-sensitive information. Some speech bubbles appear automatically, while others appear after a slight delay when the cursor is moved over an appropriate object. For example, moving the cursor over a declaration with a syntax error will cause a speech bubble containing an error message to appear.



Speech bubbles are used to show:

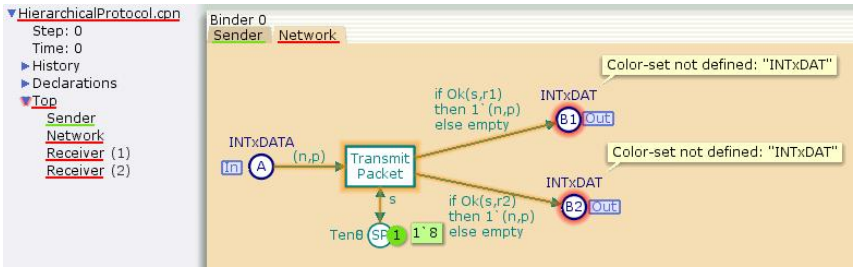
- Error messages during syntax checking.
- Error messages when simulating nets.
- Tool tips for tools in palettes and toolglasses.
- Detailed information for status bubbles.
- The result of applying the Evaluate ML tool.
- The full path to a saved net. To see the full path, move the cursor over the name of the net in the index.

Status bubbles are color-coded bubbles that occasionally appear at the bottom of the index. Move the cursor over a status bubble to see the corresponding speech bubble:

Status bubbles have one of the following colors:

- Green indicates that an operation was completed successfully.
- Red indicates that an error occurred when executing an operation.
- Light purple indicates that a time-consuming operation, such as a long simulation, is currently being executed.

Color-coded **auras** are used to highlight objects with particular characteristics or to indicate different kinds of relationships between objects. Auras are associated with places, transitions, arcs, inscriptions, declarations, page tabs, and index entries, such as page names and net names:



Auras have the following colors:

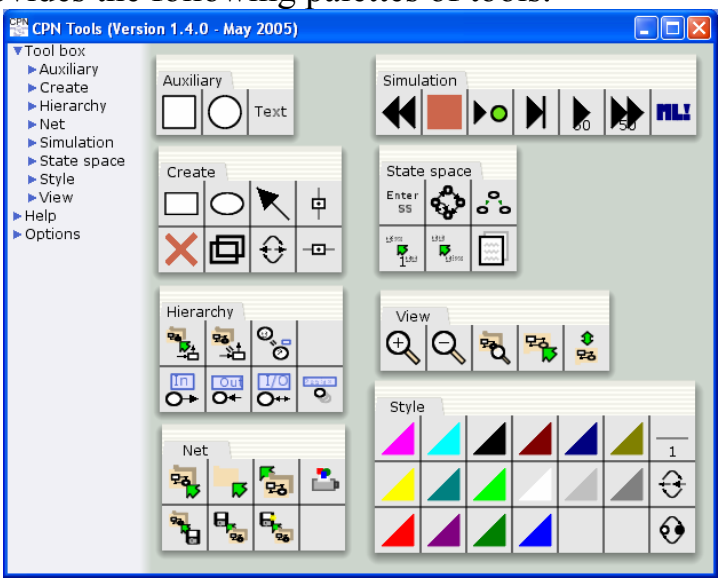
- (Bright) Red indicates objects with errors during syntax checking and when simulating nets.
- Dark redish auras indicate non-unique names of places and transitions when syntax checking.
- Green indicates enabled transitions when simulating nets.
- Dark blue indicates dependency between declarations and other elements, such as places, transitions, and pages.
- Aqua indicates which object an inscription belongs to.
- Orange indicates that syntax checking of an object has not yet begun.
- Yellow indicates that syntax checking of an object is currently being performed.
- Pink indicates which fusion places belong to a fusion set.
- Aqua indicates port/socket assignments and super-/subpage relationships when working with hierarchical nets.

The **cursor icon** changes to indicate which actions can be, or are being, performed. For example:

- The standard cursor is an arrow or just an arrowhead 
- The hand cursor  indicates that an item can be moved.
- The crossbar cursor indicates that it is possible to edit text.
- The double-headed arrow cursor indicates that an item can be resized. The directions of the arrow heads indicate which direction the item can be resized horizontally  , vertically, or both simultaneously.
- After picking up a tool from one of the palettes, the cursor will change to indicate which tool has been picked up.
- For multi-phase tools, i.e. tools that are applied by clicking on more than one object, the cursor will indicate which phase of the tool will be applied next. Examples of multi-phase tools are the assign port-socket tool and the set subpage tool.

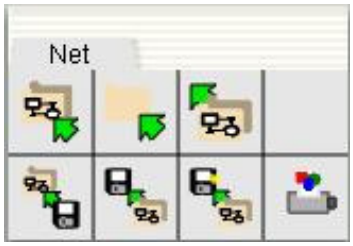
4. Tool Box of CPN Tools

Tool box provides the following palettes of tools:



- Net tools: for operations with whole nets
- Create tool: for drawing and editing Petri nets
- Simulate tools: for simulation of net's behavior
- State Space tools: for creation and analysis of state space
- Hierarchy tools: for creation of multilevel nets
- Style tools: for peculiarities of nets' appearance
- View tools: for scale choice and highlighting groups
- Auxiliary tools: for improvement of nets' readability

4.1. Net Tools

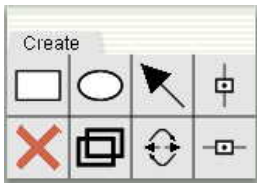


The items of the palette have the following meaning (from left to right and from up to down):

- creates a new net;
- creates a new page;
- closes a net;
- loads in a net;
- saves a net;
- saves a net with a new name;
- prints a net.

To create a new net you should start with “creates a new net” item and finish with “saves a net” item. To open an existed net you should start with “loads in a net”. Nets are printed to file in .eps (Extended Post Script) format and may be inserted, for instance, as pictures into MS Word documents. New pages are created mainly for hierarchical nets.

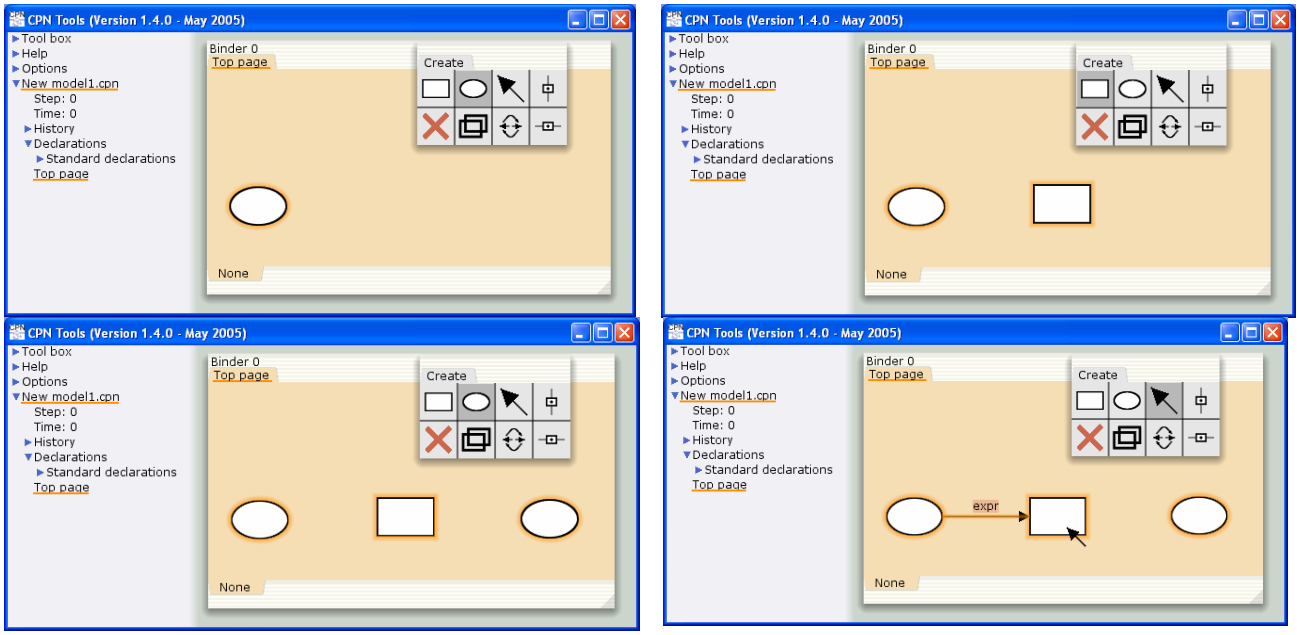
4.2. Create Tools

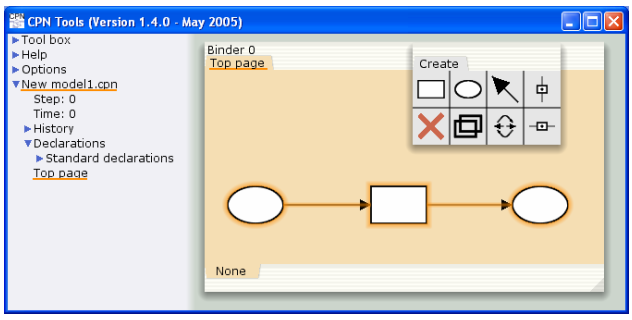


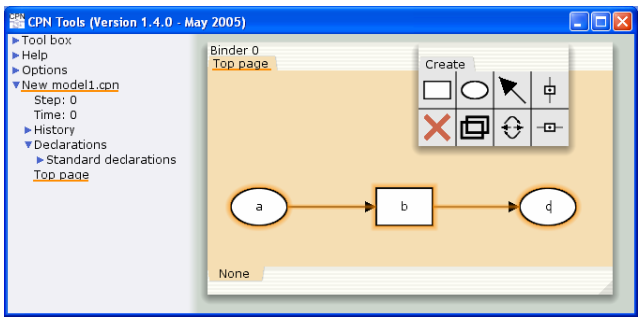
The items of the palette have the following meaning:

- creates a transition;
- creates a place;
- creates an arc;
- creates a vertical magnetical guideline;
- deletes an element;
- clones an element;
- cycles between the possible directions of arc;
- creates a vertical magnetical guideline.

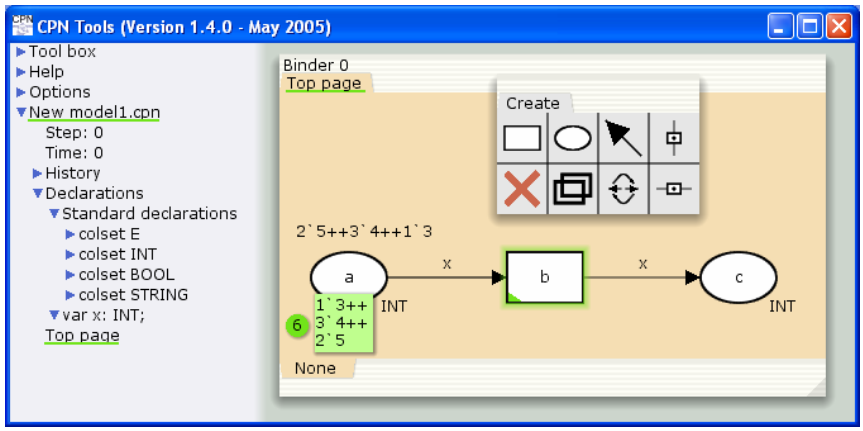
Let us start with this palette to draw our first net:







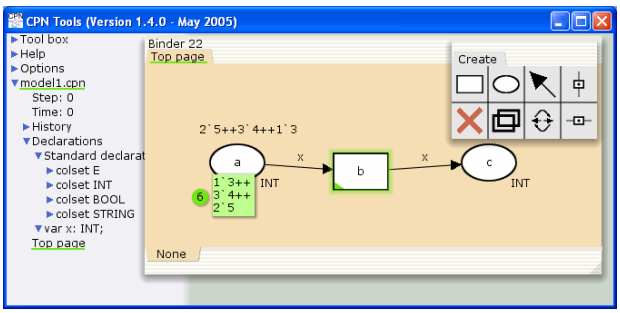
The simple Petri net was drawn but it is still not correct because its elements have no attributes. We shall use type `INT` from standard declaration to make this example work and add a variable `x` in declarations:

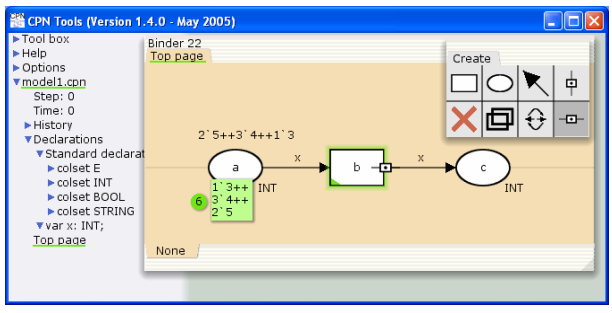


The net is correct now. Notice that place `a` contains 6 tokens: 1 of kind 3, 3 of kind 4 and 2 of kind 5. Let us show more meticulously the way to input attributes of nets' elements. Each node has its own set of attributes. After pointing an element you may switch among its attributes using `Tab` key of keyboard:

In the above example attributes of transition are not used. The current marking of the place is written in green color by CPN Tools automatically.

Magnetical guidelines are very useful to arrange elements of net in order. Elements are moved automatically to the nearest guideline. For instance:



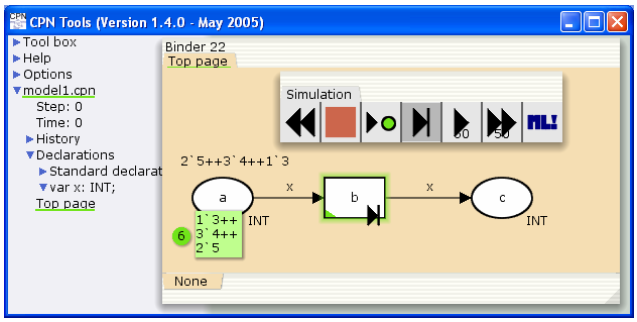


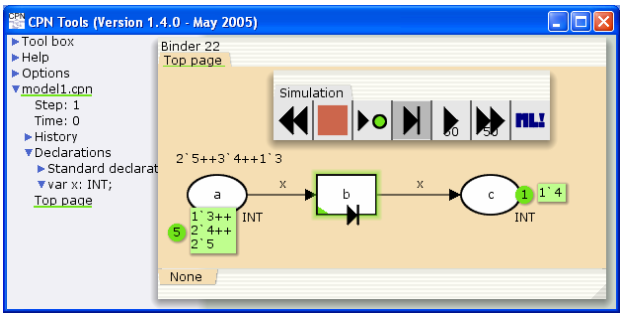
4.3. Simulation Tools

The items of the palette have the following meaning:

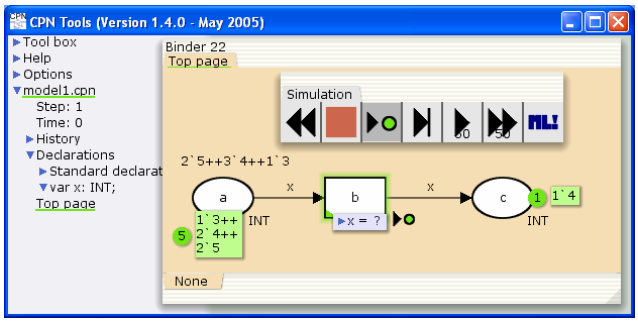
- goes to the initial state;
- stops ongoing simulation;
- executes a transition with a chosen binding;
- executes a transition;
- executes the specified number of transitions showing intermediate markings;
- executes the specified number of transitions without showing intermediate markings;
- evaluates a text as ML code.

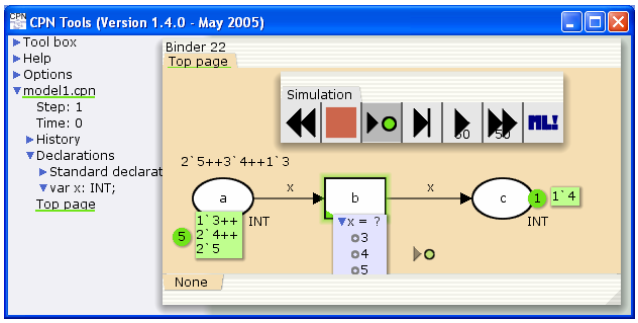
Items that "execute a transition" are aimed to the debugging of nets with step-by-step simulation. We may click on a firable transition to choose it or on empty spot of a binder to allow the choice of transition to CPN Tools. Let us consider the simulation process:

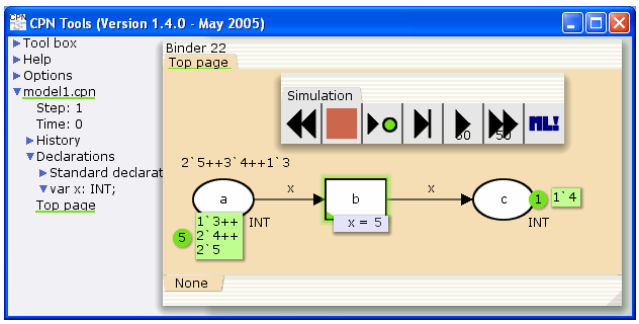


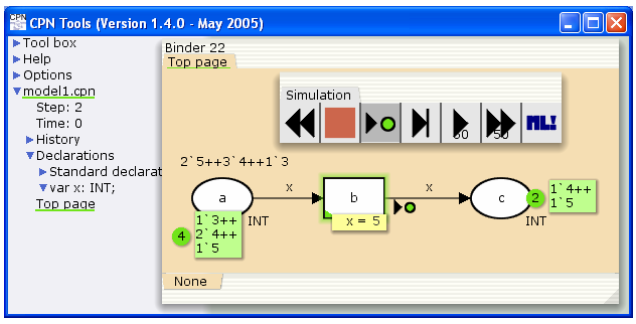


The token 4 has been chosen by variable `x` in a random way from place `a` and moved by transition `b` to place `c`. Executing transition "with a chosen binding" serves for minute debugging. In this case you can choose manually a token that satisfies to the inscription of transition's input arc:



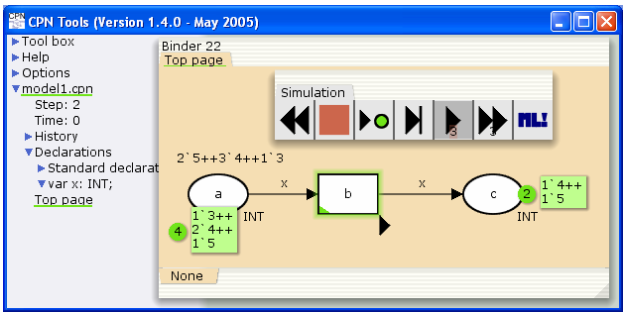


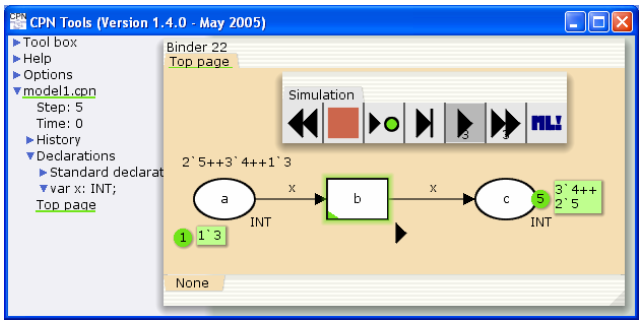




In this example the choice was proposed between tokens 3, 4, 5 and the token 5 has been chosen manually.

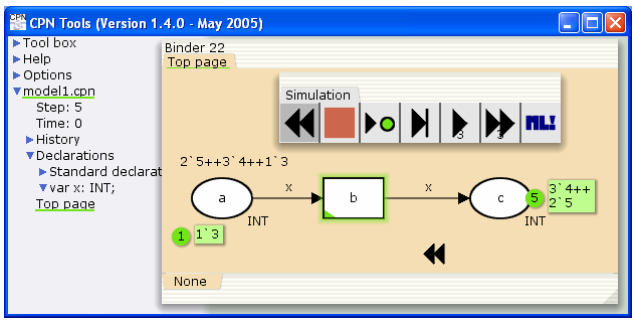
As for the "execution of specified number of transitions", they are chosen randomly by CPN Tools. You may enter the required number of transitions:

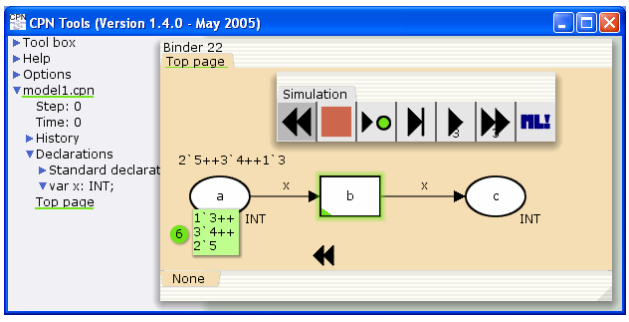




The number of three transitions was inputted in the above example. The only difference between two modes is that CPN Tools stops and shows intermediate marking in the first case and shows only the final marking in the second case. The result is the same but the mode without showing intermediate markings is much faster. It is used for simulation on large intervals of time for accumulation of statistical information.

Item "stops outgoing simulation" allows the interruption of simulation process when something is going wrong or it lasts too long time. Item "goes to the initial state" allows the return to the initial state:





Evaluation of a text as ML code is required to force syntax checking in language constructions.

4.4. Overview of Other Tools

Other tools are either accessory as Auxiliary, Style, View or more complicated as Hierarchy, State Space. Meticulous description of hierarchy and state space tools will be given in the following sections.

Auxiliary tools allow the creation of boxes, circles and text labels that do not have any semantic meaning, but can ease the readability of the net.

Style tools are used for underlining important net structures with colors, line thickness, size of arc's head (arrow), filling of elements to improve readability. None of these tools have any semantic effect on the net.

View tools are used to change the view of a page and its elements through grouping and zooming.

Hierarchy tools are used to edit the hierarchical structure of the net. The palette contains tools for both bottom-up and top-down structuring of the net.

State space tools are used to calculate state spaces of a net, to transfer states between the simulator and the state space tool, and to generate state space reports.

5. Basics of CPN ML

CPN Tools uses the CPN ML language for declarations and net inscriptions. CPN ML provides declarations of color sets (data types), variables, functions, values (constants). Each place of colored Petri net should have a definite color set as its attribute; it may contain tokens only of the specified color set. Variables and functions are used as the inscriptions of transitions and arcs.

Declarations are situated in the index as a part of net. There are standard predetermined declarations of such color sets as: E - elementary, INT - integer, BOOL - Boolean, STRING - string. User's declarations may be added after standard declaration using context-sensitive menu. Moreover, for complicated nets CPN Tools provides external declarations that may be loaded from a file.

CPN Tools automatically syntax checks your nets as you create them or when you load in a net. You can see by colors indications how far the check has gotten. The colors indications are shown in the index, underlining the name of the page where the color belongs. If the page is open in a binder, the color is also shown in the page tab at the top of the page, and on the CP-net element where the color belongs. The orange aura indicates that an element is not currently checked. When you load a net, the syntax check takes a couple of minutes to complete. During this phase, the elements will change aura from orange to yellow to no aura (or red, if there is an error). If the orange aura stays, it is probably because either there is something missing or there is an error on a related net element.

Declarations are checked starting from the top. If a declaration depends on a later declaration, it will get an error saying the other declaration is not defined. Declarations with errors are rechecked when a change is made in any declaration. If

there is an error in the declarations, the declaration with the error will be underlined with red. The net entry and all affected pages will also be underlined with red.

A red aura means the element has been checked but had an error. A speech bubble should appear with the exact error message. Elements connected to the element with the error are not checked until the error is fixed.

5.1. Simple Color Sets

CPN ML provides such simple color sets as: Unit, Boolean, Integer, String, Enumerated, Index.

The **unit** color set comprises a single element. The declaration has syntax:

```
colset name = unit [with new_unit];
```

Without option the name of token coincides with the name of color set. In the Cinderella example we used such units as:

```
colset p=unit with pumpkin;
colset c=unit with Cinderella;
colset m=unit with mouse;
colset f=unit with Fairy;
```

The **boolean** values are `true` and `false`. The declaration has syntax:

```
colset name = bool [with (new_false, new_true)];
```

The option allows the new names for `true` and `false`, for instance, `yes` and `no`:

```
colset Answer = bool with (no, yes);
```

Following operations may be applied to boolean variables:

not b	negation of the boolean value b
b1 **andalso** b2	boolean conjunction, and
b1 **orelse** b2	boolean disjunction, inclusive or

Integers are numerals without a decimal point. The declaration has syntax:

```
colset name = int [with int-exp1...int-exp2];
```

The option allows the restriction of the integer color set to an interval determined by the two expressions in `int-exp1` and `int-exp2`:

```
colset Dozen = int with 1..12;
```

Following operations may be applied to integer variables: `+, -, div, mod, abs, Int.min, Int.max`.

Strings are specified by sequences of printable ASCII characters surrounded with double quotes. The declaration has syntax:

```
colset name = string [with string-exp1..string-exp2
[and int-exp1..int-exp2]];
```

The option specifies the ranges of valid characters:

```
colset LowerString = with "a".."z";
```

The following operations may be applied to string variables: `^` - concatenation, `String.size, substring`.

Enumerated values are explicitly named as identifiers in the declaration. The declaration has syntax:

```
colset name = with id0 | id1 | ... | idn;
```

In the example with Cinderella the following enumerated color set was used:

```
colset g=with rice | wheat | oat;
```

Indexed values are sequences of values comprised of an identifier and an index-specifier. The declaration has syntax:

```
colset name = index id with int-exp1..int-exp2;
```

Indexed values have the form: `id i` or `id(i)` where `i` is an integer and `int-exp1 <= i <= int-exp2`. For example in the task about dining philosophers we may declare philosophers and forks as:

```
colset PH = index ph with 1..5;
colset FR = index fork with 1..5;
```

And philosopher ph(2) takes forks fork(1) and fork(2).

5.2. Compound Color Sets

Compound color sets constitute a combination of simple color sets. CPN ML provides such compound color sets as: `products, records, unions, lists, subsets and aliases`. As `lists` and `unions` are rarely used and more complicated they will be considered in the last section.

Products and records represent corteges of data formed by Cartesian products of components' color sets. The only difference between them consists in: components of product color set are unnamed while components of record color set have their names. There is close resemblance with record data type in Pascal programming language or structures in C language.

Declaration of **product** color set has syntax:

```
colset name = product name1 * name2 * ... * namen;
```

Values of this color set have form:

`(v1, v2, ..., vn)` where `vi` has type `namei` for `1<=i<=n`.

To extract `ith` element of a product the following operation is used:

```
#i name
```

Declaration of **record** color set has syntax:

```
colset name = record id1:name1 * id2:name2 * ... * idn:namen;
```

Values of this color set have form:

`{id1=v1, id2=v2, ..., idn=vn}` where `vi` are values of type `namei` for `1<=i<=n`.

To extract `ith` element of a product the following operation is used:

```
#idi name
```

Let us consider the above color sets on the example of Ethernet frame description. Ethernet frame consists of: source address, destination address and data. We represent MAC addresses with integer color set and frame's data with string color set.

```
colset MAC = int;
colset DATA = string;
colset frame = product MAC * MAC * DATA;
colset frame1 = record src : MAC * dst : MAC, d : DATA;
```

Ethernet frames may be represented either with `frame` or `frame1` color sets. For `frame` color set the value `x = (2, 4, "Hello")`, for instance, describes the frame sent by device 2 to device 4 containing staring `"Hello"`. The same value for `frame1` color set has the form `x1 = {src=2, dst=4, d="Hello"}`.

To extract destination address in `frame` color set we write:

```
#2 x
```

and in `frame1` color set:

```
#dst x1
```

An **alias** color set has exactly the same values and properties as a previously declared color set. It's introduced to use different name of color set. The declaration has syntax:

```
colset name = name0;
```

5.3. Declaration of Variables and Constants

A variable is an identifier whose value can be changed during the execution of the model. Variables are used in Petri net elements' inscriptions.

Declaration of **variable** has syntax:

```
var id1, id2, ..., idn : cs_name;
```

where `idi` is an identifier, `cs_name` is the name of a previously defined color set. For instance:

```
var f1, f2 : frame;
var f3, f4 : frame1;
```

A **value** declaration binds a value to an identifier (which then works as a constant). Declaration of value has syntax:

```
val id = exp;
```

where `id` is an identifier and `exp` is a CPN ML expression. The expression represents the value to be associated with the identifier. For instance:

```
val CheckFrame = (3, 5, "Ping" );
val ResponseFrame1 = {src=5, dst=3, d="OK"};
```

5.4. Functions

Functions of CPN ML implement standard control structures of a programming language such as "if" and "case" operators. But as ML constitutes in essence a language of functional programming the most power of it is revealed with recursive functions.

Declaration of **function** has syntax:

```
fun id pat1 = exp1
  | id pat2 = exp2
  | ...
  | id patn = expn;
```

where `pat1, pat2, …, patn` are patterns and `exp1, exp2, ..., expn` all have the same type. The declaration means that in the case actual arguments satisfy pattern `pati` then the value of the function is calculated as `expi`. For instance, the following function calculates factorial of an integer number using recursion:

```
fun fact ( 0 ) = 1
    | fact ( i ) = i * fact( i-1);
```

if-then-else and **case** control structures are available for functions' description:

```
if bool-exp then exp1 else exp2;
```

where `exp1` and `exp2` have the same type.

```
case exp of
     pat1 => exp1
   | pat2 => exp2
   | ...
   | patn => expn;
```

where `exp1, exp2, ..., expn` all have the same type. Their meaning is usual like in other programming languages. For instance, function that calculates the sign of a number may be written:

```
fun sign (x) = if x>0 then 1 else if x<0 then ~1 else 0;
```

The function that returns the name of a number may be written as:

```
fun nname (x) =
          case sign(x) of
             1 => "positive"
           | ~1 => "negative"
           |  _ => "zero";
```

The underscore _ in the last line indicates that the string "`zero`" will be chosen for all the other values of checked expression (`sign(x)`).

The **let** construction permits the declaration of locally-scoped variables within a function definition:

```
let
   val pat1 = exp1
   val pat2 = exp2
   ……
   val patn = expn
in
   exp
end;
```

For instance, in calculation of the size in meters on millimeters:

```
fun metr (x) =
  let
     val mminm=1000;
  in
     x div mminm
  end;
```

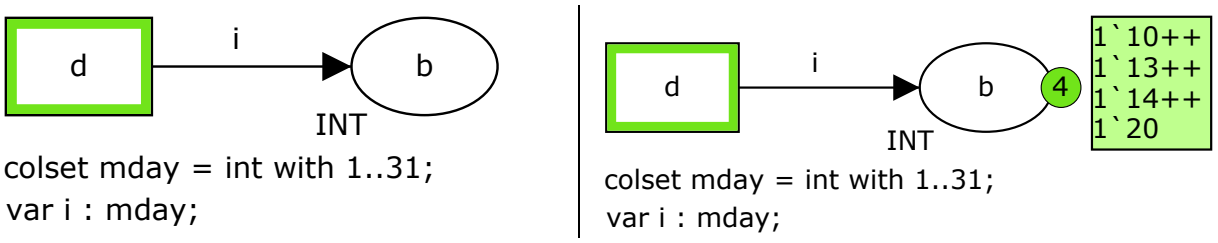
5.5. Random Functions

Random functions provide facilities for modeling of statistical characteristics. For instance, they allow the description of traffic's intensity or faults' frequency in telecommunication systems. There are a few ways for random choice description in CPN Tools:

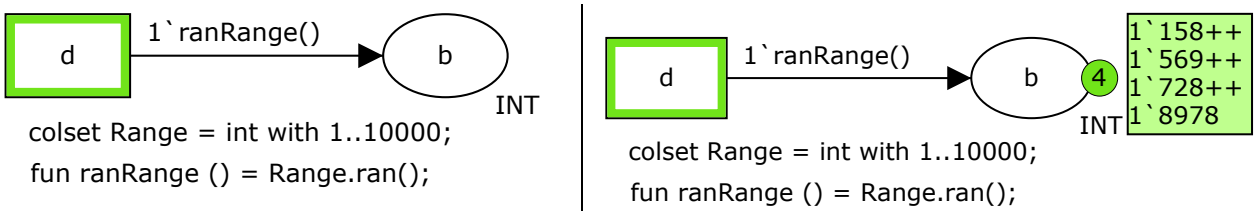
- free variables;
- function `ran`;
- special random distribution function;

Free variables are output arc variables that have not been bound on an input arc or in the guard. They are assigned random values when executing a CP-net. The type of free variables must be small color sets. Color sets can be classified as large or small. This distinction determines which predefined functions are meaningful for a particular color set. A color set is large if it contains too many (default 100) elements to enumerate, otherwise it is small. Unit color sets, Boolean color sets, index color sets, and enumerated color sets are small.

In the following example the variable `i` is a free variable with the range 1..31. In the second picture the state after four steps is shown; four values of `i` were taken by CPN Tools in a random way:



Function **`ran`** generates a random value for large color sets. In the following example function `ranRange()` generates a random value in the range `1..10000`:



CPN Tools provides also a series of special **random distribution function** for such well-known distributions as Bernoulli, binomial, Erlang, exponential, normal, Poisson, student, uniform (discrete and continuous). For instance, function

```
erlang(n:int, r:real) : real
```

where `n>=1` and `r>0.0`, returns a drawing from an n-Erlang distribution with intensity `r`.

5.6. Multi-sets

Multi-sets are widely used in CPN Tools for place's marking representation and other purposes. Let us remind the concept of multi-set. In spite of usual set it contains each element with a definite multiplicity in other words in definite number of copies. Multi-sets are also named bags.

The back-quote (`) operator is the multi-set constructor. For example, `3`5` is the multi-set with three appearances of the color 5. The description of a multi-set has the syntax:

```
i`c
```

The integer `i` must be non-negative. If this is not the case then the empty multi-set will be returned. The multi-set operator combined with multi-set addition (++) and subtraction (--) provide a succinct method for specifying multi-sets. For example, in the described in Section 1 Cinderella model the place "`sack of mixture`" has initial marking:

```
1000`rice ++ 2000`wheat ++ 3000`oat
```

It means that the sack contains 1000 grains of rice, 2000 grains of wheat and 3000 grains of oat. Please pay your attention to the sign of multi-set constructor: it is backquote (`) but not apostrophe (‘). For the example of Ethernet frames with color set `frame1` the content of a buffer may be presented as:

```
1`{src=2, dst=5, d="request"} ++ 1`{src=5, dst=2, d="answer"}
```

It means two frames: the first with data "`request`" directed to device 5 from device 2 and the second with data "`answer`" directed to device 2 from device 5.

The following Constants, Operations, and Functions are available for multi-sets:

`empty`	the `empty` constant constructs an empty multi-set that is identical for all kinds of multi-sets
`ms1` **`==`** `ms2`	multi-set equality
`ms1` **`<><>`** `ms2`	multi-set inequality
`ms1` **`>>`** `ms2`	multi-set greater than
`ms1` **`>>==`** `ms2`	multi-set greater than or equal to
`ms1` **`<<`** `ms2`	multi-set less than
`ms1` **`<<==`** `ms2`	multi-set less than or equal to
`ms1` **`++`** `ms2`	multi-set addition
`ms1` **`--`** `ms2`	multi-set subtraction (`ms2` must be less than or equal to `ms1`), raises `Subtract` exception if `ms2` is not less than or equal to `ms1`.
`i` **`**`** `ms`	scalar multiplication
`size` `ms`	size of multi-set ms
`random` `ms`	returns a pseudo-random colour from ms
`cf``(c,ms)`	returns the number of appearances of colour c in ms
`filter` `p ms`	takes a predicate `p` and a multi-set `ms` and produces the multi-set of all the appearances in ms satisfying the predicate

For instance, let

```
m1 = 2`5  ++ 3`4 ++ 4`5;
m2 = 1`5 ++ 2`4 ++ 3`5;
```

then

```
m1 ++ m2 = 3`5 ++ 5`4 ++ 7`5
m1 – m2 = 1`5 ++ 1`4 ++ 1`3
m1 >> m2       is true
size m1 = 9
cf(4,m1) = 3
```

In CPN tools initial and current marking of a place is represented with multi-set of color set of the place. And at the choice of a token by a variable in the inscription of place's output arc CPN Tools provides a random choice like with function `random`.

5.7. Timed Multi-sets

Timed multi-sets are used in CPN Tools to represent timed delays in the model. The declaration of corresponding color set should be accomplished with modifier `timed`. The @, @+, and @@+ operators are used to add time stamps to colors. Adding a *time delay* of `x` to a color `c` will attach a time stamp with a value that is equal to the *current model time* + `x` to the color `c`. The following operations are valid for timed multi-sets:

`c @ t`	attach the time stamp `t` (with type `Time.time`) to the colour `c`
`ms @+ i`	add the integer time delay `i` to each of the colours in multi-set `ms`, returns a timed multi-set
`tms1 +++ tms2`	timed multi-set addition

For instance the declaration

```
colset tint = int timed;
var t : tint;
t = 1`2@100 ++ 1`3@200 ++ 1`4@300;
```

means that token `2@100` might be taken by CPN Tools only after model time instance equaling to 100, the token `3@200` – only after model time instance equaling to 200 etc. Before the time of activation a token can not be taken by any transition of a model.

6. The Language of Models' Description

In CPN Tools each element of Petri net has its attributes described in CPN ML language. Using `Create` palette we put an element onto a page of the model. Then

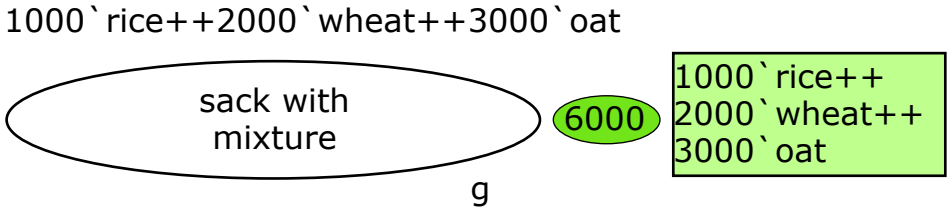
element's attributes should be added. For this purpose you should click by mouse on the corresponding element and use `Tab` key for switching among attributes. Pressing `Esc` key lets you leave the chosen element; the same result may be obtained by clicking mouse at the other spot of the model. Let us consider attributes of each element separately.

6.1. Place Inscriptions

There are three inscriptions that may be associated with a place. Two are optional and one is required:

- Color set inscription - required
- Initial marking inscription - optional
- Place name inscription- optional

On the initial marking CPN Tools automatically creates current marking which is written in green color and shows the total number of tokens and marking's details. For instance, in Cinderella example:



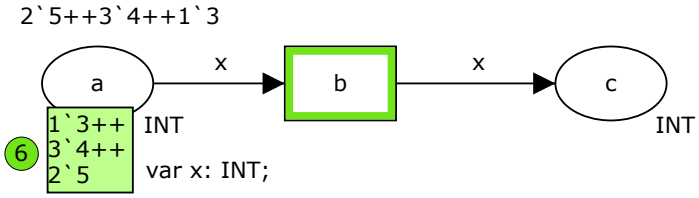
6.2. Arc Inscriptions

The inscription has the form:

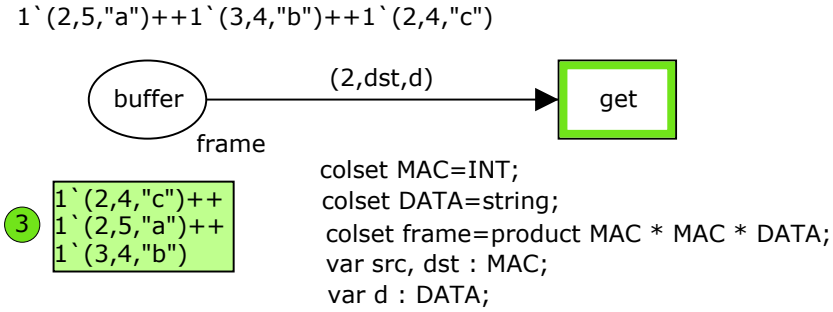
The color set of the arc expression `expr` must match the color set of the place attached to the arc. If the color set of an arc expression does not match the color set of the place attached to the arc, an error message will appear near the arc during syntax checking. There is essential difference between inscriptions of input and output arcs of a transition.

The transition's input arc expression constitutes a pattern for choice of token. This pattern is described by a predicate which may be applied in function `filter`

for corresponding color set. In the simplest case this predicate consists of a single variable of the corresponding color set as in the above examples. In the net

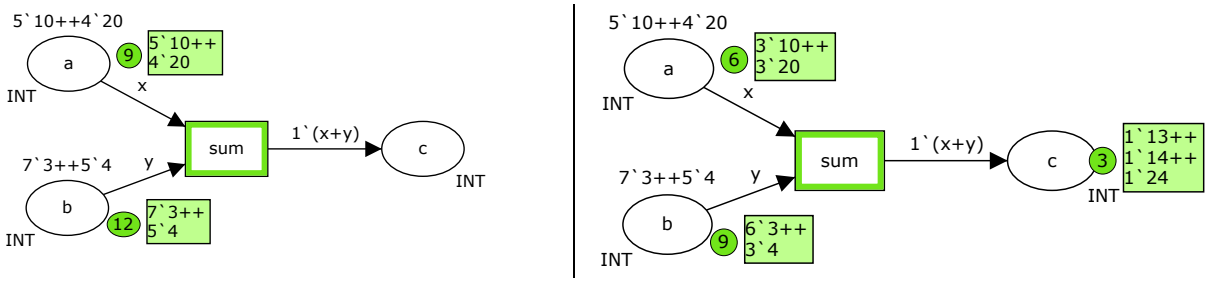


any of 6 tokens may be chosen by variable `x`. More complicated case constitutes the choice of frames with source address equaling to 2 for Ethernet frames:

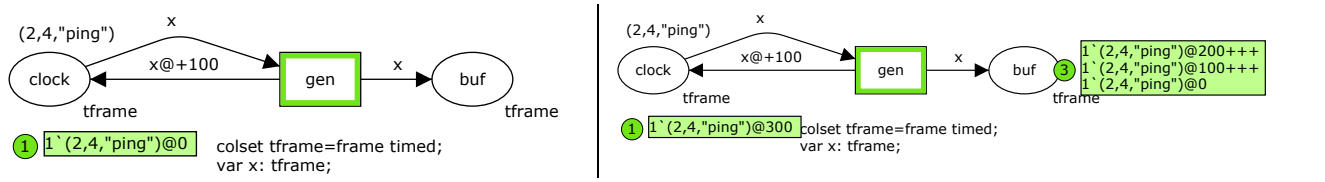


Any of frames `(2,5,"a")` and `(2,4,"c")` may be taken by inscription `(2,dst,d)`.

The transition's output arc expression constitutes a constructor for new tokens creation. This constructor often uses variables of input arcs inscriptions and in the simple cases may coincide with one of them. In the following example transition sum calculates the sum of input tokens:



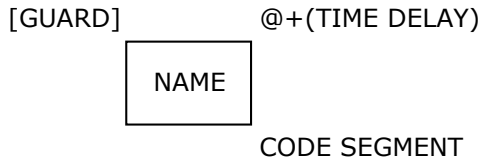
Moreover, time delays may be applied to output tokens. In the following example we provide generation of a frame every 100 units of model time:



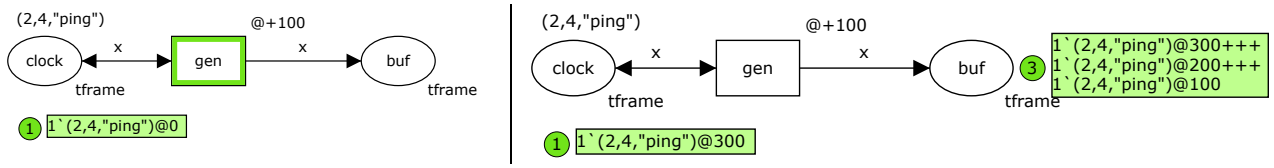
6.3. Transition Inscriptions

There are four inscriptions that may be associated with a transition. All are optional:

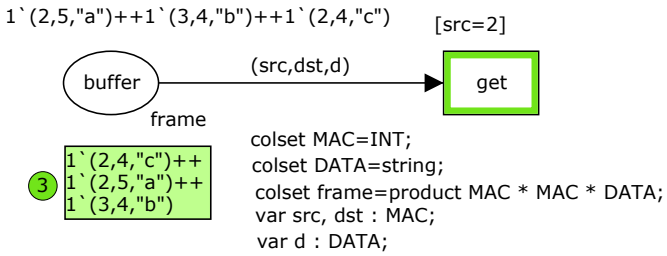
- Transition name inscription
- Guard inscription
- Time inscription
- Code segment inscription



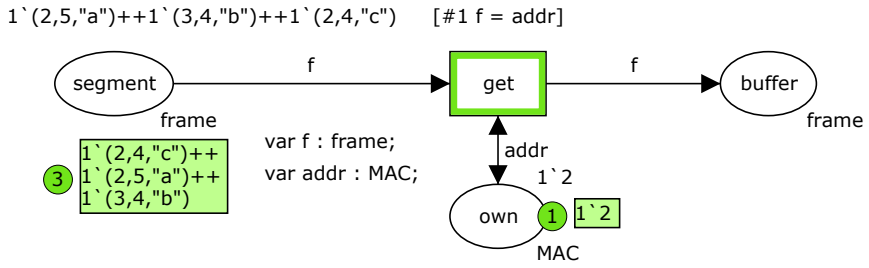
A transition **delay** must be a positive integer expression. The expression is preceded by @+, and this means that the time inscription has the form @+ `delay-expr`. Before a time inscription has been added, the default text for the inscription is @+. Time delay is always added relative to the current time. For example, if current time is 10 and the time delay is `@+2`, then the time stamp of tokens sent to the output places will be 12. A missing time inscription is equivalent to a zero delay. In spite of arcs' delay transition's delay is applied to all the output tokens of transition. In the case of one output arc it's the same. For instance, compare the following net with the example in section 6.2:



A **guard** is a CPN ML Boolean expression that evaluates to true or false. Before a guard has been added, the default text for the inscription is `[]`. Guard may be a single Boolean expression or a list of Boolean expressions `[b-exp1, b-exp2, ..., b-expn]`. The transition fires only in the case its guard is true and guard restricts the choice of input tokens. For instance, the checking of input frames presented in section 6.2 may be written as:



Moreover, guard allows the comparison of parameters of tokens from different places using their combination in expressions. The following fragment models the process of frames' extraction from Ethernet segment:



The own address of workstation is stored in the place `own`. It's checked only so bidirection arc is used.

Each transition may have an attached **code segment** which contains ML code. Code segments are executed when their parent transition occurs. Code segments may use CPN variables and may bind CPN variables located on output arcs that are not bound elsewhere. Each code segment may contain:

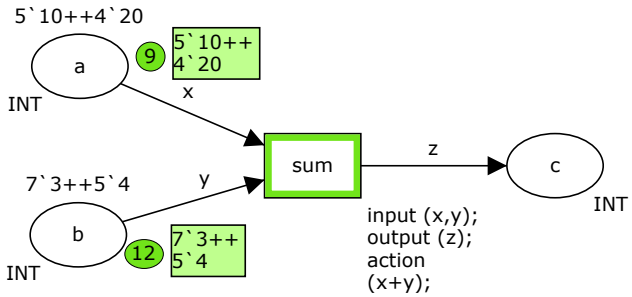
- Input pattern (optional)
- Output pattern (optional)
- Code action (mandatory)

An input pattern is a tuple of CPN variables, preceded by the keyword `input`. The input pattern lists the CPN variables that can be used in the code action. The code action can use the values of these CPN variables but it cannot change them. The CPN variables listed in the input pattern can be used in the code action even if you have declared an ML identifier with the same name in the declaration node. If the input clause is omitted, it implies that no CPN variables can be used in the code action.

An output pattern is a tuple of CPN variables, preceded by the keyword `output`. The output pattern lists the CPN variables to be changed as a result of the execution of the code action. An output pattern must be a CPN variable or a tuple of CPN variables without repetitions. If the output clause is omitted, it implies that no CPN variables are calculated.

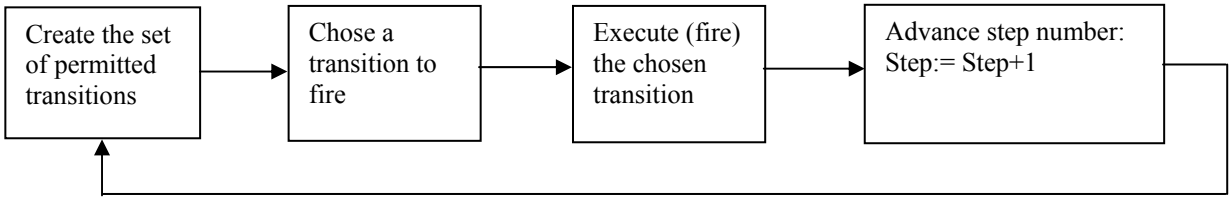
A code action is an ML expression, preceded by the keyword `action`. The code action cannot contain any declaration of color sets, CPN variables, or reference variables. It can, however, apply user-declared and predeclared constants, operations, and functions. In addition, new functions and constants can be defined for local use by means of `let-in-end`. The code action is executed as a local declaration in an environment containing the CPN variables specified in the input pattern. This guarantees that the code action cannot directly change any CPN variables but only local copies of them. When the code action has been executed, its result is applied to bind the CPN variables in the output pattern. The code action, when evaluated in an environment containing the input pattern variables must yield a result of the same type as the output pattern. If no output pattern is given, its type is assumed to be unit.

Code segments are used for more complicated processing of input tokens. The example for sum of tokens described in section 6.2 may be represented using code segment as:



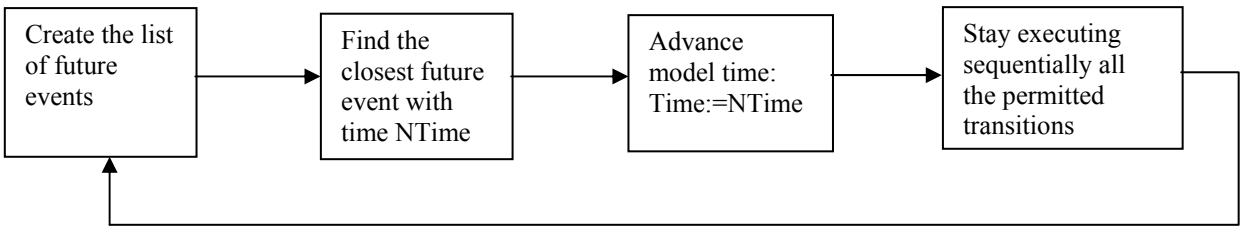
7. Peculiarities of Timed Nets in CPN Tools

CPN Tools provides both: Petri net without times and timed Petri nets. If none of color sets of a model has modifier `timed` then net is considered as untimed. In this case CPN Tools uses only internal Step variable that means the number of executed transitions. The algorithm of CPN Tools simulation may be represented in the following way:

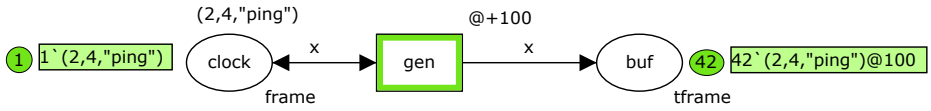


Notice that, the transition's choice may be implemented either manually at step-by-step simulation or automatically in a random way by CPN Tools at execution of specified number of steps.

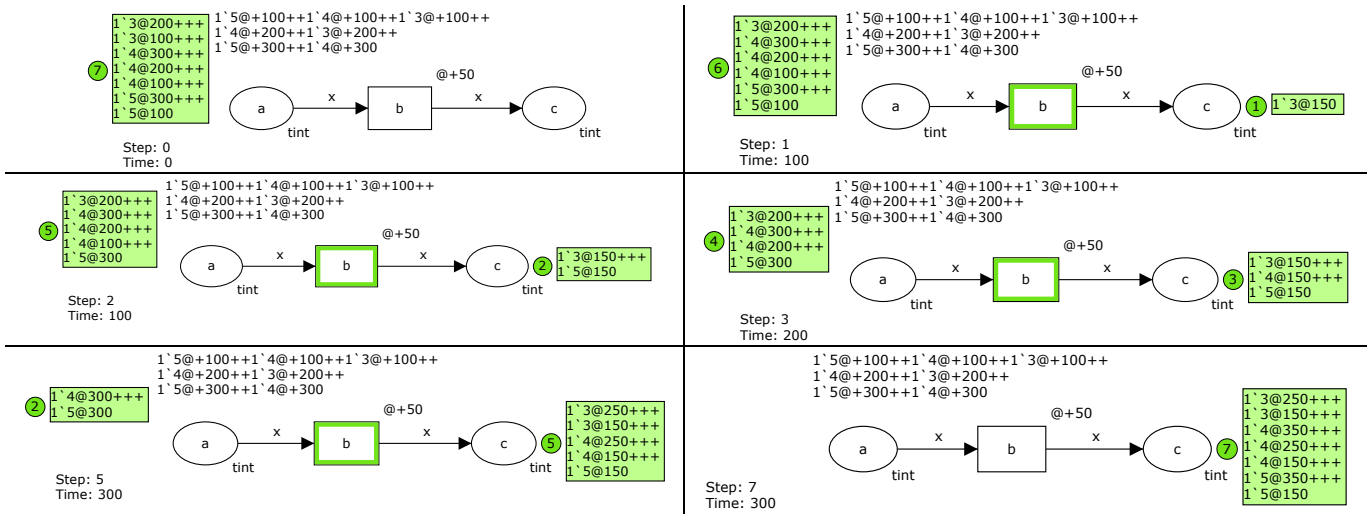
For timed Petri nets the algorithm is more complicated because the way of time advancing:



The next instant of the model time will not be the instant Time+1. The system advances time to the nearest event in the future Time:=time-of-nearest-future-event and then executes all the events that may occur in this instant of time advancing the variable Step in the described above way. So we should be careful combining timed and untimed nets because permitted transitions fires till the set of permitted transitions becomes empty. It may cause the so named infinite looping in models' behavior. For instance, in the following net time does not move and the number of generated frames is constantly increasing:



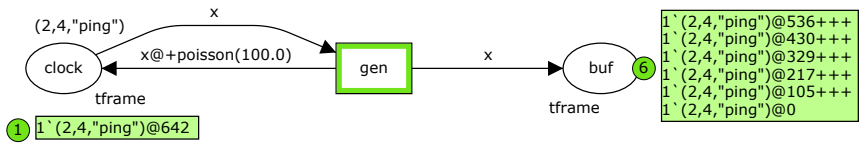
It should be noticed that CPN Tools implements very simple and powerful class of Timed Petri nets due to time stamps usage. Each token is supplied with its time stamp t (k@t). For moments of model Time Time<t the token is not processed by simulation system; it is being in so named void state and does not take part in firing any transitions. After moment of time t token k wakes up and participates in firing of transition. This way of times representation allows the instantaneous firing of transitions. Output tokens supplied with time delays (k@t+d) wait their time of activation inside corresponding output places. The following example of tracing shows the way of time processing in CPN Tools:



In the instance of time 0 no transition is permitted so no token has time stamp 0. The next instance of time equals to 100. CPN Tools executes 3 steps at this moment of time moving 3 tokens to place c; all these tokens have initial time stamps equaling to 100. In the instance of time 200 it moves 2 tokens with time stamps equaling to 200 and finally in the instance of time 300 it moves 2 remained tokens.

For description time delays either delays of transition or delays of arcs may be used. Delays of transitions in most cases are more comprehensible because transitions model actions of an object. But the usage of arcs' delays gives us more flexibility at the models construction.

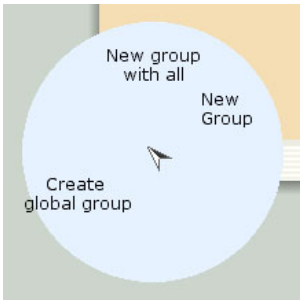
For modeling traffic it is convenient the usage of random functions as time delays. The wide choice of random distribution functions described in section 5.6 allows the description of traffic peculiarities. In the following example the Poisson flow of frames is generated:



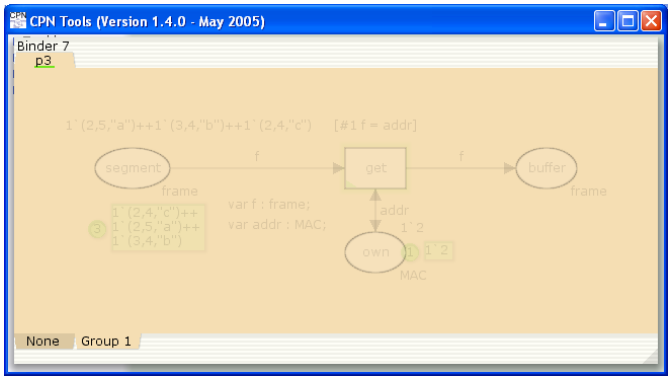
8. Working with Nets' Fragments

Working with fragments is considered as a standard opportunity of a graphical editor. Most graphical editors (for instance, Corel Draw) provide operations with highlighted rectangular fragments of a picture. The concept of nets' fragments is quite generalized in CPN Tools and formulated in the terms of group of elements. A group may have any shape and it is given by switching (toggling) elements belonging to it. Then the fragment may be duplicated and moved to any location using `Clone tool` of the palette `Create`.

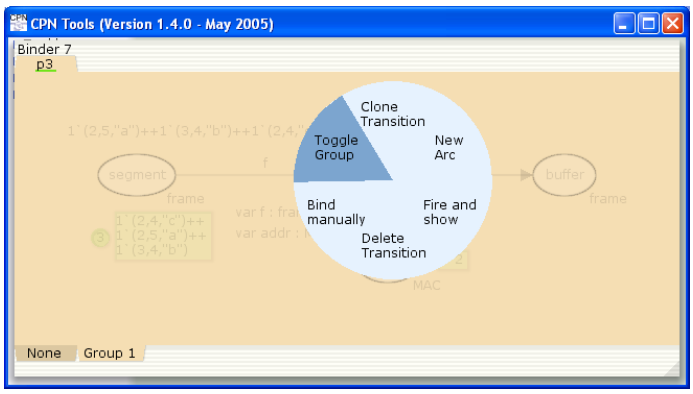
To create a group, use the `New group` tool. The corresponding menu appears at clicking on group tab at the left-bottom of the binder:



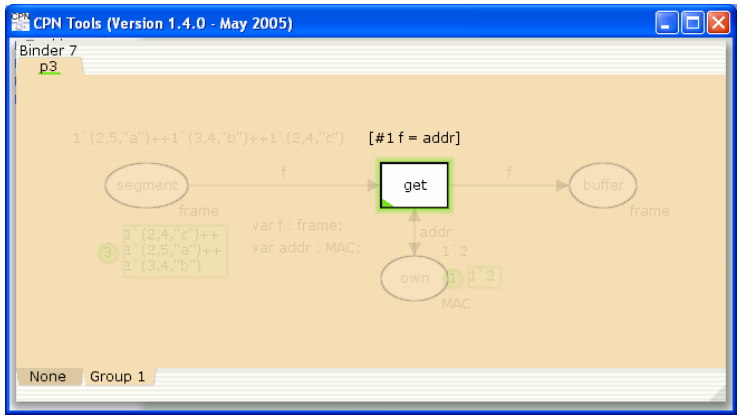
Select the "New group" entry. Now a new group tab appears, and the new group is set as the active group. All objects on the page are dimmed to show that there are no members in the new group:



To change the active group, just click on the group tab of the group you wish to activate. The group tab to the far left (the first group tab, called NONE) works differently - clicking this tab means deactivating all groups. When this tab is clicked, all objects on the page appear normal and work normally. To add an object to the active group, use the `Toggle group` tool. For example, bring up the marking menu on the object and select the "Toggle group" entry:



If the object is not in the active group, it is added to the group (and highlighted). Inscriptions, etc. around an object are automatically added to the group:



To remove an object from the active group, use the `Toggle group` tool. For example, bring up the marking menu on the object, and select the "Toggle group" entry. If the object is a member of the active group, it is removed from the group (and dimmed).

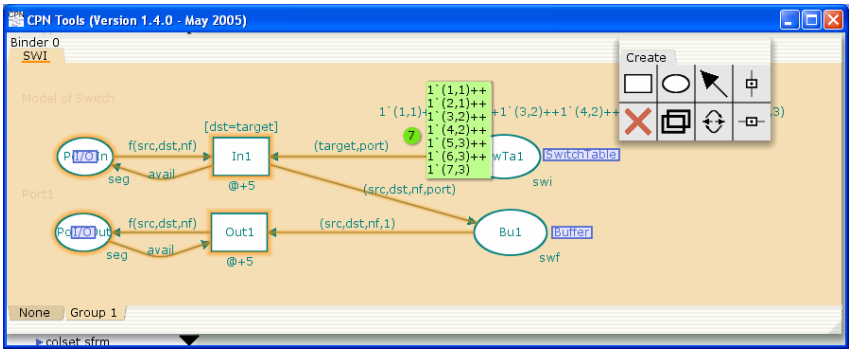
Groups can be used for several different purposes:

- Changing attributes
- Moving groups of elements
- Cloning groups of elements
- Deleting groups of elements

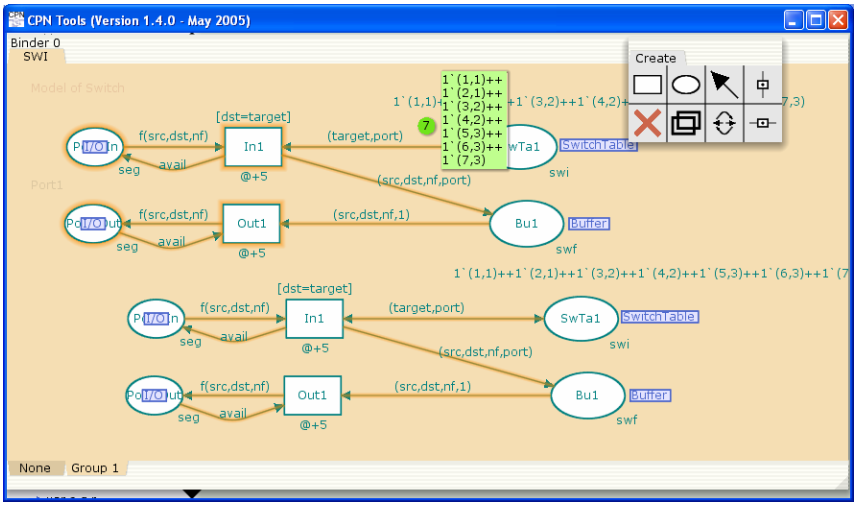
Groups can be used to change attributes on several objects at the same time. If, for instance, a number of places, transitions and arcs are placed in a group, you can change the color of all of them at the same time. Groups can also be used to move several different objects at one time. To move a group of objects, simply drag one of the objects in the group, and all other objects in the group will be moved at the same time. Deleting the member elements of a group works similarly to the other operations. Normal groups only span a single page, but with global groups is it possible to manipulate elements across all pages in the net.

Cloning of groups of elements is an extension of the basic cloning, where more than one element is cloned at the same time. If a group is selected, and the target of the clone tool is a member of the selected group, then the *elements in the group* will be cloned. As with cloning of individual elements, the result is a cursor with the cloned elements attached that may be used to insert them one or more times. Cloning

of groups is a very useful operation for creation of nets with a regular structure or reusing of early created nets. In the example of Ethernet model in Appendices it is very convenient to assemble the model of Switch by cloning submodels of a single port. We toggled the grope with the elements of the first port:



And simply clone it to create models of other ports:



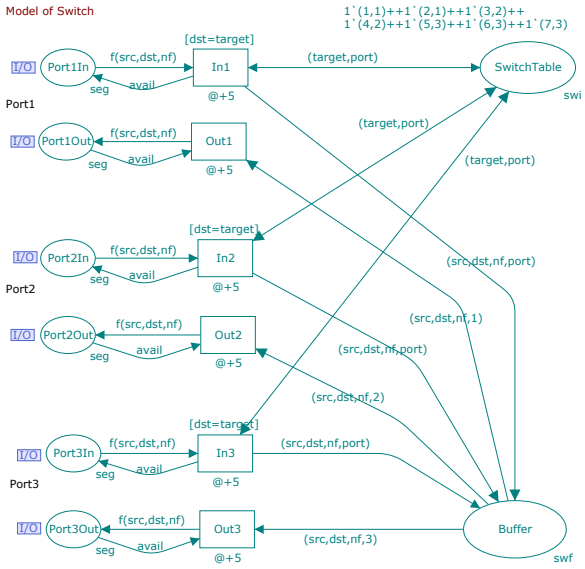
The only thing we need is to move group in the appropriate location and to correct the names of elements.

Notice that, every cloned element can be placed in an arbitrary open net. In this way cloned elements can be transported across net. The same goes for groups of elements, meaning you can clone entire groups of elements to other nets.

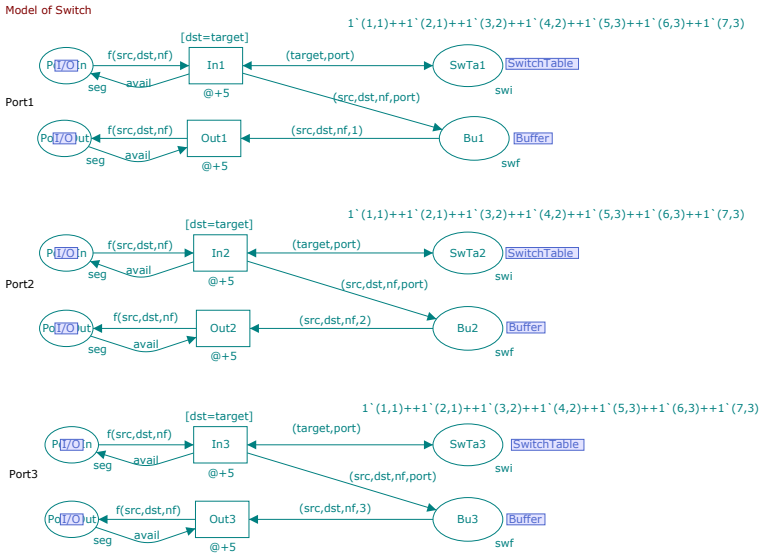
9. Fusion Places

Fusion places provide both opportunities: to make your model more vivid and to settle connections between pages of net. Fusion places may be considered as a first step to hierarchical Petri nets so item “Fusion” is situated in `Hierarchy` palette.

Each place of a fusion set is supplied with the fusion tag of the same name; all the places in a fusion set are considered by CPN Tools as the same place. Visually the changing of marking for one of the places causes in the changing of other fusion places marking. Fusion places should be of the same color set. Let us consider the example of Ethernet switch model:



Frames are extracted from the input channel of a source port `Port*In`, put to `Buffer` and then directed to output channel of destination port `Port*Out`. To find the number of destination port switching table `SwitchTable` is used. The model contains a lot of crossed lines and at the increase of ports' number becomes unreadable. The same model is constructed in Appendix A2 using two fusion sets: `SwitchTable` and `Buffer`.



The model can be easily expanded for an arbitrary number of ports using cloning of port's group.

Fusion places are created using the `Assign fusion set` tool from the `Hierarchy` tools. After applying the tool, a fusion tag is added to the place. The tag is set in a default position with a default name for the fusion set. The tag can be repositioned, and the name of the fusion set can be changed by editing the text in the fusion tag. The members of a fusion set can be located by placing the cursor over a fusion tag. An aqua aura indicates which place the fusion tag belongs to. Pink auras and highlights indicate other places in the same fusion set and the pages which contain these places. Fusion sets may contain places from several different pages.

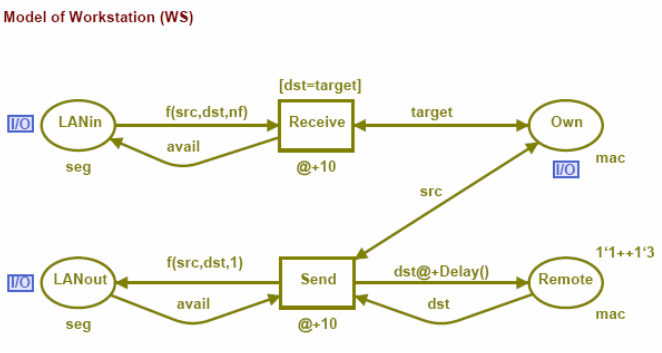
10. Hierarchical Models' Construction

The usage of hierarchy is very common for engineering. A telecommunication device consists of blocks, blocks consist of boards, boards consist of chips, capacities, resistances etc. In programming a program is assembled of modules (procedures, functions). Hierarchical model means a nested construction: net inside net. Generally any element of Petri net may be substituted by nested net but CPN Tools uses substitution of transitions only.

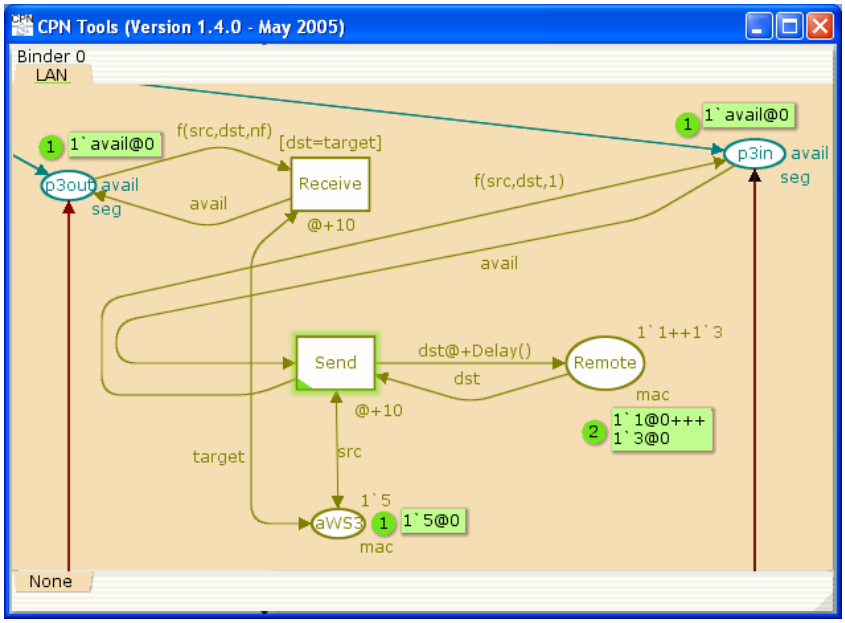
10.1. Basics of Transition Substitution

At the substitution of transition we have a pair of nets at least. A transition of upper-level net is substituted by lower-level net. Let us consider the model of LAN described in Appendices. At the top page `LAN` transition `WS3` is substituted by net `WS`:

Fragment of LAN Model



The substitution is pointed by tag `WS` drawn about transition `WS3`. Logically, the behavior of a whole net is the same as subnet `WS` was put inside net `LAN`:



Places of lower-level net which are used in connection with upper-level net are named contact places or *ports* and marked with special tag (I/O). Corresponding places of upper-level net are named *sockets*. In the above example we have ports LANin, LANout, Own in page WS and sockets p3out, p3in, aWS3 in page LAN.

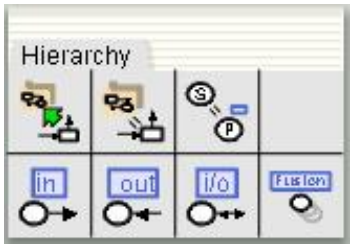
To prepare the substitution of transition we should:

- create both upper-level and lower-level nets situated in separate pages of model (LAN, WS);
- point out ports of lower-level net (LANin, LANout, Own);
- provide the number of sockets equaling to number of ports (3).

To make the substitution of transition we should:

- assign net to transition (WS3->WS);
- assign ports to sockets (LANin->p3out, LANout->p3in, Own->aWS3).

Ports of lower-level net are pointed out using Hierarchy palette:



Three types of tags are available: In, Out, I/O. Ports of type In are used when the corresponding place has only input arcs in upper-level net and only output arcs in lower-level net. Ports of type Out are used when the corresponding place has only output arcs in upper-level net and only input arcs in lower-level net. They may be considered as input/output variables in programming language routine. When there are no restrictions on arcs, port of type I/O is used. Such a place may have incidental arcs of both directions in upper-level net as well as in lower-level net.

To assign page to transition the second tool of Hierarchy palette is used. You should click with it on transition (WS3) and then on subpage (WS). The correspondent tag of subpage will be attached to transition.

The assignment of ports to sockets is more complicated because the operation should be executed for each port of subpage. The third tool of Hierarchy palette "assign a port to a socket" is used. You should have both pages on the screen. To make the assignment of a port you should click on it and then click on the corresponding socket.

Notice that the usage of hierarchical nets makes the development of models considerably easy as the usage of modules in programming languages. At first it allows the hiding of details and the management of models' complexity. At second, it provides reusage of submodels. For instance, in the example of LAN model there are 5 workstations and 2 servers but we have single subpages for workstation (WS) and server (S).

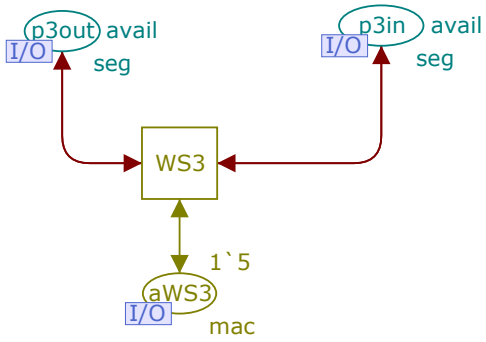
10.2. Bottom-up Development

When creating a hierarchical net "bottom-up", you start by creating separate pages. In contrast to top-down development, this approach involves creating the most detailed parts of the net first. Later, existing pages are set as subpages for substitution transitions as it was described in the previous subsection. It looks like usage of libraries in programming languages.

For instance, in LAN example you should create pages `WS, S, MWS, SWI` then create top page `LAN` and then assign substitution of transitions (`WS1-WS4->WS, S1-S4->S, WS5->MWS`) and corresponding port/socket mapping.

10.3. Top-down Development

When creating a hierarchical net "top-down", you start by creating the top-level page that shows the overview of the subpages and how they are connected. For instance, in LAN example we should create top page `LAN` first. There is special operation “move a transition to a subpage” in `Hierarchy` palette (the first tool). As the result subpage tag is added to the transition, which is now a substitution transition. A new page is created with a copy of the places surrounding the target transition. The page is named after the target transition. This page is a pattern for creation of lover-level net. For instance applied to `WS3` in LAN example it creates the pattern:



You should edit the pattern to create the submodel of workstation. For instance, rename page `WS3` in `WS`, rename its places `p3out` in `LANin`, `p3in` in `LANout`, `aWS3` in `Own`, delete transition `WS3` and draw the net `WS` using created ports. Described tool executes assignment of subpage tag, port type tags and assignment of ports to sockets automatically. So it allows you the avoidance of a lot routine operations.

Notice that, the development of models requires both bottom-up and top-down approaches as tool “move a transition to a subpage” may be applied only once for each required subpage. For instance in such a way we may create `SWI, MWS` then `WS` for `WS1` and `S` for `S1`. But for `WS2-WS4` and `S2` we use early created `WS` and `S` in bottom-up way.

11. Analyzing a CP-net

CPN Tools provides two basic ways of models' analysis: simulation of net behavior and generation of state space. Moreover, you should be confident that model is adequate to the object and works in the proper way. That involves the preliminary stage very common for programming languages and usually named debugging. At this stage you acquire confidence that your model works properly and correct errors. We propose also a special way for models' analysis named measuring fragments. For estimation of models' characteristics we create special fragments of nets which calculate characteristics during simulation. Such a fragment is created in Appendix A5 for estimation of Ethernet response time.

11.1. Debugging of Models

Debugging of models involves syntax checking and step-by-step simulation. CPN Tools automatically syntax checks your nets as you create them or when you load in a net. You can see by color indications how far the check has gotten. The color indications are shown in the index, underlining the name of the page where the color belongs. If the page is open in a binder, the color is also shown in the page tab at the top of the page, and on the CP-net element where the color belongs. The orange aura indicates that an element is not currently checked. When you load a net, the syntax check takes a couple of minutes to complete. During this phase, the elements will change aura from orange to yellow to no aura (or red, if there is an error). If the orange aura stays, it is probably because either there is something missing or there is an error on a related net element. A yellow glow indicates that the place/transition/arc/page/net is currently being checked. Declarations are checked starting from the top. If a declaration depends on a later declaration, it will get an error saying the other declarations is not defined. Declarations with errors are rechecked when a change is made in any declaration. A red aura means the element has been checked but had an error. A speech bubble should appear with the exact error message. Elements connected to the element with the error, e.g. transitions connected to a place with errors, are not checked until the error is fixed. If there is an error in the declarations, the declaration with the error will be underlined with red. The net entry and all affected pages will also be underlined with red. To see the error message for a declaration with an error, move the mouse over the declaration.

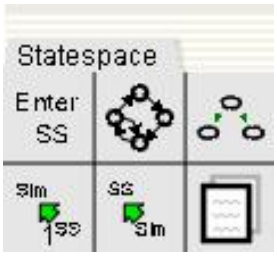
Step-by-step simulation described in Section 4.3 is used to trace the way of tokens in your model. For instance, you may trace frames in the Ethernet model described in Appendices on their way from workstation to server and backwards. You can also choose manually the bindings of firing transition parameters, for instance, make the choice of input token among available. For further debugging the execution of specified number of transitions is useful to estimate the behavior of the model on larger intervals of time.

11.2. State Space Analysis

State space of colored Petri net is more complicated than reachability set or reachability graph of classical Petri net. In classical Petri net marking of places is represented by vector of natural numbers but in colored Petri net by vector of multi-sets (timed multi-sets).

The analysis of state space is possible for rather small or simple models because of well-known effect of state space explosion. The number of states for l-bounded Petri net with m places is estimated as l^m. In telecommunications the analysis of state space is applied mainly at verification of protocols when we need knowledge about formal properties of nets such as boundedness, safeness, liveness etc.

State Space palette has the form:



It contains such tools as:

- Enter State Space;
- Calculate State Space;
- Calculate SCC graph;
- State Space to Sim;
- Sim to State Space;
- Save Report.

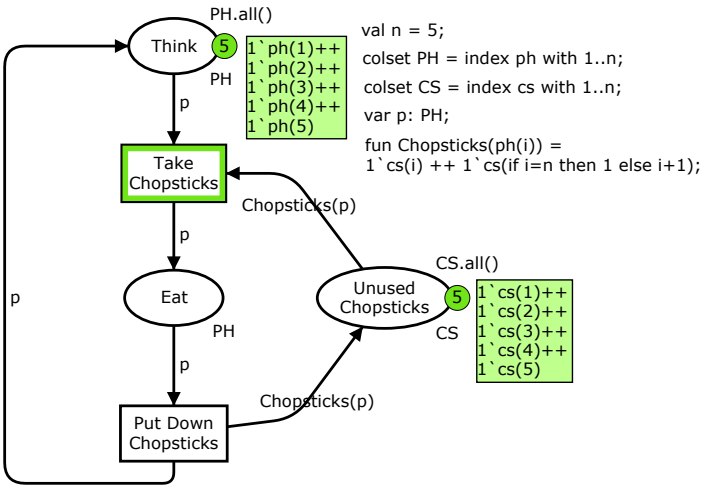
Before a state space can be calculated and analyzed, it is necessary to generate the state space code. This code is generated when you apply `Enter the state space` tool. Entering the state space tool will take some time. Then, if the state space is expected to be small, you can simply apply the `Calculate state space` tool to a sheet containing a page from the net. If the state space is expected to be large, you may need to change the options for the `Calculate state space tool`. The options for the `Calculate state space tool` allow you to determine when the calculation of a state space stops. In addition the strongly connected components (SCC) of state space graph may be calculated using corresponding tool. Calculated state space is stored in CPN Tools temporary files. There are two ways to analyze it:

- save report into a file;
- create State Space queries.

To save a report, you have apply the `Save state space report` tool to a sheet containing a page from the net. You enter the name of report file. The contents of the report are determined by the options for the `Save state space report`

tool. Queries are used to investigate properties of a CP-net by writing special CPN ML functions. They are quite complicated and use special predefined functions. CPN Help contains a reference to State Space Queries Manual.

Let us consider the well-known example of dining philosophers:



The model in CPN Tools has the compact representation due to usage indexed color sets for description of philosophers (`PH`) and chopsticks (`CS`) and function `Chipsticks` that returns the numbers of sticks used by philosopher `ph(i)`.

The saved report has the form:

```
 Statistics
------------------------------------------------------------------------

  State Space
    Nodes:  11
    Arcs:   30
    Secs:   0
    Status: Full

  Scc Graph
    Nodes:  1
    Arcs:   0
    Secs:   0

 Boundedness Properties
------------------------------------------------------------------------

  Best Integers               Bounds    Upper      Lower
  Page'Eat                    1         2          0
  Page'Think                  1         5          3
  Page'Unused_Chopsticks      1         5          1

  Best Upper Multi-set Bounds
  Page'Eat 1            1`ph(1)++
                        1`ph(2)++
                        1`ph(3)++
                        1`ph(4)++
                        1`ph(5)
  Page'Think 1          1`ph(1)++
                        1`ph(2)++
                        1`ph(3)++
                        1`ph(4)++
                        1`ph(5)
  Page'Unused_Chopsticks 1
                        1`cs(1)++
```

```
                        1`cs(2)++
                        1`cs(3)++
                        1`cs(4)++
                        1`cs(5)

  Best Lower Multi-set Bounds
  Page'Eat 1                  empty
  Page'Think 1                empty
  Page'Unused_Chopsticks 1    empty

 Home Properties
--------------------------------------------------------------------------
  Home Markings:  All

 Liveness Properties
--------------------------------------------------------------------------
  Dead Markings:  None
  Dead Transitions Instances: None

  Live Transitions Instances: All

 Fairness Properties
--------------------------------------------------------------------------
  Page'Put_Down_Chopsticks 1
                          Impartial
  Page'Take_Chopsticks 1 Impartial
--------------------------------------------------------------------------
```

Statistics section describes the size of state space and SCC graph. Boundedness section gives upper and lover bounds of markings in numerical and multi-set forms. Home properties section lists home markings. Liveness properties section describes deadlocks and live transitions. Fairness properties section describes type of net's fairness.

Notice that CPN Tools does not give a way to save complete generated state space. It is kept internally in CPN Tools temporary files. To investigate it beyond the bounds of the standard report you should write special queries to state space in CPN ML language.

11.3. Simulation of Net Behavior

CPN Tools may be used as a typical simulation system. When behavior of the net is rather complicated we may simulate it on large intervals of time and make conclusions about characteristics of the modeled system. Especially when random functions are widely used in a model we are more interested in its statistical properties than in its state space. For instance, we may simulate behavior of Ethernet model during one day of real time and make conclusions about such its characteristics as average response time, percentage of collisions etc.

From the point of view of standard simulation system CPN Tools gives us scarce facilities for such analysis. It has no tools for time management except of fast execution of a given number of steps in `Simulation` palette. But we can choose huge enough number of steps to provide time durations corresponding to hours of real time. Moreover, CPN Tools does not calculate elementary statistical information such as maximal, minimal and average number of tokens in places, frequencies of

transitions' firing etc. But it gives us the language to describe the processes of accumulation and calculation of characteristics. It is the same language of colored Petri nets and it may be applied for estimation of statistical characteristics of models. Such additional nets are named measuring fragments and studied in the next Section.

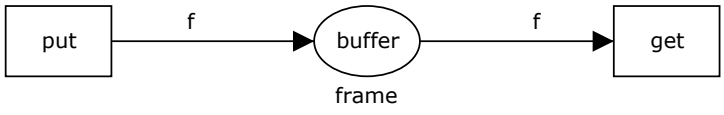
But simulation itself implies special consideration of experiments with model. The first thing is the *scaling of times*. Time in CPN Tools is measured in Model Time Units (MTU) which has no dimension and represented as a natural number. That's why we are interested in the scaling of times to make the model realistic. The example of times scaling is described in Appendix A7 for Ethernet model. We acquire times in real units (ms, ns) from the description of the hardware and software. Then we choose MTU as the smallest time duration. But for future development of the model it is reasonable to choose MTU smaller. For representation more fast equipment in future. For instance, the smallest delay for Ethernet model is 500 ns but MTU equaling to 100 ns has been chosen. Then all the times of the model were recalculated in MTU. For instance, 200 ms corresponds to 200000 ns / 100 ns = 2000 MTU. After obtaining the result times should be recalculated in real time units. For instance, average response time equals to 389 MTU or 38900 ns or 38.9 ms.

The second thing is the existence of the *state stable mode* of model's behavior. If the state stable mode exists the model is balanced. The increase of modeling time duration does not cause the significant change of its statistical characteristics. The simplest way to determine state stable mode is the sequential enlargement of modeling time duration. If characteristics do not change after certain moment of model time then the state stable mode has been achieved. Notice that state stable mode can not exist because source network is unbalanced; for instance, 100 Mbps flow directed to 10 Mbps network.

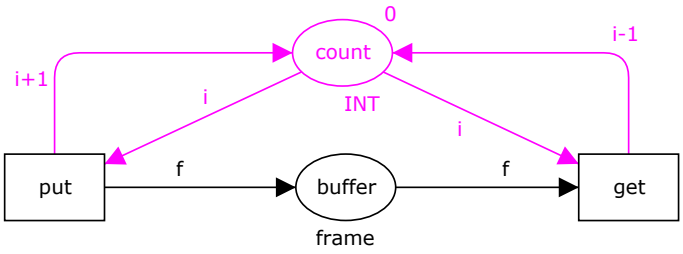
The third thing is the *estimation of averages* of characteristics in state stable mode. Suppose our model calculates characteristics. But the results taken in a single experiment with model are not valuable. A few experiments should be provided with the model and according to mathematical statistics its number should be about 20. In more complicated estimations the confidence interval should be taken into consideration.

11.4. Measuring Fragments

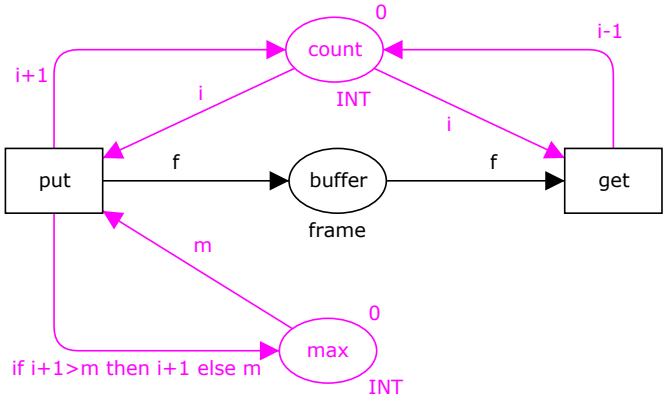
As the language of colored Petri nets constitutes a complete algorithmic system it may by applied for accumulation and calculation of statistical characteristics. Extra parts of net added to the source model for accumulation and calculation of statistical characteristics are named *measuring fragments*. Let us study a few simple measuring fragments. Let we have a buffer and transition `put` increasing its marking and transition `get` decreasing its marking:



We can easily calculate the current number of tokens in buffer with the following fragment:



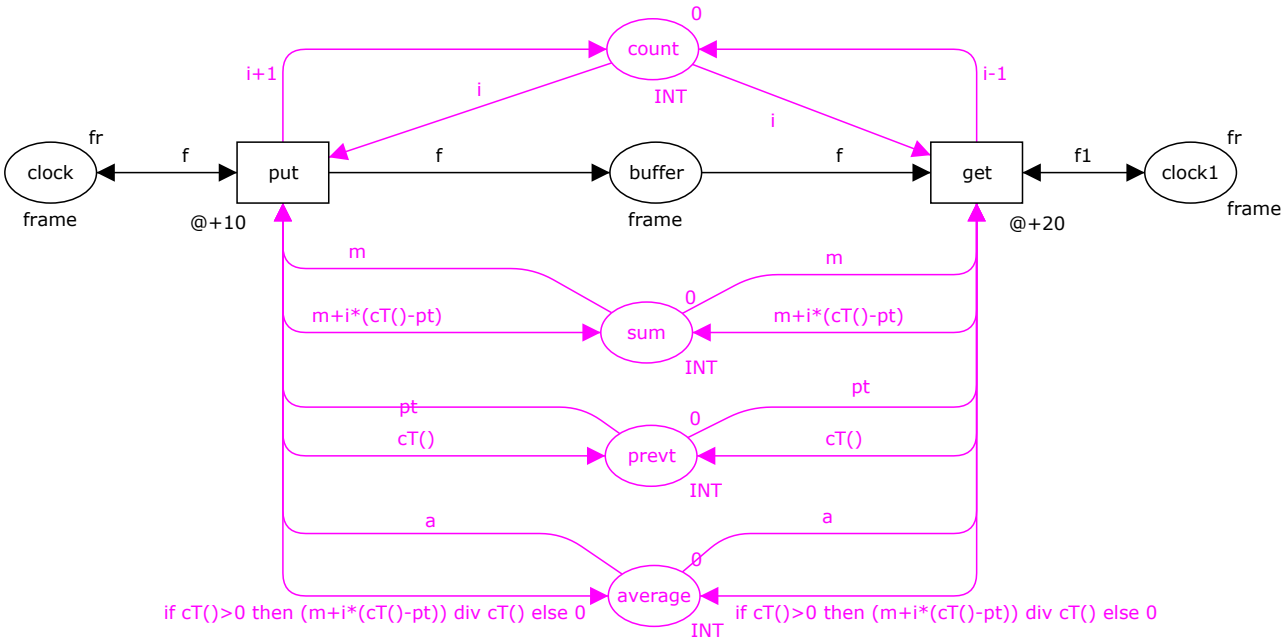
To calculate the maximal number of tokens in place `buffer` the following fragment may be used:



To calculate the average number of tokens, time intervals should be taken into consideration because average of distribution is calculated with formulae:

```
ac=(c1*dt1 + c2 *dt2 + … + ck*dtk) / dt
```

where `ac` is the average, `ci` is the value on time interval `dti` and `dt` is the total interval of time. The following fragment calculates average number of tokens in place `buffer`:



The recalculations are started both: at the increase (put) and at the decrease (get) of tokens number. Function cT() as described in Appendix A2 returns the current value of model time. Place `sum` keeps the current sum of products. Place `prevt` keeps the value of previous moment of time when the marking of place `buffer` had been changed. Place `average` keeps the average number of tokens in the place `buffer`.

More interesting example of measuring fragment for estimation of Ethernet response time is described in Appendix A5. Measuring fragments may be designed for estimation of networks' throughput as well as QoS characteristics.

12. Additional Features of CPN Tools

There are a lot of more specific features of CPN Tools described in online help as well as in separate documents, for instance, in the manual on ML language. This section contains an overview of most significant of them for modeling of telecommunication systems.

12.1. Unions

A union color sets is a disjoining union of previously declared color sets. It is very hard limitation that place contains tokens of the same color set; special union color set helps you to overcome this limitation. In union color set you may combine different color sets of tokens which you want to collect in the same place.

The declaration has the syntax:

colset name = **union** id1[:name1] + id2[:name2] + ... + idn[:namen];

If `namei` is omitted then `idi` is treated as a new value, and it can be referred to simply as `idi`. Simple operators are used to retrieve values of concrete color set in union:

`idi v` or `idi(v)`

where `v` has type `namei`.

In the example of Ethernet network described in Appendices union color set is used to model segments of Ethernet. If collisions are not considered that is the common case in completely switched Ethernet, then a segment is either idle or transmitting a frame. To distinguish this cases special union `seg` was described:

```
colset seg = union f:frm + avail timed;
```

Unit `avail` means that segment is free and available for transmission. In other case segment is transmitting a frame `f`. There is no simple way in CPN Tools to check a place on absence of tokens (inhibitor arcs) so color set `seg` is used. Let us consider the submodel of switch (fig. 4). Each input channel of switch ports `Port*In` extracts frame `f` and puts instead of it label `avail`. It means that transmission of

frame has finished and segment is available and ready for transmission of another frame. Each output channel of switch ports `Port*Out` waits for label `avail` before transmission, extracts this label and puts transmitting frame instead of it.

12.2. Lists

List color set constitutes the sequence of elements of the same color set. List is a variable-length color set. Standard functions allow the access to both ends of a list. To process elements inside list recursive functions should be used.

Declaration of list has the syntax:

```
colset name = list name0 [with int-exp1..int-exp2];
```

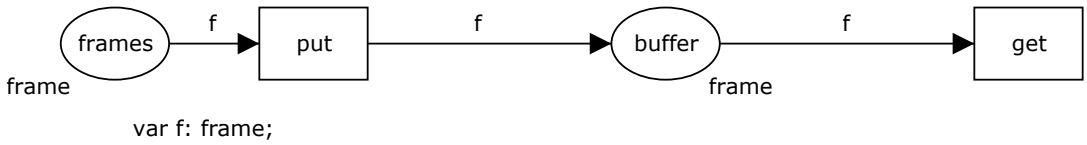
The `with` clause specifies the minimum and maximum length of the lists. Values of list color set have the form:

`[v1, v2, ..., vn]` where `vi` has type `name0` for `i=1..n`.

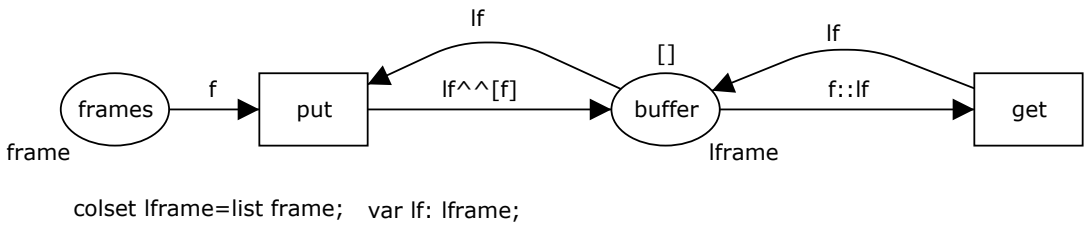
The following operations are available for lists:

`nil`	empty list (same as [])
`e::l`	prepend element e as head of list l
`l_1^^l_2`	concatenate the two lists l_1 and l_2
`hd` `l`	head, the first element of the list l
`tl` `l`	tail, list with exception of first element
`length` `l`	length of list l
`rev` `l`	reverse list l
`map` `f l`	use function f on each element in list l and returns a list with all the results
`List.nth``(l,n)`	nth element in list, where 0 <= n < length l
`List.take``(l,n)`	returns first n elements of list l
`List.drop``(l,n)`	returns what is left after dropping first n elements of list l
`List.exists` `p l`	returns true if p is true for some element in list l
`List.null` `l`	returns true if list l is empty

Usual place of CPN tools provides the discipline of random access because an arbitrary valid token may be taken by transition:



But telecommunication devices widely use queues with FIFO and priority disciplines, stacks with LIFO discipline etc. List color set helps us to organize the required discipline. Let us consider an example of FIFO queue creation:



You may put concrete frames into place `frames` and trace the behavior of this net to acquire the comprehension of list color set. Notice that in the last case place `buffer` contains only one token and this token is the list of frames. In the initial marking the list is empty `[]`.

In CPN Tools help LIFO and priority disciplines are considered. List allows also the representation of inhibitor arcs; corresponding examples are presented in CPN Tools help. More complicated examples with recursive functions usage are studied in paper [5].

Appendices:

An Evaluation of Network Response Time using a Colored Petri Net Model of Switched LAN

A1. Switched LAN

Recently the Ethernet has become the most widespread LAN. With gigabit technology it started a new stage of popularity. And this is not the limit yet. Hubs are dumb passive equipment aimed only at the connection of devices as wires. The base element of the Local Area Network (LAN) Ethernet (IEEE 802.x) is a switch of frames. Logically a switch is constituted of a set of ports. LAN segment (for example, made up via hub) or terminal equipment such as workstation or server may be attached to each port. The task of a switch is the forwarding of incoming frame to the port that the target device is connected to. The usage of a switch allows for a decrease in quantity of collisions so each frame is transmitted only to the target port and results in an increased bandwidth. Moreover the quality of information protection rises with a reduction of ability to overhear traffic. The scheme of sample switched network is presented in Fig. 1.

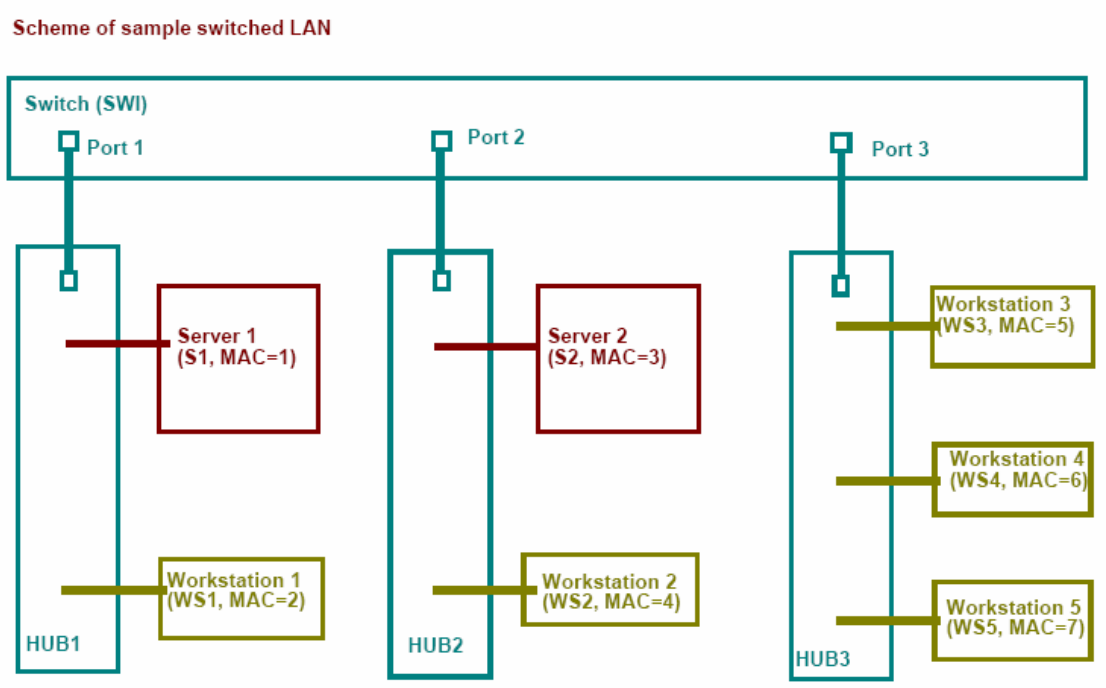


Fig. 1. Scheme of sample switched LAN

As a rule, the Ethernet works in a full-duplex mode now, this allows simultaneous transmission in both directions. To determine the target port number for the incoming frame a static or dynamic switching table is used. This table contains the port number for each known Media Access Control (MAC) address. Only static switching tables will be modeled in the present paper.

A2. Model of LAN

A model of sample LAN with topology, shown in Fig. 1, is represented in Fig. 2. Let us describe the model constructed. Notice that the model is represented with colored Petri net and consists of places, drawn as circles (ellipses), transitions, drawn as bars, and arcs. Dynamic elements of the model, represented by tokens, are situated in places and move as a result of the transitions' firing.

The elements of this model are sub models of: Switch (**SWI**), Server (**S**), Workstation (**WS**) and Measuring Workstation (**MWS**). Workstations **WS1-WS4** are the same type exactly **WS**, whereas workstation **WS5** is the type **MWS**. It implements the measuring of network response time. Servers **S1** and **S2** are the same type exactly **S**. Hubs are a passive equipment and have not an independent model representation. The function of hubs is modelled by common use of the corresponding places **p*in** and **p*out** by all the attached devices. The model does not represent the collisions.

Each server and workstation has it's own MAC address represented in places **aS***, **aWS***. A switch has separate places for input (**p*in**) and output (**p*out**) frames for each port. It represents the full-duplex mode of work. Bidirected arcs are used to model the carrier detection procedures. One of the arcs checks the state of the channel, while another implements the transmission.

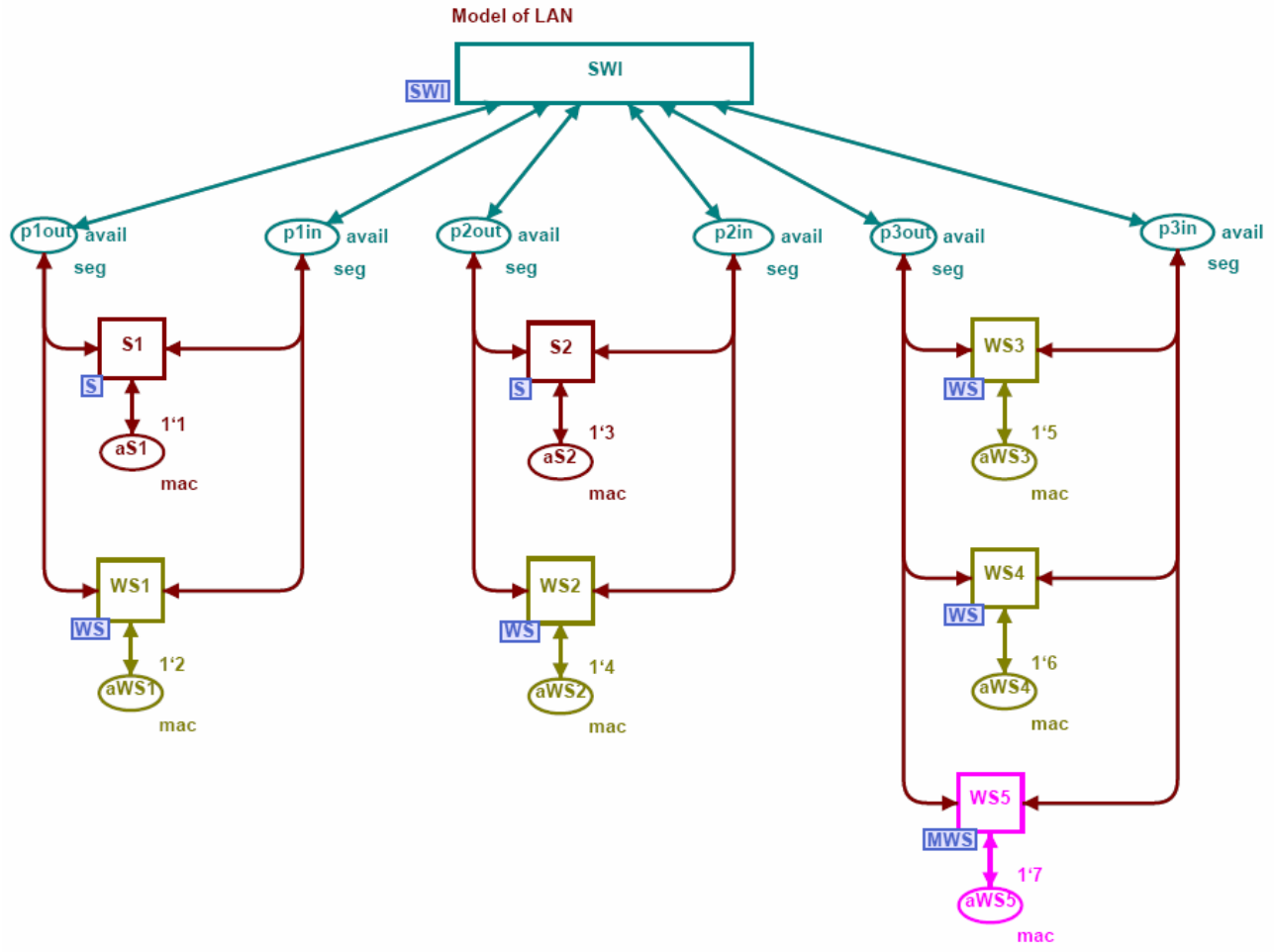


Fig. 2. Model of sample LAN

All the declarations of color sets (**colset**), variables (**var**) and functions (**fun**) used in the model are represented in Fig. 3. The Ethernet MAC address is modeled with integer number (color **mac**). The frame is represented by a triple **frm**, which contains source (**src**) and destination (**dst**) addresses, and also a special field **nfrm** to enumerate the frames for the calculation of response time. We abstract of other fields of frame stipulated by standard of Ethernet. The color **seg** represents unidirectional channel and may be either available for transmission (**avail**), or busy with transmission of a frame (**f.frm**). It is represented with a **union** type of color. Notice that the descriptor **timed** is used for tokens, which take part in timed operations such as delays or timestamps.

```
colset mac = INT timed;
colset portnum = INT;
colset nfrm = INT;
colset sfrm = product nfrm * INT timed;
colset frm = product mac * mac * nfrm timed;
colset seg = union f:frm + avail timed;
colset swi = product mac * portnum;
colset swf = product mac * mac * nfrm * portnum timed;
colset remsv = product mac * nfrm timed;
var src, dst, target: mac;
var port: portnum;
var nf, rnf: nfrm;
var t1, t2, s, q, r: INT;
colset Delta = int with 1000..2000;
fun Delay() = Delta.ran();
colset dex = int with 100..200;
fun Dexec() = dex.ran();
colset dse = int with 10..20;
fun Dsend() = dse.ran();
colset nse = int with 10..20;
fun Nsend() = nse.ran();
fun cT()=IntInf.toInt(!CPN'Time.model_time)
```

Fig. 3. Declarations

The marking of places is represented with multi-sets in CPN Tools. Each element belongs to a multi-set with defined multiplicity, in other words – in a few copies. For instance, the initial marking of the place **aWS2** is **1`4**. It means that place **aWS2** contains 1 token with a value of 4. The union of tokens is represented by a double plus sign (++). Tokens of timed color have the form **x @ t** which means that token **x** may be involved only after a moment of time **t**. So, notation **@+d** is used to represent the delay with the interval **d**.

A3. Model of Switch

Let us construct a model for a given static switching table. We consider the separate input and output buffers of frames for each port and common buffer of the switched frames. The model of switch (**SWI**) is presented in Fig. 4. The hosts' disposition according to Fig. 1 was used for the initial marking of a switching table.

The color **swi** represents records of switching table. It maps each known MAC address (**mac**) to the number of port (**nport**). The color **swf** describes the switched frames, waiting for output buffer allocation. The field **portnum** stores the number of the target port. The places **Port*In** and **Port*Out** represent input and output buffers of the ports correspondingly. The fusion place **SwitchTable** models the switching table; each token in this place represents the record of the switching table. For instance, token **1`(4,2)** of the initial marking means that the host with MAC address 4 is attached to port 2. The fusion place **Buffer** corresponds to the switched frames' buffer. Notice that a fusion place (such as **SwitchTable** or **Buffer)** represents a set of places. The fusion place **SwitchTable** is represented with places **SwTa1, SwTa2, SwTa3**. The fusion place **Buffer** is represented with places **Bu1, Bu2, Bu3**. It allows the convenient modelling of switches with an arbitrary number of ports avoiding numerous cross lines.

The transitions **In*** model the processing of input frames. The frame is extracted from the input buffer only in cases where the switching table contains a record with an address that equals to the destination address of the frame (**dst=target**); during the frame displacement the target port number (**port**) is stored in the buffer. The transitions **Out*** model the displacement of switched frames to the output ports' buffers. The inscriptions of input arcs check the number of the port. The fixed time

delays (@+5) are assigned to the operations of the switching and the writing of the frame to the output buffer.

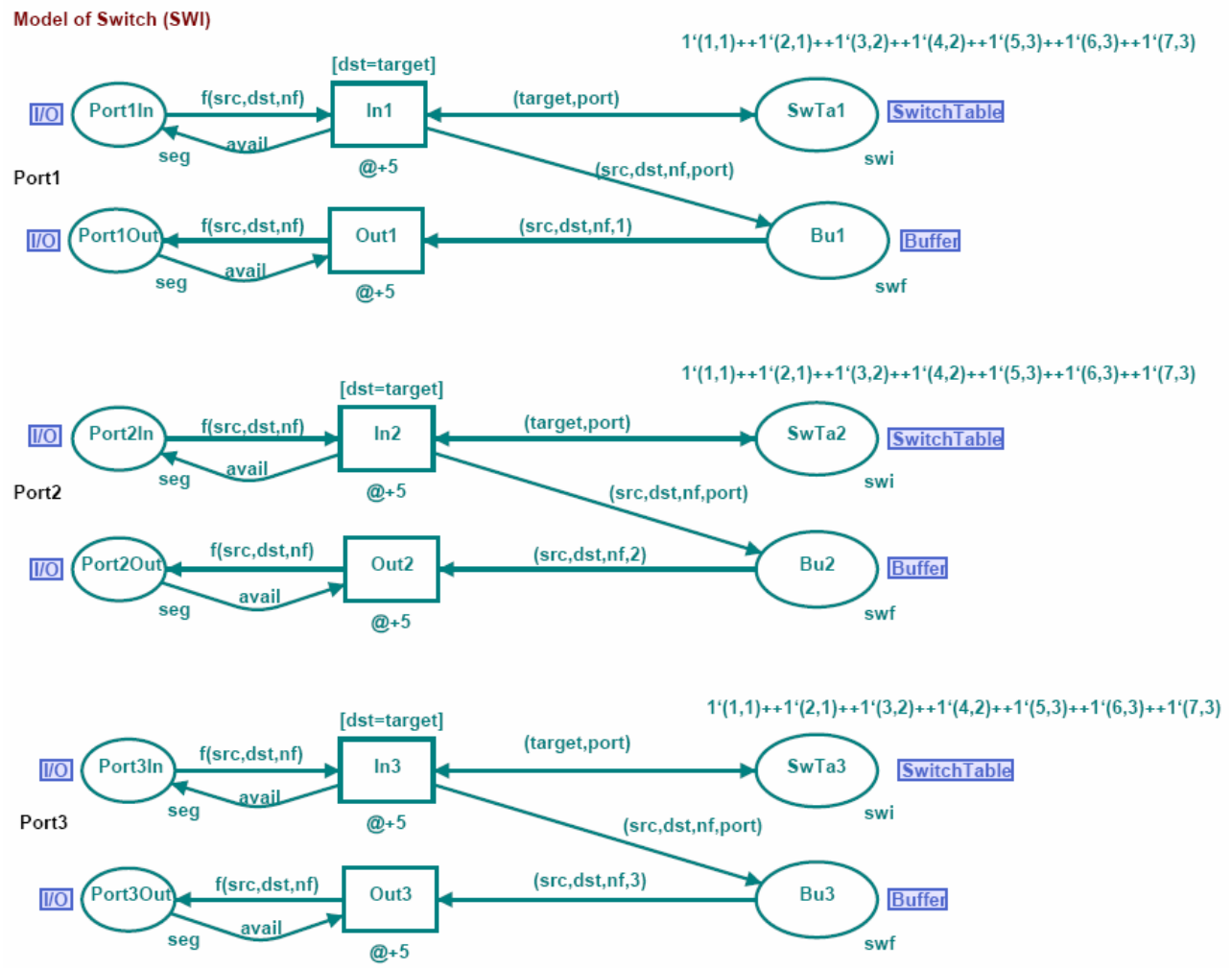


Fig. 4. Model of switch

It is necessary to explain the CSMA procedures of LAN access in more detail. When a frame is extracted from the input buffer by transition **In*,** it is replaced with the label **avail**. The label **avail** indicates that the channel is free and available for transmission. Before the transition **Out*** sends a frame into a port, it analyses if the channel is available by checking the token **avail**.

Notice that places **Port*In** and **Port*Out** are contact ones. They are pointed out with an **I/O** label. Contact places are used for the construction of hierarchical nets with substitution of transition. For example, the transition **SWI** in the top-level page of model (Fig. 2) is substituted by a whole net **SWI** represented in Fig. 3. Places **Port*In** and **Port*Out** are mapped into places **p*in** and **p*out** correspondingly.

A4. Models of Workstation and Server

To investigate the frames' flow transmitting through LAN and to estimate the network response time it is necessary to construct the models of terminal devices attached to the network. Regarding the peculiarity of the traffic's form we shall separate workstations and servers. For an accepted degree of elaboration we consider periodically repeated requests of workstations to servers with random uniformly distributed delays. On reply to an accepted request a server sends a few packets to the address of the requested workstation. The number of packets sent and the time delays are uniformly distributed random values.

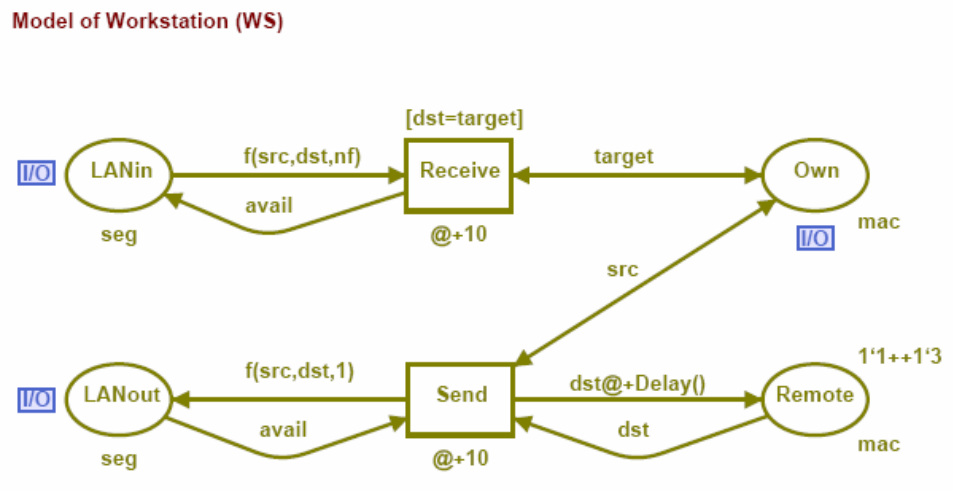


Fig. 5. Model of workstation

A model of workstation (**WS**) is represented in Fig. 5. The places **LANin** and **LANout** model the input and output channels of the local area network correspondingly. The workstation listens to the network by means of transition **Receive** that receives frames with the destination address, which is equal to the own address of the workstation (**dst=target**) saved in the place **Own**. The processing of received frames is represented by the simple absorption of them. The workstation sends periodic requests to servers by means of transition **Send**. The servers' addresses are held in the place **Remote**. After the sending of a request the usage of the server's address is locked by the random time delay given by the function **Delay()**. The sending of the frame is implemented only if the LAN segment is free. It operates by checking place **LANout** for a token **avail**. In such a manner the workstation interacts with a few servers holding their addresses in the place **Remote**.

Notice that the third field of frame, named **nfrm,** is not used by the ordinary workstation **WS**. The workstation only assigns the value of a unit to it. This field is used by a special measuring workstation **MWS**. The copies of the described model **WS** represent workstations **WS1-WS4**. To identify each workstation uniquely, the contact place **Own** is used. This place is shown also in the top-level page (Fig. 2) and contains the MAC address of host.

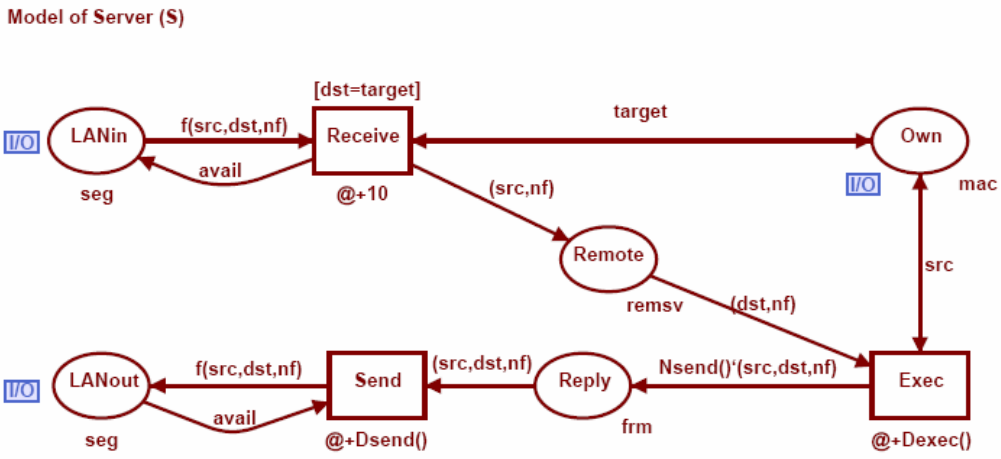


Fig. 6. Model of server

A model of server (**S**) is represented in Fig. 6. The listening of the network is similar to the model of the workstation but it is distinct in that the frame's source address is held in the place **Remote**. The transition **Exec** models the execution of the workstation's request by a server. As a result of the request execution the server generates a random number **Nsend()** of the response frames, which are held in the place **Reply**. Then these frames are transmitted into the network by

the transition **Send**. Notice that the request number **nf** is stored in the place **Remote** also. It allows us to identify the response with the same number as the request.

A5. Model of Measuring Workstation

A model of the measuring workstation (**MWS**) is represented in Fig. 7. In essence, it is an early considered model of workstation **WS**, supplied with the measuring elements (the measuring elements are drawn in magenta).

Let us consider the measuring elements in more detail. Each frame of a workstation's request is enumerated with a unique number contained in the place **num**. The time, when the request was sent, is stored in the place **nSnd**. The function **cT()** calculates the current value of the model's time. The place **nSnd** stores a pair: the frame's number **nf** and the time of request **cT()**.

The place **return** stores the timestamps of all the returned frames. As the network response time we consider the interval of time between the sending of the request and receiving the first frame of response. This value is stored in place **NRTs** for each responded request. The transition **IsFirst** determines the first frame of response. The inscription of the arc, connecting the transition **IsFirst** with the place **NRTs**, calculates the response time (t2-t1).

A residuary part of the measuring elements calculates the average response time. The places **sum** and **quant** accumulate the sum of response times and the quantity of accepted responses correspondingly. The arrival of a new response is sensed by the place **new** and initiates the recalculation of average response time with the transition **Culc**. The result is stored in the place **NRTime**.

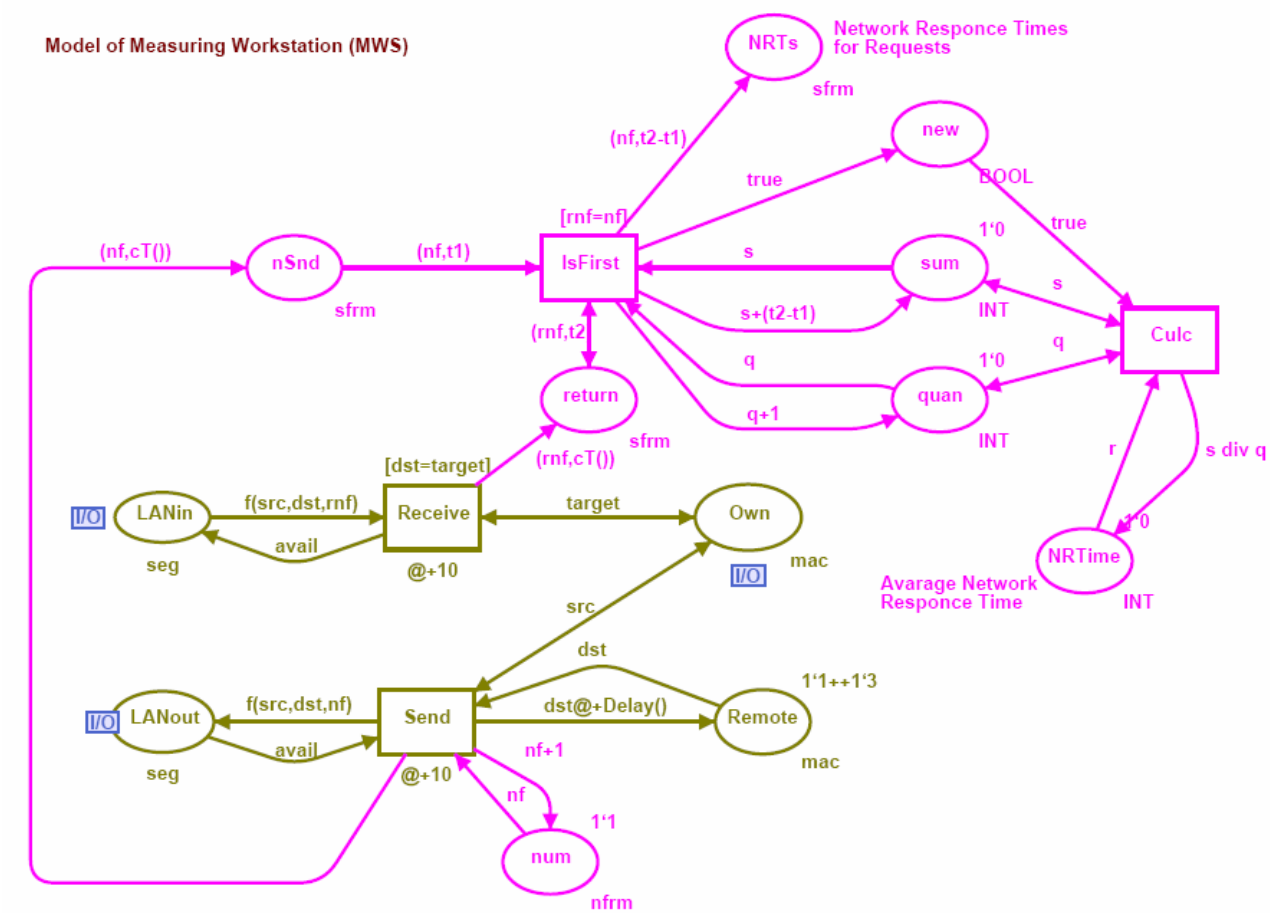


Fig. 7. Model of measuring workstation

A6. Evaluation Technique

The model constructed was debugged and tested in a step-by-step mode of simulation. For these purposes the frame generated by the workstation was traced through the network to the server and back. Also we observed the behavior of the model in the process of automated simulation with a display of net's dynamics – in the mode of the so-called game of tokens. It allows us to estimate the model with a glance at the top-level page and at sub pages during simulation.

To estimate the network response time precisely, rather huge intervals of model time are required. It is convenient for such purposes to use the simulation mode without displaying intermediate marking aimed at the accumulation of statistics.

A snapshot of the measuring workstation model is represented in Fig. 8. The rectangular labels (drawn in bright green) describe the current marking of the simulation system; the circular labels contain the number of tokens. The place **LANin** contains frame (1,5,1). The place **LANout** represents the available state of the channel **avail**. The number of the next request, according to the marking of place **num**, is 7. The place **return** indicates that 83 frames of responses have arrived. The place **NRTs** contains the response times for each of the 6 responded requests. For instance, the network response time for request 5 equals to 235. It should be calculated easily, that the average network response time 389 in the place **NRTime** equals to 2337/6 according to the markings of the places **sum** and **quant**.

A7. Parameters of Model

The right choice of time unit for model time measurement is a key question for an adequate model construction as well as the calculation of timed delays for elements of the model. It requires an accurate consideration of the real network hardware and software characteristics.

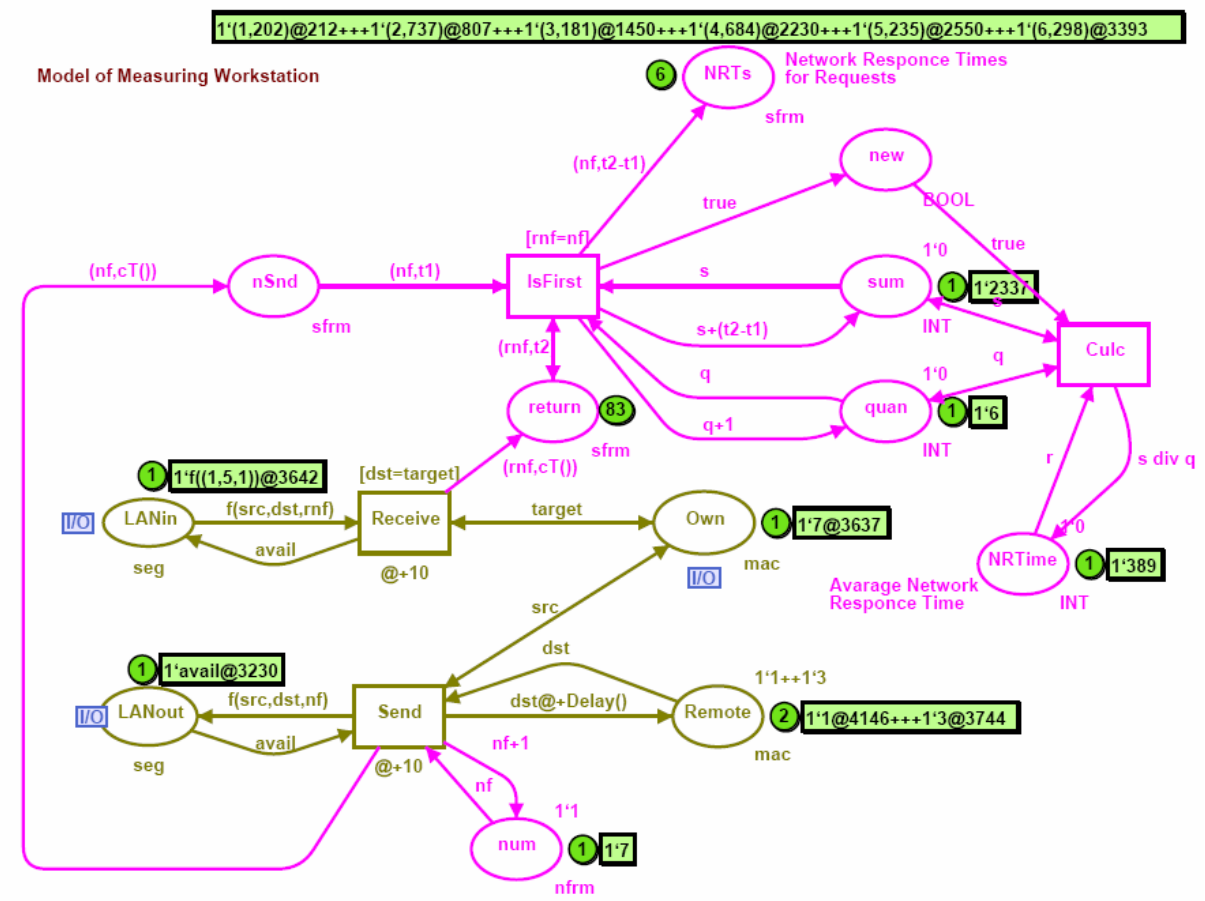


Fig. 8. Estimation of network response time

The scheme shown in Fig. 1 represents a fragment of a railway dispatch centre LAN supplied with special railway CAM software GID Ural. The core of the system constitutes a pair of mirror servers **S1** and **S2**. The workstations **WS1-WS5** are situated in the workplaces of railway dispatchers.

We have to consider the performance of the concrete LAN switch and LAN adapters to calculate the timed delays of transitions **In***, **Out***, **Send**, **Receive**. Moreover, the peculiarities of client-server interaction of GID Ural software ought to be considered for the estimation of such parameters as delay between the requests **Delta** and the time of request execution **dex**. Since the unit of information transmitting through net is represented with a frame, we have to express the lengths of messages in numbers of frames. For these purposes the maximal length of an Ethernet frame equaling 1.5 Kb was chosen.

The types of LAN hardware used are represented in Table 1.

Table 1. Types of hardware

Device	Type
LAN adapter	Intel EtherExpress 10/100
LAN switch	Intel SS101TX8EU
Server	HP Brio BA600
Workstation	HP Brio BA200

In Table 2 the parameters of the model described are represented. LAN switch and adapter operations are modelled with fixed delays so they are small enough in the comparison with client-server interaction times. Moreover, in reliable Ethernet frames of maximal length are transmitted mainly, since the time of frame's processing is a fixed value. Stochastic variables are represented with uniform distribution, which corresponds to Ural GID software behavior. The smallest timed value is the LAN switch time of read/write frame operation. But for the purposes of future representation of faster equipment we choose the unit of model time (MTU) equaling 100 ns.

Table 2. Parameters of model

Parameter	Variable/Element	Real value	Model value
LAN switch read frame delay	In*	500 ns	5
LAN switch write frame delay	Out*	500 ns	5
LAN adapter read frame delay	Receive	1 ms	10
LAN adapter write frame delay	Send	1 ms	10
Server's time of request processing	Dex	10-20 ms	100-200
Client's delay between requests	Delta	100-200 ms	1000-2000
Length of request		1.2 Kb	1
Length of response	Nse	15-30 Kb	10-20

Thus, the average network response time obtained equals 389 MTU or about 39 ms. This delay satisfies the requirements of train traffic control.

References

1. Jensen K. Colored Petri Nets - Basic Concepts, Analysis Methods and Practical Use. – Springer-Verlag, 1997. – Vol. 1-3. – 673 p.
2. Albert K., Jensen K., Shapiro R. Design/CPN: A Tool Package Supporting the Use of Colored Nets // Petri Net Newsletter. – April 1989. – P. 22-35.
3. Zaitsev D.A. Switched LAN Simulation by Colored Petri Nets // Mathematics and Computers in Simulation. – 2004. – Vol. 65, № 3. – P. 245-249.
4. Zaitsev D.A. An Evaluation of Network Response Time using a Colored Petri Net Model of Switched LAN // Proc. of Fifth Workshop and Tutorial on Practical Use of Colored Petri Nets and the CPN Tools, October 8-11, 2004. – Aarhus (Denmark). – 2004. – P. 157-167.
5. Zaitsev D.A., Shmeleva T.R. Switched Ethernet Response Time Evaluation via Colored Petri Net Model // Proc. of International Middle Eastern Multiconference on Simulation and Modelling, August 28-30, 2006. – Alexandria (Egypt). – 2006. – P. 68-77.