Internal Report 2010-05 March 2010

Universiteit Leiden
Opleiding Informatica

Demonstration Tools

for

Petri Nets

Wouter de Zwijger

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Demonstration Tools for Petri Nets

Wouter de Zwijger
Supervisor: Jetty Kleijn

Leiden Institute of Advanced Computer Science
Universiteit Leiden
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract. We search for a tool that can be used in education to support the understanding of petri nets. We determine the constraints such tools should satisfy, compare tools, and create new content for the tool with the best comparison result.

Key words: petri net, tools, pipe2, pipe2 modules, EN systems, p/t-net, petri net tool comparison

1 Introduction

Petri nets are useful representations of all kind of distributed systems, and especially useful to monitor concurrency between parts of a system. Petri nets have a graphical representation, but are also mathematically exact defined. These properties make petri nets a very useful tool to represent concurrent systems and their behavior.
A nice feature of petri nets is that their behavior not only can be computed by the underlying mathematics but also can be visualised, which gives an intuitive idea of what is going on in a net. This visualisation of the behavior of a petri net is based on a token game. You can play this on plain paper with a draft of the petri net and a few coins, but it would be nicer to have a program to show you this visualisation.
Such programs do exist, in fact many are freely available on the internet, and they can do much more than simple simulation. For instance based on the underlying mathematics they can compute certain properties for a petri net, show special graphs that include the behavior of all possible visualisations and much more.
However there exist different kinds of petri nets, and these vary in complexity. Some tools are more focused on the more complex petri nets, the so called higher level petri nets, while others on the more simple variants.
Since the way of education is to start simple, in most courses simple petri net models would be introduced before the more complicated nets. Unfortunately these simple nets are very limited from a practical point of view, and since most advanced tools are designed for the purpose of development of applications, and

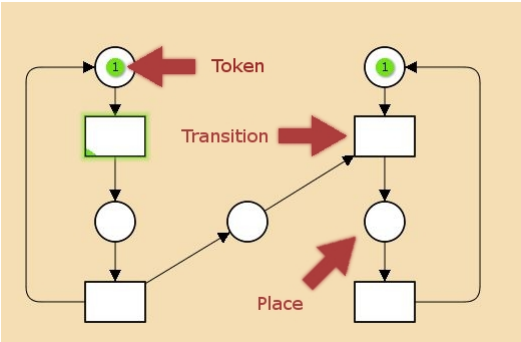

Fig. 1. A petri net

not for educational use, these tools most of the time don't support the simple nets. This is why finding tools with advanced features and supporting the simplest types of petri nets can be challenging, and rewarding for better education of petri nets in the future. Such tools could be used for a student lab, hands on experience with petri nets, visualisation of the theory and for homework exercises.
This is what this paper is about; providing an overview of available tools for simple, low level petri nets, selecting the tool most suited for our purposes, and drafting some examples and exercises with this tool and where necessary extending it.
To get an overview of available and useful tools we first formulate basic criteria that all tools that will be examined should satisfy. Secondly a list of criteria for comparison is compiled. Then a limited number of tools is examined in detail. After carefully comparing the examined tools, we select one tool which then may be altered, to match it even better with the desired purpose. Finally, for educational purposes, a number of examples and exercises is implemented in this tool.
In chapter 2 we will discus some basic terminology of petri nets. In chapter 3 we will determine the criteria for finding and comparing tools. In chapter 4 the determined criteria from chapter 3 will be investigated for each tool and plotted in a table. This table will support our choice of the right tool. In chapter 5 the tool chosen in chapter 4 will be described in more detail as well as any work done to the tool to make it more suitable for the desired purposes.

2 Preliminaries

In this section we list notions [1] [2] used in the paper.

Petri net Petri net is the name for a group of different models. They have in common that they have a net as their underlying structure.

Definition 1. *A net* is a triple $N = (P, T, F)$, where:
(1) P and T are finite sets with $P \cap T = \varnothing$,
(2) $F \subseteq (P \times T) \cup (T \times P)$.

T is the set of *transitions* of N and P is its set of *places*. A transition is here, as usual, graphically represented by a rectangle and a place is drawn as a circle, see also figure 1.
Transitions may be connected to places and places to transitions. These connections are visually represented by arrows, as connections have a direction.

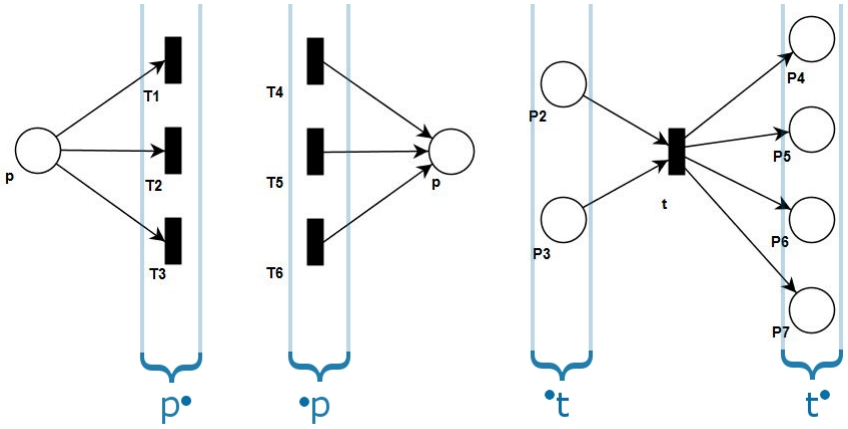

Fig. 2. Dot notation

dot notation For a place p all transitions that p is connected to form the set $p^\bullet$ (output transitions of p). Conversely, ${}^\bullet p$ is the set of transitions that are connected to place p (its input transitions). Similarly, for a transition t, we write ${}^\bullet t$ for the set of places connected to t (its input places) and $t^\bullet$ is the set of places to which t is connected (its output places).

p/t-net A *place transition net*, p/t-net for short, is defined as follows:

Definition 2. *A* place/transition net *or p/t-net, is a tuple*
$M = (P, T, F, W, K, C_{in})$*, where*
(1) (P, T, F) *is a net,*
(2) $W : F \to \mathbb{N}_+$ *is the* weight *function,*
(3) $K : P \to \mathbb{N}_+ \cup \omega$ *is the* capacity function
(4) $C_{in} : P \to \mathbb{N}$ *is the initial configuration.*

Here ω is a special symbol denoting that a place p with $K(p) = \omega$ has unbounded capacity: $\forall n \in \mathbb{N} : n \leq \omega$.

A p/t-net has an underlying net with weights added to its arrows, capacities to its places, and it has an initial "state". This can be explained as follows. First we consider the concept of "state".

Definition 3. *A* configuration *of a p/t-net is a function* $C : P \to \mathbb{N}$*, such that* $\forall p \in P$ *: if* $K(p) \in \mathbb{N}_+$*, then* $C(p) \leq K(p)$*.*

configuration A configuration of a p/t-net can be seen as a distribution of "tokens" over the places. When you have a configuration of a net, you know for all places how many tokens they contain in that configuration (never more than their capacity).

tokens Tokens represent (graphical) information regarding local states (as represented by the places). Tokens are visualised by dots, so each dot represent one token. A place can contain a number of tokens, represented by multiple dots in that place.

initial configuration In a p/t-net the *initial configuration* is the configuration the net has when none of its transition has been fired.

capacity The amount of tokens a place can maximally hold is the capacity of a place. A place can not have a negative amount of tokens, nor fractions of a token. For example, a place can have 10 tokens, as long as its capacity is 10 or higher. A place can never have a token number like -10,5 since a place can't hold a negative amount of tokens nor fractions of tokens.

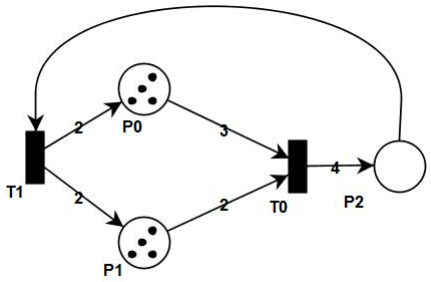

Fig. 3. A p/t-net with weights

weight Arrows between places and transitions regardless of their direction, can have a weight. This is a positive number. Graphically this is represented by a number next to the arrow, see figure 3. If there is no number given for an arrow, its weight is one. Notice that when a $p \notin {}^\bullet t$ then $W(p,t) = 0$, and when $p \notin t^\bullet$ then $W(t,p) = 0$.

enabled Transitions in a p/t-net can be *enabled*. An enabled transition when it does occur, changes the configuration of a net. For a transition t to be enabled both ${}^\bullet t$ and $t^\bullet$ must satisfy certain conditions.

Basically a transition is enabled if for all places in ${}^\bullet t$ their current number of tokens minus their input weight to t is zero or higher. And for all places p in $t^\bullet$ their current number of tokens plus the weight of the arc from t to p minus the weight of the connection from p to t does not exceed the capacity of p.

Definition 4. *For a p/t-net* (P, T, F, W, K, C_{in})*, A transition* $t \in T$ *is* enabled *at a configuration* C *if*
(1) $\forall p \in P \ : W(p,t) \leq C(p)$ *and,*
(2) $C'(p) = [C(p) - W(p,t)] + W(t,p) \leq K(p)$

firing Until now p/t-nets only have some structure, existing of places and transitions, connected with each other by arrows. A configuration of a p/t-net tells how many tokens each place has. But the real action comes when you allow a p/t-net to change from one configuration to another one. This means the structure is still the same, only the tokens are distributed differently. This change from one configuration to another one occurs when a transition is *fired*. A transition can only fire when it is *enabled*. If in a configuration C a transition t is fired, the new configuration will be C' as defined in definition 4(2). In words this means for all places p in ${}^\bullet t$ will lose $W(p,t)$ tokens and all places q in $t^\bullet$ will gain $W(t,q)$. Thanks to (2) in definition 4, all places will satisfy their capacity constraint in the new configuration.

token game The token game, see figure 4, starts at the initial configuration. Then enabled transitions may fire. After an enabled transition has fired, a next transition may fire. The process of firing transitions and changing configurations is called the token game.

firing sequences Transitions can fire in sequence. A firing sequence is only valid for a p/t-net if the first transition in this order can fire in the initial configuration, and every other transition in this sequence can fire in the configuration the net is in after previous transitions have fired. For instance, firing sequence t2,t3,t1 can occur in a given p/t-net if t2 can fire from the initial configuration and after t2 has fired, t3 is enabled and after t3 fired, t1 is enabled.

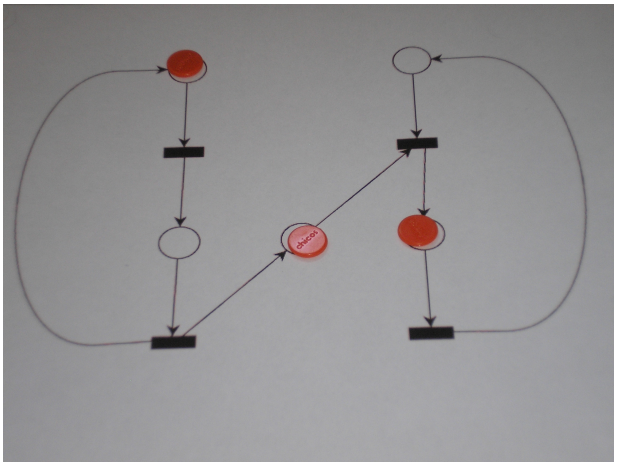

Fig. 4. A token game

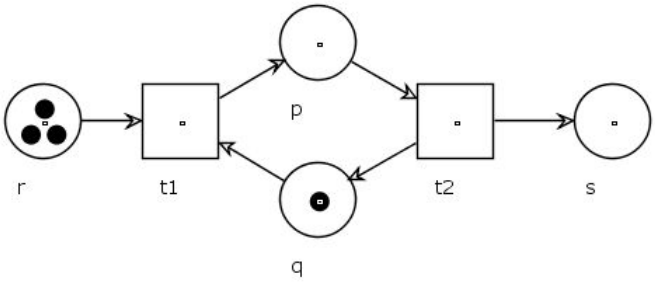

Fig. 5. q is the complement place of p. Therefore p will never have more than one token

complement places Complementarity of places is defined as follows:

Definition 5. *Let* M *be an p/t-net and let* $p, q \in P_M$. *Then* p *and* q *are* complementary*, denoted by* p ***com*** q*, if*
(1) $p^\bullet = {}^\bullet q$ *and* ${}^\bullet p = q^\bullet$
(2) $\forall t \in {}^\bullet p \; W(t,p) = W(q,t)$
(3) $\forall t \in p^\bullet \; W(p,t) = W(t,q)$
(4) $K(p) = K(q)$

See figure 5, take place p and its complement place q. When a transition t fires and p loses a token, q will get a token since ${}^\bullet p = q^\bullet$ and $W(p,t) = W(t,q)$. Also whenever q loses a token, then p will gain a token and vice-versa because $p^\bullet = {}^\bullet q$ and $W(q,t) = W(t,p)$. Thus the sum of tokens of a place and its complementary place is always the same. These complementary places are useful to limit the

token amount in a place without having to use capacity.

configuration graph While the token game is a nice visualisation of the dynamics of a net, this is not the most efficient nor complete way. Not all possible transitions enabled in the token game are fired in a single token game (firing sequence). It can take quite some runs and in the end trouble to find all possible configurations a p/t-net can reach from its initial configuration (potentially infinitely many!).
Instead of playing a lot of token games, one may use a *configuration graph* which is a single object representing the behavior of a net. This graph has the 'reachable' configurations of the net as its nodes. An arrow indicates a change from one configuration to another one, caused by the firing of a transition. Since the only thing that (dis)allows a transition to fire is the configuration, it does not matter what transitions are fired to get to a certain configuration and thus a state in the configuration graph. This means that two complete different firing sequences of transitions can end up in the same configuration, and from there have the exact same transitions enabled. A good thing about the configuration graph is that it gives a complete view of the behavior of a p/t-net and (when finite) allows checking for certain behavior and properties of that net.

elementary nets A p/t-net is only one of the many different types of petri nets. Other types of nets have slightly different mechanisms, however the core is the same. Higher level nets can have variables instead of tokens. Or the weights on arrows can contain conditions that must hold. They could also impose more enabling conditions on transitions, for instance based on time. On the other side of the p/t-nets there are the *elementary nets*. These nets can be seen as special cases of p/t-nets. However, elementary nets are entirely motivated by concurrency properties, where p/t-nets and other net models are used for modeling and describing real world systems and processes.

Definition 6. *An elementary net system, EN system for short, is a quadruple* $M = (P, T, F, C_{in})$*, where:*
(1) (P, T, F) *is a net*
(2) $C_{in} \subseteq P$ *is the initial configuration.*
(3) for every $t \in T$ *there exist* $p, q \in P$ *such that* $(p, t), (t, q) \in F$*,and*
(4) for every $t \in T$ *and* $p, q \in P$*, if* $(p, t), (t, q) \in F$*, then* $p \neq q$*.*

For all transitions t of an EN system, ${}^\bullet t$ and $t^\bullet$ are never empty. Visually this means a transition has always an arrow from the transition and an arrow to this transition.
While p/t-nets have no restrictions on connections, EN systems have. A self loop, where ($p \in {}^\bullet t$, $p \in t^\bullet$) is not allowed in an EN system.

However the main difference between p/t-nets and EN systems is that the places of an EN system have an implicit capacity of 1 and thus never can hold

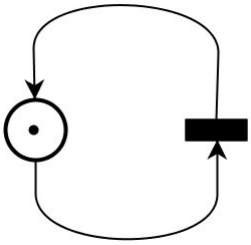

Fig. 6. A self loop, where the place has capacity one

more than one token. Thus the configurations of an EN system can be seen as subsets of its places. This also means that it would be useless to have arrows with weights more than one.

Definition 7. *For an EN system* (P, T, F, C_{in})*, A transition* $t \in T$ *is enabled in configuration* C $(C \subseteq P)$ *if*
(1) $\forall p \in {}^\bullet t \ : p \in C$
(2) $\forall q \in t^\bullet \ : q \notin C$
(3) $C' = C - {}^\bullet t + t^\bullet$

Formally a transition t of EN system M is enabled at $C \subseteq P$ when all input places of t are in the configuration i.e. contain a token, and all output places of t are not in the configuration i.e. contain no tokens. And when t fires, all input places of t will lose their token and all output places of t will gain a token.
In case an arrow has a weight greater than one, it will never be used, simply because places can't have more than one token.

The slightly different firing rules as described in (1) and (2) of definition 7 for an EN system explain why a self loop should not be permitted. If there would exist a place that is both input and output place for a certain transition, that transition will never be enabled. Because a place can't contain a token and be empty the same time. But a self loop as shown in figure 6 can fire in a p/t-net, even if the place has capacity one. This is because the condition $[C(p) - W(p,t)] + W(t,p) \leq K(p)$ holds. $1 - 1 + 1 = 1$, which equals the capacity $K(p)$. The firing rule of p/t-nets and EN systems goes in harmony as long as the net does not contain a self loop. An EN system is not allowed to have such loops. The firing rule of an EN system can be replaced by the firing rule of p/t-nets, if all place capacities and all weights are one and all transitions have at least one input and one output place. Thus we can now define EN systems as a special case of a p/t-net.

Definition 8. *An elementary net system, EN system for short, is a p/t-net where*
(1) $\forall p \in P, \ t \in T \ : W(p,t) \leq 1$

(2) $\forall p \in P,\ t \in T\ : W(t,p) \leq 1$
(3) $\forall p \in P\ : K(p) = 1$
(4) $t^\bullet \cap {}^\bullet t = \emptyset$
(5) $\forall t \in T\ : {}^\bullet t \neq \emptyset$ *and* $t^\bullet \neq \emptyset$

safeness A p/t-net is *safe* if it has no reachable configuration with a place holding more than one token. This can easily be checked by looking at the *configuration graph*; which is finite and all configurations have not more than one token in any place. It is obvious that an EN system is always safe, since it is not allowed for a place in an EN system to have more than one token.

boundedness When all places in a net never exceed a given finite amount of tokens, the net is *bounded*.

Definition 9. *a p/t-net* (P, T, F, W, K, C_{in}) *is* bounded *if there exists a* $k \in \mathbb{N}$ *such that* $\forall C \in \mathbb{C}_M, \forall p \in P, C(p) \leq k$

The configuration graph of a bounded p/t-net is finite. Moreover, if a p/t-net is not bounded, then at least one place can always contain more and more tokens. And all these different amounts of tokens in this place require a different node in the configuration graph. Because the place will have infinite amount of different number of tokens, the *configuration graph* will have an infinite amount of states, and thus can't be constructed. If the configuration graph is infinite, the *coverability graph* can be constructed instead. This type of graph does not show exact configurations in its states, but a generalisation of configurations instead.

liveness Another property visible in the *configuration graph* is liveness. A transition of a net is *live* when in any configuration you can get from the initial configuration, there is a firing sequence that results in enabling this transition. This is visible in the configuration graph, if finite, by simply checking for every state if it is possible to get to a state from which the given transition leaves. If the configuration graph is infinite, the coverability graph can be used instead.

contact-free An important property of EN systems is *contact-freeness*. This is defined as follows:

Definition 10. *Let* $M = (P, T, F, C_{in})$*be an EN system.*
M is contact-free *if for all* $t \in T$ *and* $C \in \mathbb{C}_M$*, if* ${}^\bullet t \subseteq C$ *then* $t^\bullet \cap C = \emptyset$.

So in a contact-free EN system, whenever a transition has its input places filled with sufficient tokens, all its output places are empty. So to check enabledness of a transition, it is sufficient to only check input places for the presence of a token.

3 Tools for petri nets

Tools for petri nets exist in many forms, from simple dos applications to complicated programs with nice and polished graphics. A tool can specialise in one kind of petri net, or be more general and applicable to more types of petri nets. Some tools allow the user to manually fire transitions, showing the petri net in action, while other tools only examine the net and come with a conclusion of properties that hold for the examined net.

3.1 Basic requirements

Many petri net tools are available on the internet, however, not all are suitable for our purposes. Between tools there is a difference on how easy it is to get such a tool working. For instance, tools can be for free, or have to be bought. Whether or not a tool is paid for, one should be able to install it. Installing can be made easy by running a simple install wizard, hard by manually doing the installation or not at all, because of components incapable to work together. These differences regarding the process of getting a tool working on one's own machine, form the basic requirements.

Tools that don't cost money and are available for download online have the preference, since they will be easy to get and thus for students easy to download on their own machines. Even if a tool doesn't cost money, they sometimes requires a license. This requires an extra step (requesting a license) and is therefore not preferred.

Furthermore a tool should be easy to install, to make it more attractive for students to install on their own machines. Easy to install means not too many manual things to do. For instance Yasper (see Appendix A for more details) requires a .net platform to install. While this is not hard to install, Yasper seems to require an old deprecated version of the .net platform. The installer of Yasper refuses to run until this old .net version is installed. Installing Yasper requires one to find an old version of .net and install it before one can install Yasper. Installing older versions of something in general is not a good idea as it might disturb other programs that use this, as well as vulnerability for old exploits fixed in newer versions. Therefore Yasper failed the basic requirements and is not included in this research from this point.

Another criterium is that the tool must have some graphical representation of the petri nets it supports. Basically this is necessary because you would like the tool to be a bridge between the graphic representation and the mathematical layer of the net. This critirium eliminates all dos based tools as candidates for our end goal.

In case no decent tool can be found, then these minimal criteria may have to be modified.

With these initial criteria determined, we now have to figure out what further properties are important for our use of the tool. The tools will be compared with respect to these properties. This should and will make it possible to find

the best tool for the job.

3.2 Functional requirements

net types The first property for which we will compare the tools concerns the supported net types. This follows directly from the reason why we search for a tool; most tools do not support the net types we would like it to support, EN systems and p/t-nets. If one or both net types are not supported, we would like to know if it would be possible by manipulating the code, or by other smart tricks, to implement these net types.

graphical environment While our initial search criteria already require that all tools at least must have some sort of graphical representation of the supported petri nets, it is also a good property for comparison. How clear are the graphical represented nets, how does the tool as a whole look, how are properties of a net shown, etc.

token game Visualisation of the token game, as defined in chapter 2. This is a desired property because it allows the user to play with the net, and gives an easier more intuitively way of understanding the operations of nets.

configuration graph The configuration graph is an important visualisation of the behavior of a net, and can be used to find many properties in a net. Not only the possibility to generate such a graph is important, but also the representation of this graph can be very important, since a bad representation can make it unnecessarily hard to grasp the net's behavior.

net properties Each petri net has an underlying mathematical definition, and therefore calculations can be done to find certain properties of that net. While these properties can often be extracted from the graphical representation and configuration graph, it would be a great help if a tool can automatically tell whether or not a net has certain properties. Examples of properties that a tool could help to determine are: boundedness, liveness, and deadlock.

extendability While a tool can be close to perfection, it is always a good thing if it can be adjusted to meet one's specific needs. Modification may be realised in many ways. Newly built-in manipulating properties or built-in script recognition would be fine. Access to source code would be optimal, since this means you can change anything, when required.

3.3 Using the criteria

Now that we have determined criteria and properties to be investigated, it is time to find candidate tools that could qualify. As a first attempt, a search was carried out using the global search engine Google. In this way we found a nice list [3] on the petri net world [4] website. Unfortunately, It is not very clear from this list whether a selected tool has a graphic representation of the petri nets, and whether it is possibly to let the tool work with elementary nets. The same site allows you however to search [5] for properties of the tools. Using this search page, we have chosen a few tools for investigation.

3.4 Tools selected

Although we have limited already the number of tools to be considered, it is still not possible to investigate each of the remaining one's due to time constraints. The tools eventually selected have been chosen because of promising descriptions on their home pages or the search page described earlier (3.3). These tools are listed here. For more detailed information about these tools see Appendix A.

- CPNtools
  a tool mainly for high level petri nets, with a lot of features.
- WoPed
  a petri net tool for work flow analysis
- NetLab
  a petri net tool for p/t-nets
- Platform Independent Petri Net Editor 2 (pipe2)
  petri net tool for timed and p/t-nets
- PetriLLD
  limited tool for elementary nets
- (collection of java applications)
  small applications with little capabilities, but supporting EN systems

4 The comparison table

Every tool has its own good and bad points, where some are more relevant to us than others. Based on a few important points, table 1 was constructed. It lists for each of the six selected tools its score for the requirements from subsection 3.2. In addition, there is a column headed "source code". This is included to indicate whether the tool can be modified and new features added. The second column indicates for which types of petri nets, a tool can be used. As explained earlier, we are especially interested in the support for EN systems and p/t-nets. The types are indicated by a letter, as given by the list to the left of the table. We also acknowledge EN system support if the tool allows transitions without either input or output. However, a tool that accepts firing of self loops, i.e. a place which is both input and output to the same transition, is not considered

Symbol	Representing net
A	Elementary Net Systems
B	Place Transition Nets
C	Timed Petri nets
D	High level petri Nets
X	Workflow Net

Application	Source code	Type Nets	Configuration graph	Analysis	Capacity	Weights	Invariant	Graphic environment	Token Game	Interface
CPNtools		C,D	+			✓		✓	✓	++
WoPeD	✓	B,X	+	✓			✓	✓	✓	+
NetLab	✓	B	-		✓	✓	✓	✓	✓	±
Pipe2	✓	B,C	++	✓	✓	✓	✓	✓	✓	+
PetriLLD	✓	A						✓	✓	-
Java Applications		A,B			✓			✓	✓	+

Table 1. The tools compared

as supporting EN systems. Note that if a net supports capacity and p/t-nets it is possible to simulate EN systems by adding some extra constraints to the p/t-nets.

The third column concerns the ability to generate a configuration graph representation. Tools might represent these configuration graphs in different ways, from simple text to nice polished graphs with small pop-up information if the mouse hovers above a node. The required steps a user most do to generate a configuration graph can also vary from a click on one button to assigning some special properties to a net and generating the configuration graph step by step. Since these steps and representations make quite a difference between the tools, the comparisons between the quality of the configuration graph is shown by +'s and -'s where – is worst and ++ best, and empty is no configuration graph support at all.

The column table headed "Analysis" concerns the ability of a tool for establishing properties such as boudedness, safeness and deadlock.

"Capacity" indicates the possibility to give each place an independent capacity, with the tool guaranteeing that capacity will never be exceeded.

The column "Weights" represents the possibility to assign weights on the arcs.

The "Invariant" property is met when the tool is able to generate place and/or transition invariants. An invariant is a property that is true for all configurations of a net.

The columns "Graphic environment" and "Token game" are included in the table for the sake of completeness. However the tools selected for further examination and thus the tools included at the table should have at least a graphic environment and token game, so all tools included at the table meet these features.

Finally, the column "Interface" refers not only to the layout but to "look-and-feel" in general, i.e. is everything going smoothly, don't you need to click a lot of times before the tool responds, does it support drag and drop. But also does

the tool look good, are the created nets clear and well arranged.

4.1 Selecting the right tool

As described in 3, there are quite a number of tools but not all qualify as useful tools. Some tools are obviously less useful for the purpose we want to use them for than others. The six tools we selected and our criteria led to table 1.

Using the criteria we can eliminate tools one by one till the best suitable tool is left over. It is very important to have support of EN systems or adequate possibility to simulate this behavior. For CPN with no access to source code the only possibility would be to use complement places. This because CPN tools does not check for output capacity. Even a simple net will require many complement places to behave as a EN system i.e. a complement place for each place in the net. Nets will become unnecessary complex and hard to understand. Thus CPN tools is dropped.

WoPed also does not check for output limitations. It however is open source and can be modified to do this. Still it might be not a good idea to actually mess with the built-in firing rule, as it could trigger unexpected behavior and malfunction of certain components of the tool. Therefore we decided to keep WoPed as a backup; if all other tools fail, WoPed is the best to use. But we prefer a tool with capacity support or EN systems, so we would have only to put some restrictions on the net or do nothing at all. The simple Java applications are great for familiarisation with simple EN systems, but are very simple and do not support a little more advanced stuff like configuration graph. Because this is very important for discovering certain properties from a net, it is desirable to have this kind of functionality. The same applies for PetriLLD, a simple tool with EN support but lack of advanced features. Therefore both the simple java applets and PetriLLD are dropped. Probably the reader has noticed that we haven't any tool left that supports EN systems. As said earlier, a good way to simulate EN behavior is sufficient. We can implement an EN system ourselves using the ability of p/t-nets and setting the capacity for all nodes to one. Of course this does not affect the side conditions like a self loop or transitions without input or output. But if we have the source code or at least some ability to modify the tool we can implement EN systems correctly for these side conditions. This means that the only two tools that we have tested that have the functionality of both configuration graph and p/t-nets with capacity support are NetLab and Pipe2. NetLab has a configuration graph, but this is presented in text and not really easy to use where Pipe2's configuration graph is displayed as a graph which is very clear. Since Pipe2 has a nicer configuration graph, we drop Netlab. Pipe2 is open source under Open Software License 3.0 (OSL3.0) and so if any functionality is missing it can always be added by changing the source code. Since it requires less modifications in the source code to go from a p/t-net supported tool with capacity support compared to a tool without capacity support, we chose Pipe2 over WoPed. And thus we chose Pipe2 to use as our tool. For a detailed description of each tool, see appendix A.

5 Pipe2 as the selected tool

5.1 tool summary

general info Pipe2 is a graphically petri net tool that runs on java. It supports p/t-nets and timed nets. Pipe2 and its source code are freely available for downloading.

general features Pipe2 has a token game, supports drag and drop, allows you to save and load a net, allows to set capacity for places, zooming in on nets.

advanced features Pipe2 comes with a number of default modules. Modules can also be written and added manually. This allows users to add functionality without modifying the original program. In general all modules apply their functionality on the currently opened net, but are able to allow the user to load a net from a file to apply their functionality on.

The default modules can do many different things e.g. constructing the configuration graph or finding the invariant of a net. For more detail about all pipe2's default modules, as well as more info about pipe2, see appendix A.

5.2 adapt the tool for final use

Now that we have selected a tool, we investigate its possibilities for classroom support in more detail. First we modify pipe2, to implement EN system support. It is possible to view nets as EN systems by simply letting all places have capacity one. These nets should not have self loops nor transitions that have neither input nor output. This allows us to implement many EN system examples that can be used in lectures. However, we would like the tool to formally interpret a net as an EN system.
Pipe2 works with modules, modules can examine an opened net and even modify it if desired. If we can create our own module that validates a net to have the structure of an EN system, we would have formally implemented EN systems in pipe2.

writing a module In order to write and use new modules in pipe2, one should first figure out how modules are implemented in pipe2. Creating modules is done by programming them in java. A module is a class that is an extension of a predefined interface. The interface forces modules to have at least the following two functions:
getName(), which returns a string containing the name of the module. This name will be displayed in the list of modules when running pipe2.
And run(DataLayer pnmldata), which will be called when the module is activated by the user.
The given datalayer object is a class containing all info a module could need

about the currently opened net. It also allows the module to fire transitions within this datalayer, save and reload configurations, and possibly even manipulate the given datalayer and thus the selected net. The interface of a module is in general a collection of pre-defined widgets, and so creating a new module interface exists mostly of copying and slightly modifying the interface of an existing module. Most modules in pipe2 exist of three parts;
A load petri net widget, allowing the user to load another net for the module than the current net in the opened tab.
A html based widget, mostly used for returning some results after calculation and analysis of a specified net.
A button to start the analysis process.
Some modules also have extra input text areas and check boxes.

Adding new modules to pipe2 should be done by using the *find module* feature when pipe2 is running. However, this feature requires a new module to have a .properties file that specifies how the new module should be installed. Unfortunately limited documentation about the syntax of this file type is available. It appears that using similar syntax from already existing module .properties files for new modules did not work. Placing a new module in the modules folder of pipe2 did work, and became the work-around to add new modules.

the EN validation module Using the basic layout and structure of the built-in modules, we have implemented a module that will make sure that an input p/t-net is actually an EN system. The module achieves this goal by a mixture of two methods. One, all aspects of the input p/t-net are checked if they satisfy the conditions for an EN system. And two, not satisfying nets may be changed (adapted).

Since changing the net will mean it may modify its behavior, it is preferred to change as little as possible. And only when the user didn't intend the net designed to work that way. Thus when a user designs a net we want to have the module to point out which EN properties are not satisfied. But it will change only a few automatically.

To decide what will be modified by the module and what will only be checked, we need to take a closer look at the creation process of a net by an user. A p/t-net is created in Pipe2 by creating, dragging and dropping elements in a screen. Places that are created have a default capacity of infinite. The user could change this by right clicking the place and specify another capacity. This means an extra step, which would be required to create an EN system, since an EN system has only places with capacity one. To eliminate this extra step it is convenient to let the module change these place capacity automatically. Once places and transitions are created, a user can drag arcs between them. The arrows have by default a weight of one, the same as for an EN system. It would require an extra step to have a non EN system with weighted arcs. Thus a user who does, might have a reason for this. Therefore the module should tell the user it is invalid for an EN system but not automatically change it by itself. Tokens are easily

added to places by using the scrolling wheel of the mouse. While scrolling, it is clear how many tokens created in a place. Once again, if a place has more than one token, the user has probably done this on purpose. Thus the module checks for the token limitation for EN systems, but won't change it automatically. As described in the preliminaries, we define EN systems as nets without self loops. Since a user probably has created a self loop on purpose, the module should only check for this condition.
When a net contains objects that are not part of an EN system, and possibly not even of a p/t-net, it can be assumed these are added on purpose. One should therefore check for the absence of these objects. An example of such an object is a timed transition (see pipe2 in Appendix A for more detail about these timed transitions).

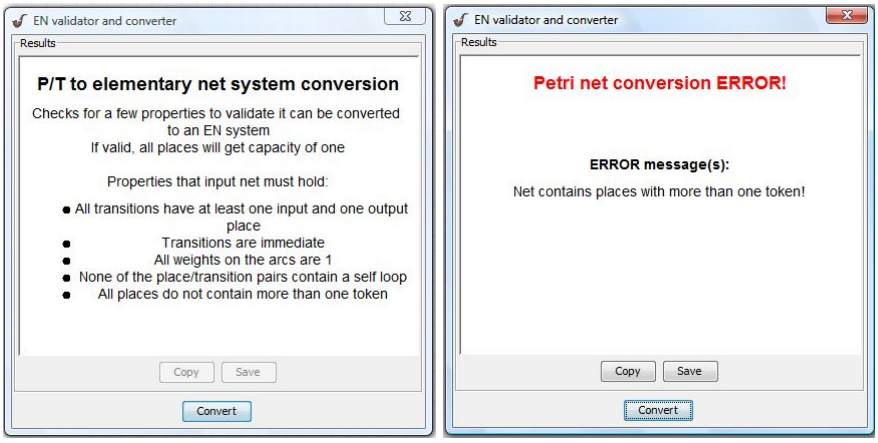

Fig. 7. EN system validation module

Now that is determined what will be checked and what will be corrected by the module, we summarise the duties of the validation module. The module should:

- check if no place contains more than one token
- check for self loops
- check for transitions without either input or output places
- check if no transition is timed
- check if all arcs have weight one
- if all of the above hold, set the capacity of all places to one,
  return that the net is now an EN system.
  else report an error with explanation why it is not a EN.

The written source code of this module is included in Appendix B.

processes module As a demonstration that pipe2 can be extended to better support teaching classes, we implement a new module with some new functionality. Due to time restrictions, we have chosen to implement a simple module just as a proof of concept.
Given an EN system and a firing sequence the simple module creates a process [2]. The output will be a new net placed in a new tab in pipe2.
A *process net* is defined as:

Definition 11. *A net $N = (P, T, F)$ is a process net if:*
(1) N is acyclic, and
(2) $\#({}^\bullet p) \leq 1$ and $\#(p^\bullet) \leq 1$ for all $p \in P$.

This means that places do not branch. In other words transitions do not share output places nor input places. One transition's input place can still be another transition's output place.
A process for a contact-free EN can be constructed using a valid firing sequence. The resulting process is a *labelled* process net that may have multiple places labelled with the same name, but it will not contain any loops. A process can be used to see relations between transitions, like whether or not a certain transition must have fired before another one can fire.
The process net defined by a firing sequence of an EN system is constructed as follows:
(1) For each place that has a token in the initial configuration add a place labelled with the name of the original place to the process.
(2) Then for the next transition t in the given firing sequence:
(2.1) create in the process a new place for each place from $t^\bullet$, labeled with the name of that original place.
(2.2) Add a new transition labelled with t to the process net.
(2.3) Then connect to this transition all places labelled with a name from ${}^\bullet t$ and without any outgoing arrows.
(2.4) Connect this new transition with the last created places in the process representing $t^\bullet$.
(3) Repeat step 2 until all transitions in the sequence have been handled.
Since the process construction adds transitions only in terms of tokens in their input places, we have to ensure that the input net is contact-free. This property is checked automatically by the process module. If the given firing sequence is invalid, this will be detected during construction and the module will inform the user that the firing sequence was incorrect.

bugs Unfortunately pipe2 is not a perfect tool; it has bugs. Bugs that caused too much trouble have been fixed, as described below.

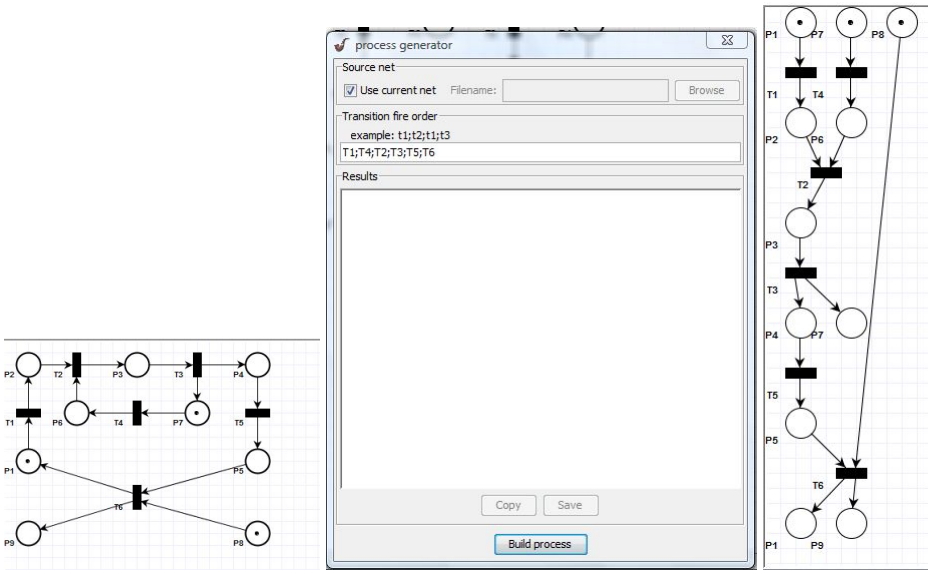

Fig. 8. The process module

load bug While creating example content for this tool, there appeared a bug that could be quite bad. It seemed that for some loaded petri nets their layout was somehow ruined, see also figure 9. After a long examination of the code, a solution was found. The following two command lines in the place class constructor, called when a place is created, should have been switched;

```
setCapacity(capacityInput);
setCentre((int)positionX, (int)positionY);
```

setCapacity updates the capacity, but it also calls an update function that would update the location of the object. However, since setCenter, a function that places the object to the given x and y coordinate is called after the place has already been replaced to a new position, things go wrong. This does not happen when places do not have a finite capacity since setCapacity will not call the update function if the capacity is the default unlimited capacity. Switching the two lines solves the problem. Apparently updating the location is allowed when the place already has a location given by the setCentre function.

minimal siphons and traps bug When a tool is equipped with modules that can find properties for you, then you want to be able to use those modules. However, it is quite a disappointment if their output is not correct.
A siphon is a collection of places of a p/t-net, for which the total number of tokens will never increase. So during a token game a siphon can lose tokens, but it can never gain tokens. Hence once it is out of tokens, it will be for the rest of the token game. A minimal set of siphons is a non-empty siphon which does not

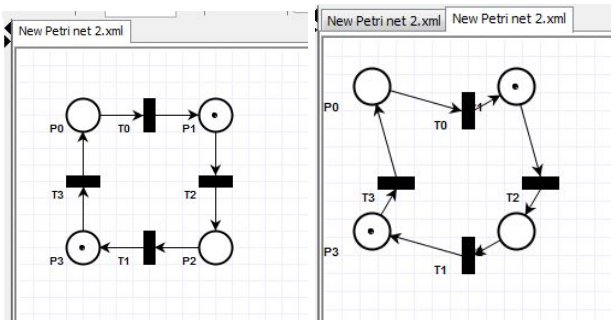

Fig. 9. left: the original net. right: the same net, after it was reloaded from disc

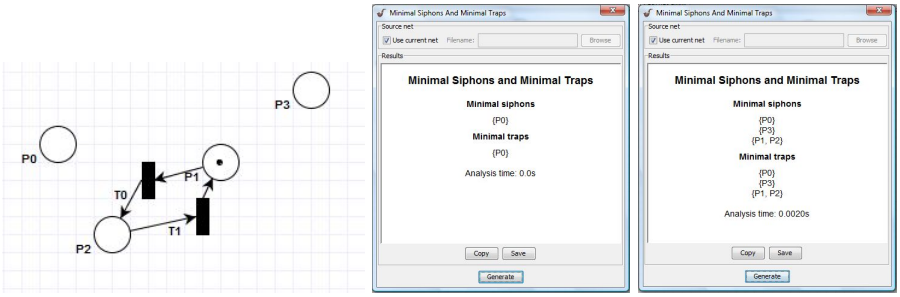

Fig. 10. For the net on the left, the siphon module returned the result showed in the middle image, but should have returned the result shown in the image on the right

contain another non-empty siphon. A trap is the opposite of a siphon; such a group of places can never have less tokens than it had in a previous configuration.

The minimal siphons and traps module finds with a click of the mouse the minimal siphons and traps. A really nice module but when there are isolated places, the output will exists of only one isolated place, even if there are other siphons. This is a bug, since the module returns good results for nets without isolated places.
The code of this module for finding the siphons and traps exists of two parts. A while loop is called to detect isolated places. These places are both traps and siphons. After the wile loop the rest of the code will find collections of places that form siphons or traps. The while loop uses a few wrong variables, which results that after the while loop is done, the rest of the code can't execute correctly. This results in the bug when there are isolated places around. See the following code for the bugged while loop:

```
/**
    * findAllMinimalSiphons()
```

```
     * Finds all minimal siphons in a given Petri Net that contain a specific
     * set of places.
     * @param G      A Petri Net
     * @param Ptilde A set of places that each siphon must contain
     * returns       A vector containg all minimal siphons found
     */
    private Vector <boolean[]> findAllMinimalSiphons(PetriNet G, SetOfPlaces Ptilde){
       Vector <boolean[]> E; // contains all minimal siphons found
       SetOfPlaces S = new SetOfPlaces(Ptilde.size()); // a siphon

       // Step 1
       E = new Vector();

       // Step 2
       // check if there is a place with an empty pre-set
       int p = G.placeWithEmptyInputSet();
       while (Ptilde.isEmpty() && (p != -1)){

          // Step 3
          S.add(p);
          E.add(S.getPlaces());
          // the Petri Net can be reduced by eliminating p
          G = reduceNet(G, Ptilde.getPlacesMinus(p));
          // check if there is another place with an empty pre-set
          p = G.placeWithEmptyInputSet();
       }
```

The first problem in this loop is in the code line:
`G = reduceNet(G, Ptilde.getPlacesMinus(p));`.
One wants to reduce the places of net G by deleting p. Thus the code should be:
`G = reduceNet(G,G.P.getPlacesMinus(p));`
where G is correctly reduced.

The second problem is how the code stores isolated siphons. Siphons are stored by adding S (which contains a collection of places forming a siphon) to E (the collection of all found minimal siphons). S is reused in each loop cycle. This reuse however requires S to get emptied after its contents has been added to E. Neglecting this results in S containing places of previous found siphons. Thus by adding `S.remove(p);` after `E.add(S.getPlaces());`, S can safely be reused. Since siphon and trap results both use this function, both are fixed by correcting this while loop, and the module provides correct results.

5.3 pipe2 as educational support

visual feedback One of the things the tool pipe2 can do to improve education is giving understandable visual feedback. While petri nets can easily been drawn on a blackboard, simulating the actual behavior of the net goes not that

smoothly. With pipe2 one can not only show large nets, one can also run the net, reset it and run it again with slightly different initial settings. This will aid students to see what happens in a net, and the effect of small changes. Moreover, using the module for the configuration graph, it is always possible to compare the configuration graph with the net behavior.

assignments Not only for visual display, pipe2 can also be used to either solve normal assignments or for new assignments. One strong point of pipe2 is the configuration graph. As explained earlier, this graph contains a lot of information about a net. For large nets, it can consume a lot of time to create a configuration graph manually. Therefore assignments requiring the construction of configuration graphs for large nets are avoided. With pipe2 a configuration graph is created with a mouse click, regardless of net size. Therefore analysing the configuration graph is one of the things that can be done more often and in more detail.

Pipe2 can also be used to analyse some properties of a net, such as siphons and traps, which allows more assignments to focus on such properties. These analyses can also be used by students when stuck trying to solve traditional assignments.

Another strong point is the ability to create one's own modules. This allows creation of extended analytic tools closer to the educational content.

implementing draft assignments Assignments could be created with a different focus. Assignments could be used to get more familiar with pipe2, its GUI and its possibilities. Assignments could be used to examine and analyse a net, which is probably the most useful type of assignment. A third category to use pipe2 assignments would be to design nets that must satisfy certain properties.

Now follow some possible example exercises with output from pipe2.

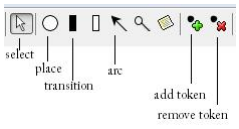

Fig. 11. GUI for creating a net in pipe2

Exercise 1 getting acquainted with pipe2 1. Open pipe2. Using the GUI of pipe2, create the producer-consumer net as shown in figure 12. Make sure you use the solid black transitions, and not the white ones, since these have a special meaning in pipe2.

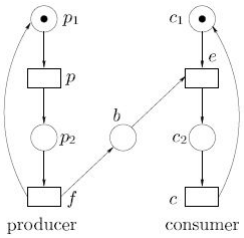

Fig. 12.

2. Pipe2 default firing rules are not the same as the firing rules of an EN. To make sure pipe2 uses the EN firing rules, one should convert the net to an EN system. Use the module, located left in the GUI of pipe2, called 'EN validator and converter' to convert the created net to an EN. If correctly applied, all places should have a thicker line around them than before validation.
3. Place a token in both the consumer and producer part of the net. Press the green flag in pipe2's GUI. Pipe2 is now in animation mode, and the token game can be played. Try pressing transitions in the net, what does (not) happen? What is the meaning of transitions that are colored red?
4. Use the Reachability module (located in the list left in pipe2) to create a graph. Each blue node indicates a certain configuration. Each arc represents the firing from the transition shown on the arc. Is the created graph the configuration graph of the producer-consumer net? Why (not)?
5. Find all the concurrent steps of the producer consumer net.
6. When the mouse hovers above a blue node in the reachability graph, more information about this configuration will be displayed. *Edges from* indicates from what configuration this configuration can be reached, and by firing which transition. *Edges to* shows the reachable configurations that can be accessed by firing one transition. The *marking* property shows the precise configuration of each blue node. Ones represent places containing tokens, while zeros indicate empty places. Places are ordered in the order the net was created. So (rather than $(1, 0, 0, 1, 1)$ notation) $\{1, 0, 0, 1, 1\}$ means place P0,P3 and P4 contain a token, while P1 and P2 are empty. Find the blue node representing the initial configuration.

Exercise 2 getting acquainted with pipe2 continued Create the EN system of figure 14 in pipe2. Use the EN validation and conversion module to make sure pipe2 recognise the net as an EN system. Add two places to the EN system, one representing the state neither winter nor spring and one for the state spring or autumn. Use the token game or reachability graph in pipe2 to see if the new places are correct.

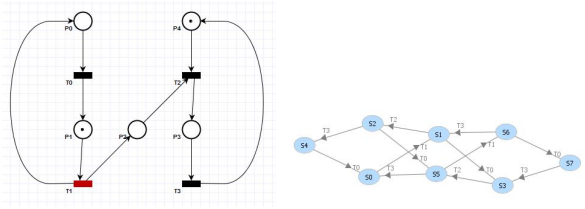

Fig. 13. Token game and sequential configuration graph of producer-consumer net in pipe2

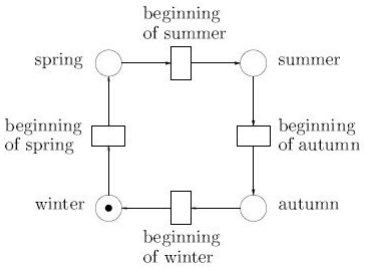

Fig. 14. The change of seasons

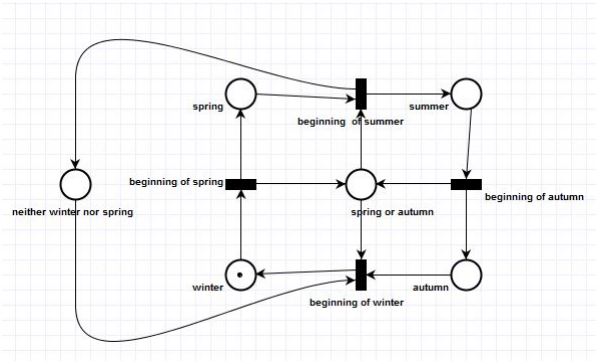

Fig. 15. The change of seasons, implemented in pipe2

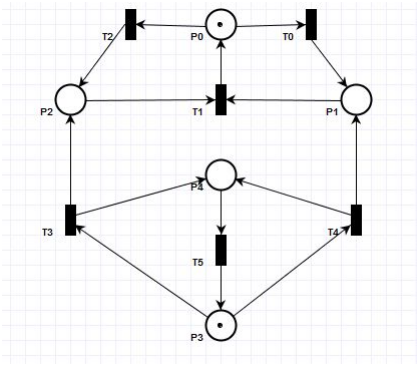

Fig. 16. the net new1.xml

Exercise 3 examined by and analysis of a net using its SCG Open net M stored in file new1.xml, See figure 16
Use the reachability module to create the $SCG(M)$.
Which transitions are useful?
Which transitions are live?
Give a firing sequence s, $s \in FS(M), s \neq \emptyset$ such that $C_{in}[s\rangle C_{in}$.

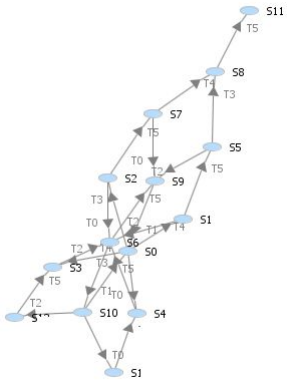

Fig. 17. reachability graph of M from exercise 3

Exercise 4 processes Let M be the contact-free EN system of figure 18. Find a FS σ of M in which every transition occurs exactly once.
Construct a process N for M and σ using the process generator module in pipe2.

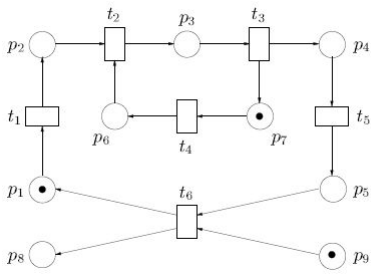

Fig. 18. The contact-free EN system M.

Determine the lines, cuts and slices of N.
Determine all sequential components of N.
Determine **ctr**(N) and **pru**(**ctr**(N)).

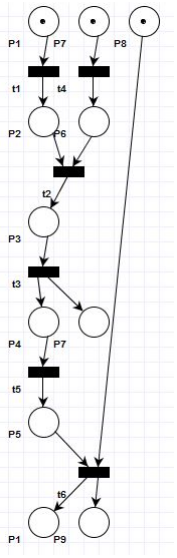

Fig. 19. process generated in pipe2 for exercise 4

Exercise 5 siphons and traps Create the p/t-net as shown in figure 20.
Use the minimal siphons and minimal traps module of pipe2 to find **all** siphons and traps in M.

Exercise 6 usefulness, liveness and confusions Create net M as in figure 21. Find all initial configurations where M is live.

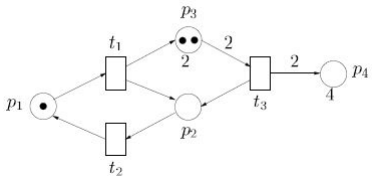

Fig. 20. The p/t-net M

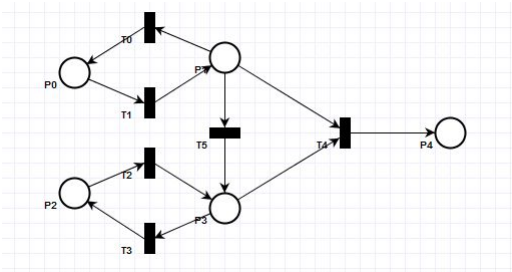

Fig. 21. net M for exercise 6

Find all limited configurations where all transitions in M are useful.
Is there a configuration whereby M is live and all transitions are useful?
Find a configuration that has a conflict increasing confusion.
Find a configuration that has a conflict decreasing confusion. Is this confusion symmetric?

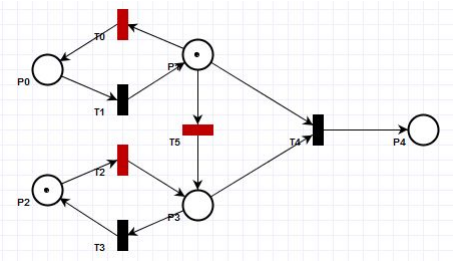

Fig. 22. $(\{P_1, P_3\}, t_2, t_0)$ is a conflict decreasing confusion

exercise 7 processes Create net N such as 23, or load it from disk if available. Use the process generator module to create a number of process nets by giving the module different firing sequences with a length of thirteen transitions. Compare the process nets generated, what do you notice?

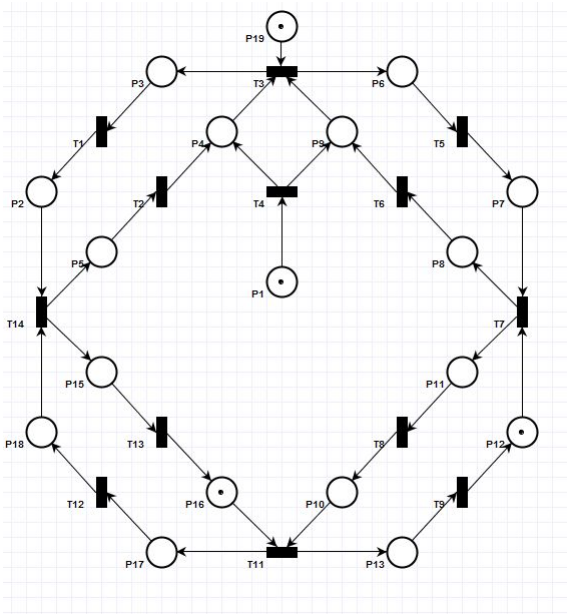

Fig. 23. net N for exercise 7

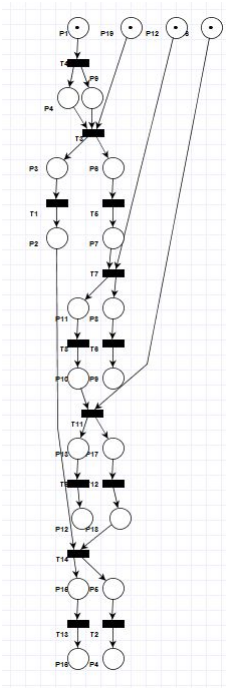

Fig. 24. a process net from N (for exercise 7)

6 Conclusion

The purpose of this project was to find a tool for petri nets that would be useful for education purposes. First, a few basic requirements for such a tool were identified. Next, it was examined what properties could be used to compare the tools selected by using the basic requirements. With the requirements and properties well defined, a search on internet resulted in a list of tools that could be compared. A table was put together using the predefined properties. Based on this table the tool pipe2 was selected.

Pipe2 is a group project at the Imperial College London, and therefore built with educational purposes in mind. Pipe2 supports advanced features in modules, which allows easy adding new modules for specific tasks.

Some new content for the selected tool was created. While creating content for the tool, bugs showed up. Since most programs contain some bugs, this isn't really a surprise. Fortunately these bugs could be fixed.

To validate to what extent pipe2 is extendable, two new modules with functionality not yet included in pipe2 were created:
One, a module for checking if a net is an EN system. With the ability to change this net to an EN system if it is close to the EN systems properties.
And two, A module to construct a process net from a given net and a firing sequence.

We concluded with some references for possibilities to use the tool in an assignment based environment. Unfortunately there was no time to make a real lab assignment for this tool, instead of the now small references to assignments. If a lab assignment would have been designed, it would show the real usefulness of the chosen tool, as well as it capabilities. However, the few small assignments created now show some of the possibilities of the tool.

Another thing that could have been done when there would have been more time was to make more modules. Once familiar with the basic structure of a module in the tool, creation of new tools is not very hard. Modules that could have been implemented or can be implemented in the future are detailed analysis of the configuration graph, automatic creation of complementary places, creation of the independence relation, and confusion detection.

References

1. Jetty Kleijn. slides. url: http://www.liacs.nl/ kleijn/thvc1-0809.html.
2. Joost Engelfriet and Grzegorz Rozenberg. Dictaat tvc. LNCS 1491, 1998.
3. Petri nets tools database quick overview. website. url: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html.
4. Frank Heitmann and Daniel Moldt. petri nets world. website. url: http://www.informatik.uni-hamburg.de/TGI/PetriNets/.
5. Petri nets tool search. website. url: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/search.html.
6. pipe2 homepage. website. url: http://pipe2.sourceforge.net/.
7. petrilld homepage. website. url: http://sourceforge.net/apps/mediawiki/petrilld/index.php?title=Main_Page.

8. Cpntools homepage. website. url: http://wiki.daimi.au.dk/cpntools/cpntools.wiki.
9. Woped homepage. website. url: http://www.woped.org/.
10. Netlab homepage. website. url: http://www.irt.rwth-aachen.de/index.php?id=101.
11. Small java tools homepage. website. url: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/java/.
12. Design/cpn online. website. url: http://www.daimi.au.dk/designCPN.

7 Appendix A: Tool information

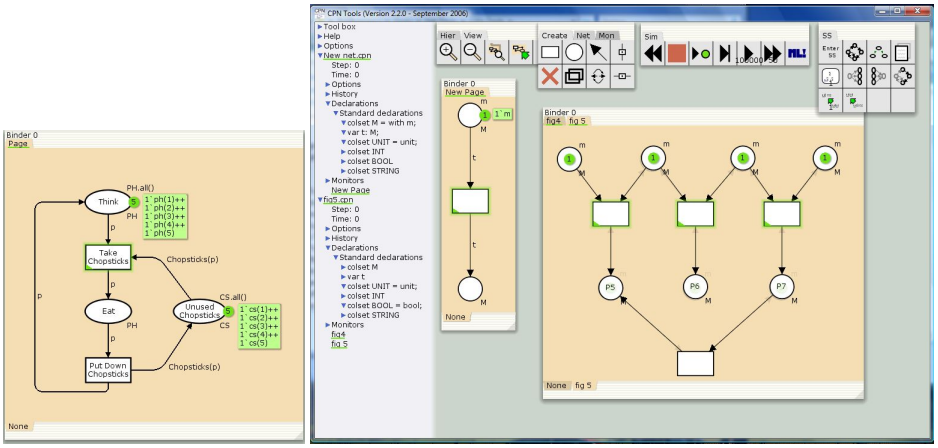

Fig. 25. Screenshots van cpntools

7.1 CPN tools

general

origin This tool has been created and is maintained by the CPN Group, University of Aarhus, Denmark. CPN tools is the successor of the tool Design/CPN.

This tool was originally created by the Meta Software Corporation, Cambridge MA, USA in cooperation with the CPN group at Aarhus University. From 1989 till 1995 the tool was sold to companies. In 1996 the focus of the Meta Software Corporation changed (to work flow analysis), and Design/CPN was no longer of their interest. As a result, the CPN Group at Aarhus took over the development and distribution of Design/CPN, which later was followed by CPN tools.

homepage http://wiki.daimi.au.dk/cpntools/cpntools.wiki

platform CPN tools will run on Windows 2000, Windows XP, and Linux Fedora Core 2. Older Windows versions are not supported. System recommendations: Pentium II @ 400Mhz, 256 MB RAM, 50 MB HD, Must have OpenGL hardware acceleration. CPN tools is a heavy program to run, a good pc is recommended.

net support Colored nets, timed nets. Timed nets are defined as follows: tokens could contain a time stamp. Such a token can only be used to fire a transition when the time stamp is lower or equal than the global system time. When a token with a time stamp is used by a firing transition with delay property, the tokens time stamp is updated according to this delay property. When all transitions are not enabled, the system clock is increased till at least one transition can be fired.

distribution To be able to install and use CPN tools a licence is required. Licences are given to non-profit organisations for free, but this is an extra step before you can install and use the tool. This tool is not open source, Only the executable program is accessible.

documentation CPN tools is not a tool that let you get immediately familiarised with. Therefore a good documentation is a very important help. These documentation comes with the installation files of the tool, but is also available to view online. The tool is well documented.

functional properties

interface CPN tools has a sort of revolutionary interface and control use. When the tool has been started, the program exists of two parts; on the left a list with items that are grouped, and on the right a window where the actual stuff will happen. To get control buttons you have to drag the required dock from the list in the left to the window. These docks can be changed in size, you can merge multiple docks into a single one by dragging them into each other. The right click mouse menu is a pie menu, and the tool even recognises certain mouse gestures. The tool gives you a lot of freedom in its interface, and the same holds for the layout. When a net is loaded, it is visible in a small window. A new net can be either created in a new tab or a new window. Certain parts of the net can be grouped together if you like, and will get a special tab just for this group. The size of places and transitions is modifiable, as is the location of variables of any object in a petri net. It also has a few features to align places and transitions on one line, and to color the outlines of a place or transition, to get a nice layout.

features High level petri net hierarchy, token game, configuration graph, performance analysis.

7.2 WoPed: Workflow Petri Net Designer

general

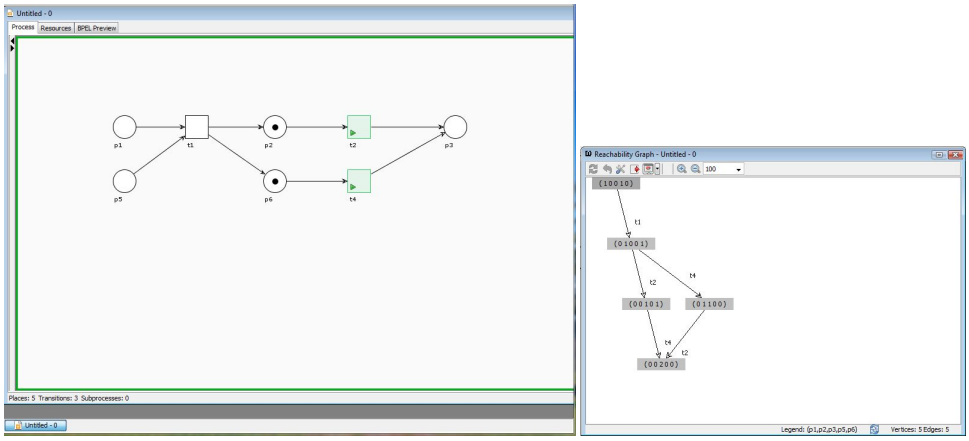

Fig. 26. screenshots WoPeD

origin A tool created by Duale Hochschule Baden-Württemberg Karlsruhe. This tool is designed for work flow process modeling and analysing. The tool is in development since 2003.

homepage http://www.woped.org/

platform WoPed can run on Windows, Linux and MacOS, using the java run time environment. All installation files are in wizards, so easy to install.

net support p/t-nets, workflow nets

distribution This tool is distributed open source under the GNU Lesser General Public License (LGPL).

documentation WoPed comes with some html pages for install instructions and basic use. It also comes with pdf files with a detailed description of the advanced abilities of WoPed.

functional properties

interface WoPed starts with a basic window with a toolbar within it. A new net will have a small window within this window, and will have tabs for resources and a special view. A net can be created by selecting elements (places, transitions) from the toolbar and clicking in the smaller window. Arcs are created by dragging a place to a transition or reversed. If a place or transition is dragged to an empty area, WoPeD will automatically create a transition or place here. This results in an easy and fast way of constructing a net.

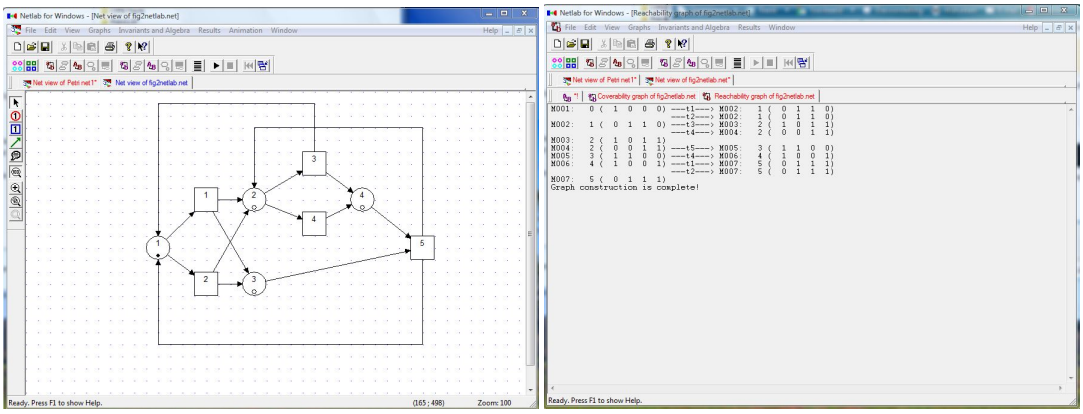

Fig. 27. screenshots netlab

features Clear state space, analysis of nets, capacity planning, information border.

7.3 Netlab

general

origin Netlab is a tool originally from the RWTH Aachen University. The first versions were developed in Fortran from 1989 till 1993. Recent versions are developed in C++.

homepage http://www.irt.rwth-aachen.de/en/downloads/petri-net-tool-netlab.html

platform NetLab runs on Windows, versions are not specified.

net support p/t-nets

distribution Free to use, included with mathlab source code.

documentation Included documentation is in German.

functional properties

interface Window with toolbar on top.

features State-space graph, generated in plain text. Reachability graph, also text rendered. token game

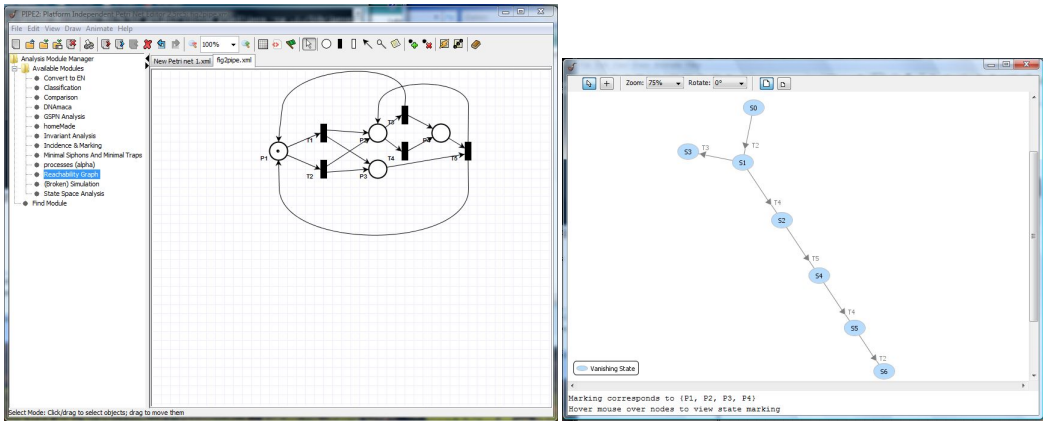

Fig. 28. screenshots pipe2

7.4 Pipe2: Platform Independent Petri Net Editor 2 specifications (V2.5)

general

origin Pipe2 started as a group project within the Department of Computing, Imperial College London in 2002. It is still maintained as a project at the college.

homepage http://pipe2.sourceforge.net/

platform Pipe2 runs on the java platform. This platform is available for most popular operation systems.

net support p/t-nets and timed nets. In pipe2, timed transitions only enabled when there are no non-timed transitions enabled. Tokens have no time stamps, unlike CPN tools.

distribution Both pipe2 and its source code are freely available to download.

documentation With the installation files of pipe2 comes a web page documentation. This documentation is good enough to get started, however it lacks details of provided modules.

functional properties

interface The program interface exists of a large window, with on top a menu with file, edit, etc. Below this menu is an interface bar with buttons for most commonly used commands, like create new net, delete net, create new place, create new transition, play token game, etc. Under the interface bar is on the

left part a list with available modules. Modules are small programs that can be used to analyse and/or modify a net. On the right side is a window which contains a visual representation of a net. In case there is more than one net used at the same time, then one will be visible and the others can be selected using the tabs above the window.

general features Pipe2 has a token game, supports drag and drop, allows you to save and load a net, allows to set capacity for places, zooming in on nets.

advanced features: modules Pipe2 comes with a number of default modules. Modules can also be written and added manually. This allows users to add functionality without modifying the original program. In general all modules use the currently opened net, but instead the user may also load a net from a file to apply the modules to. The modules that are included with pipe2 are:

classification This module analyses the selected net on a few properties and returns a display with a table containing the results. The properties are:

- State Machine
  returns true if all transitions have at most one input or output.
- Marked Graph
  returns true if all places have at most one input or output.
- Free choice net
  returns true if there are no places with shared output transitions, unless those places have only one output.
- Extended Free Choice net
  returns true if there are no places with shared input transitions unless these places outputs are identical.
- Simple net
  returns true if no two places output to the same transition, unless one of the places only has one output.
- Extended simple net
  returns true if no two places output to the same transition, unless one output set is a subset of the other.

comparison This module compares properties of nets. Like name, ID (unique name that pipe2 uses to identify the different places), marking, capacity, rate, priority, weight. The items that are compared should have the same name, same ID or in case of arcs the same connection from place to transition or reversed. For example, when both nets contain a place with ID 'P0', this module will compare the capacity, marking, ID and name of the two places.

DNAmaca This module require a program called urta, which needs to be installed separately. The program returns a graph, based on a timed petri net. Unfortunately, it is very unclear where and how to get the urta program, and what sort of graph DNAmeca returns when urta is installed as well.

GSPN Analysis To analyse timed petri nets. Returns a few tables with information. Returned tables are;

- Set of Tangible States
  (Tangible: all enabled transitions are timed)
- Steady State Distribution of Tangible States
- Average Number of Tokens on a Place
- Token Probability Density
- Throughput of Timed Transitions
- Sojourn times for tangible states

invariant Analysis This module calculates the T and P invariants of a net. It also can determine whether a net is bounded or live on basis of the discovered invariants.

incidence & marking This module gives the forwards incidence matrix, backward incidence matrix, the combined incidence matrix, the inhibition matrix, the matrix with the current marking and a table with all transitions and whether they are enabled or not at the current configuration.

minimal siphons and minimal traps This module shows the minimal siphons and minimal traps.

reachability graph This module generates a configuration graph from the source net. The graph is presented by nodes connected with arrows. It is possible to zoom in and out, to move the graph a little bit and the graph can be rotated. When the mouse is hovered over a node, a pop up give more detailed information about the configuration this node represent. If the configuration graph can't be generated because it is infinite, the module will automatically generate the coverability graph instead.

(broken) simulation As the name suggest, this module might not work correctly in the current version. This module uses the selected net and a number indicating how many firings must be simulated to calculate the average number of tokens contained by all places during the simulation.

state space analysis Determines if a net is bounded, safe, deadlock fire. If the net has a deadlock it will also return the shortest firing sequence to get to a deadlock.

7.5 Yasper: yet another smart process editor

general

origin Yasper is a tool created by the TU Eindhoven and Deloitte. It is designed for workflow modeling.

homepage http://www.yasper.org/

platform Yasper runs on windows .NET framework.

net support p/t-nets and (limited) colored nets.

distribution Not open source, download msi file to install the tool.

documentation Nice pdf with basic explanation and use of the tool.

installation Yasper requires .Net framework version 1.1.4322, and will prompt you to install this framework. However the current version of .Net is 3.5, and this version will not stop Yasper from prompting the .net install question. Ignoring this lead to breaking off the installation of Yasper. Since it is not likely that you would want to search and install a very old .Net version for your pc, we have to conclude that Yasper is not install friendly.

functional properties

interface Based on only the screenshots, Yasper has a very simple but clear interface, and generated nets are well organised.

advanced features Since Yaspers focus is on simply designing and simulating petri nets, it does not have advanced analyse tools.

7.6 PetriLLD

general

origin Original created by James Brusey and improved by Cambridge University Manufacturing Engineering Tripos students. The focus was on ladder-logic diagrams which are diagrams in a graphical language implemented by boolean statements, which is similar to EN systems.

homepage http://sourceforge.net/apps/mediawiki/petrilld/index.php?title=Main_Page

platform Platform independent, using java

net support Elementary nets.

distribution Free to use under the GNU General Public License, version 2 or higher.

documentation Documentation for PetriLLD is available in pdf and html format from the PetriLLD web site.

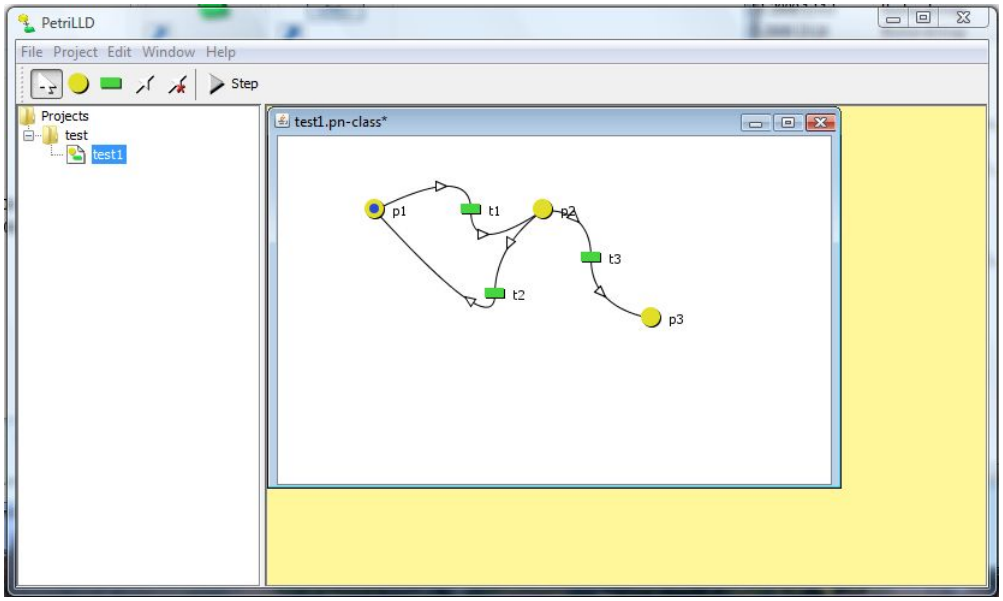

Fig. 29. Screenshot of PetriLLD

functional properties

interface Simple interface with buttons for places and transitions, and a small window for currently opened files. When creating a net, an arc from transition to place or reversed will automatically been curved, it is possible to turn this feature off.

features The main use of this tool is to convert a net to some file format. It has, except for a token game, no other useful functionality.

simple java applets

general

origin every java applet is designed by a different author. The collection of tools is hosted by Petri Nets World.

homepage http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/java/

platform Java, without installation, direct useable from a web browser with java support.

net support Elementary nets and p/t-nets

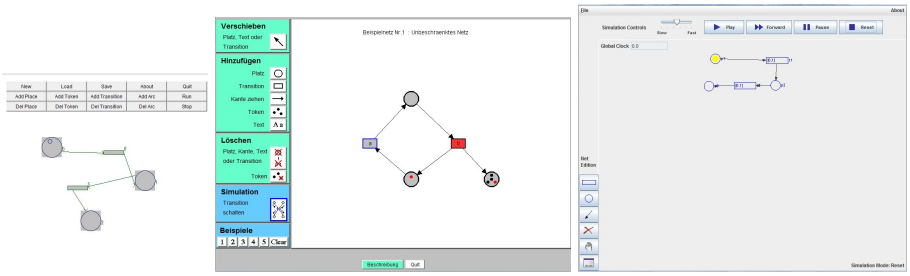

Fig. 30. screenshots from some simple java applications

distribution The applets are free to use from the web page. However no source code available.

documentation The applets come with no documentation. They are however simple and intuitive to use.

functional properties

interface They all have different interfaces, but generally they have some buttons for creation and modification of a net and one white screen area where a net can be designed.

features Although all applets are different, all support the token game.

8 Appendix B: source code

8.1 new modules

EN validation module This module is based on the standard pipe2 module layout. It verifies whether if a net is an EN system, possibly with places with a capacity higher than one. The validation tool will set the capacity of all places to one once the net has been correctly validated.

```
/**
 * EN system validate module
 * @author Wouter de Zwijger
 *
 * This module validates if a net is an EN system
 * it can also convert a p/t net into an elementary net system,
 * if it has all EN system properties except required place capacities.
 * Properties of the defined elementary net system compared to p/t nets:
 * - no self loops
 * - all places have a capacity of one
 * - all places have at initial state either one or zero tokens
 * - all transitions have at least one input and output place
 * - no timed transitions, inhibitor arcs, places with priority > 1
 */

package pipe.modules.CapToOne;
```

```
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Arrays;

import javax.swing.BoxLayout;
import javax.swing.JOptionPane;

import pipe.dataLayer.DataLayer;
import pipe.gui.CreateGui;
import pipe.gui.widgets.ButtonBar;
import pipe.gui.widgets.EscapableDialog;
import pipe.gui.widgets.PetriNetChooserPanel;
import pipe.gui.widgets.ResultsHTMLPane;
import pipe.modules.EmptyNetException;
import pipe.modules.Module;


/** Convert module implements a module */
public class CapToOne
        implements Module {

   private static final String MODULE_NAME = "EN validator and converter";

   //private PetriNetChooserPanel sourceFilePanel;
   private DataLayer sourceDataLayer;
   private ResultsHTMLPane results;
   private final static String init =
                     "<h2>P/T to elementary net system conversion</h2>"
                   +"Checks for a few properties to validate it can be converted to an EN system<br>"
                   +"If valid, all places will get capacity of one<br>"
                   +"<br>"
                   +"Properties that input net must hold: "
                   +"<ul> "
                   +"<li>All transitions have at least one input and one output place </li>"
                   +"<li> Transitions are immediate"
                   +"<li> All weights on the arcs are 1"
                   +"<li> None of the place/transition pairs contain a self loop </li>"
                   +"<li> All places do not contain more than one token </li>"
                   +"</ul>";

   public void run(DataLayer pnmlData) {
      // Build interface
      EscapableDialog guiDialog =
               new EscapableDialog(CreateGui.getApp(), MODULE_NAME, true);

      // 1 Set layout
      Container contentPane = guiDialog.getContentPane();
      contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.PAGE_AXIS));

      // 2 Add file browser
      //sourceFilePanel = new PetriNetChooserPanel("Source net",pnmlData);
      //contentPane.add(sourceFilePanel);

      // 3 Add results pane
      results = new ResultsHTMLPane(pnmlData.getURI());
      contentPane.add(results);

      // 4 Add button
      contentPane.add(new ButtonBar("Convert",classifyButtonClick,
               guiDialog.getRootPane()));

      // 5 Make window fit contents' preferred size
      guiDialog.pack();

      // 6 Move window to the middle of the screen
      guiDialog.setLocationRelativeTo(null);

      // 7 set initial text
      results.setText(init);
      sourceDataLayer = pnmlData;

      guiDialog.setVisible(true);
   }


   public String getName() {
      return MODULE_NAME;
   }


   /**
    * Action listener
    */
   ActionListener classifyButtonClick=new ActionListener() {

      public void actionPerformed(ActionEvent arg0) {

         if (sourceDataLayer == null) {
```

```
        JOptionPane.showMessageDialog( null, "Please, choose a source net",
                "Error", JOptionPane.ERROR_MESSAGE);

        return;
      }

      String s = "<h2>Petri net conversion complete</h2>";

      if (!sourceDataLayer.hasPlaceTransitionObjects()) {
        s += "Given Petri net is empty, thus already a EN";
      } else {
       if(checkNoCycles()&&checkImm()&&checkTokens()&&checkWeight()&&checkCompleteTrans()&&checkNoPriority()&&checkInhibitorArcs()){
        makeAllStatesLimitOne();
        s += "All states have now Capacity 1";
        s += "<br> This net is now an elementair net";
        s += "<br> and hold all its properties";
       } else {
        String failureText = "";
        if(!checkNoCycles())
         failureText += "Net contains place/transition loops!<br>";
        if(!checkImm())
         failureText += "Net contains timed transitions!<br>";
        if(!checkTokens())
         failureText += "Net contains places with more than one token!<br>";
        if(!checkCompleteTrans())
         failureText += "Net contains transitions without input or output place(s)!<br>";
        if(!checkWeight())
         failureText += "Net contains arcs with weights higher than one!<br>";
        if(!checkNoPriority())
         failureText += "Net contains transitions with priorities higher than one!<br>";
        if(!checkInhibitorArcs())
          failureText += "Net contains inhibitor arcs!<br>";

        s = "<h2 color ='#FF0000' >Petri net conversion ERROR!<h2>" + "<h3> ERROR message(s):</h3>"+ failureText;
       } // else
        results.setEnabled(true);
      }
      results.setText(s);

   }
};

// convert all places to places with cap 1
private void makeAllStatesLimitOne(){
  int placeCount = sourceDataLayer.getPlacesCount();
  for (int placeNo = 0; placeNo < placeCount; placeNo++)
    sourceDataLayer.getPlace(placeNo).setCapacity(1);
} // makeAllStatesLimitOne

// detects place/transition loops
private boolean checkNoCycles(){
  int MatrixPlaceToTrans[][] = sourceDataLayer.getBackwardsIncidenceMatrix();
  int MatrixTransToPlace[][] = sourceDataLayer.getForwardsIncidenceMatrix();
  // both matrixes:
  // row number represent place
  // column number represent transition
  // diference:
  // Backwards representing arcs from places to transitions
  // Forwards representing arcs from transitions to places

  // this function checks for place transition loops
  // if there exist such a loop, then both matrices contain a value higher than zero
  // at the same position, since there is a arc from and too a place to the same transition
  for(int y = 0; y < MatrixPlaceToTrans.length; y++)
    for(int x = 0; x < MatrixPlaceToTrans[y].length; x++)
      if(MatrixPlaceToTrans[y][x] > 0 &&  MatrixTransToPlace[y][x] > 0)
       return false;
  return true;
} // checkNoCycles

// checks if net does not contain timed loops
private boolean checkImm(){
  int transitionCount = sourceDataLayer.getTransitionsCount();
  for (int transitionNo = 0; transitionNo < transitionCount; transitionNo++)
    if(sourceDataLayer.getTransition(transitionNo).isTimed())
        return false;
  return true;
} // checkImm

// checks if all palces do not contain more than one tooken
private boolean checkTokens(){
  int placeCount = sourceDataLayer.getPlacesCount();
  for (int placeNo = 0; placeNo < placeCount; placeNo++)
    if(sourceDataLayer.getPlace(placeNo).getCurrentMarking() > 1
       || sourceDataLayer.getPlace(placeNo).getInitialMarking() > 1)
        return false;
  return true;
} // checkTokens

// checks if all arcs conatian a weight of one
```

```
   private boolean checkWeight(){
     int MatrixPlaceToTrans[][] = sourceDataLayer.getBackwardsIncidenceMatrix();
     int MatrixTransToPlace[][] = sourceDataLayer.getForwardsIncidenceMatrix();
     // all values in the matrix should either be 1 or 0, a other vaule
     // indecate a arc with a weight that is not 1
     for(int y = 0; y < MatrixPlaceToTrans.length; y++)
       for(int x = 0; x < MatrixPlaceToTrans[y].length; x++)
         if(MatrixPlaceToTrans[y][x] > 1 ||  MatrixTransToPlace[y][x] >1)
          return false;
     return true;
   } // checkWeight

   // check if al transitions have at least one input and output place
   private boolean checkCompleteTrans(){
     int MatrixPlaceToTrans[][] = sourceDataLayer.getBackwardsIncidenceMatrix();
     int MatrixTransToPlace[][] = sourceDataLayer.getForwardsIncidenceMatrix();
     boolean hasInput = false;
     boolean hasOutput = false;
     // all values in the matrix should either be 1 or 0, a other vaule
     // indecate a arc with a weight that is not 1
     for(int x = 0; x < MatrixPlaceToTrans[0].length; x++){
       for(int y = 0; y < MatrixPlaceToTrans.length; y++){
         if(MatrixPlaceToTrans[y][x] > 0)
          hasInput = true;
         if(MatrixTransToPlace[y][x] >0)
          hasOutput = true;
       } //for
       if(!hasOutput || !hasInput)
         return false;
       hasInput = false;
       hasOutput = false;
     } // for
     return true;
   } // checkCompleateTrans

   // checks if the net does not contain transitions with priority higher than 1
   private boolean checkNoPriority(){
     int[] pVec = sourceDataLayer.getPriorityVector();

     for(int x =0;x < pVec.length; x++)
        if(pVec[x] > 1)
          return false;

     return true;
   }// checkNoPriority

   // checks if there are no inhibitor arcs
   private boolean checkInhibitorArcs(){
     int[][] inhMat = sourceDataLayer.getInhibitionMatrix();
     for(int x =0;x<inhMat.length;x++)
       for(int y = 0;y<inhMat[x].length;y++)
         if(inhMat[x][y] > 0)
           return false;

     return true;
   }// checkInhibitorArcs


} // end module
```

processes construct module This module requires a net that is validated as a EN system with the EN validation module. It also requires a firing sequence, given in an input field. This module will, if the net is contact free, construct a new net that is a process of the original net.

```
/**
 * process generating module
 * @author Wouter de Zwijger
 *
 * This module has as input a firing sequence and an elementary net system which must be contact free,
 * Result: a process based on the firing sequence
 */

package pipe.modules.processes;

import java.awt.Container;
//import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Arrays;
```

```
import javax.swing.BoxLayout;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import pipe.dataLayer.DataLayer;
  import pipe.dataLayer.Transition;
  import pipe.dataLayer.Place;
  import pipe.dataLayer.Arc;
  import pipe.dataLayer.NormalArc;
  import pipe.dataLayer.PetriNetObject;
  import pipe.dataLayer.PlaceTransitionObject;

import pipe.dataLayer.calculations.TreeTooBigException;
import pipe.dataLayer.calculations.myTree;

import pipe.gui.CreateGui;
import pipe.gui.widgets.ButtonBar;
import pipe.gui.widgets.EscapableDialog;
import pipe.gui.widgets.PetriNetChooserPanel;
import pipe.gui.widgets.ResultsHTMLPane;
import pipe.modules.EmptyNetException;
import pipe.modules.Module;

import pipe.gui.CreateGui;
import pipe.gui.GuiFrame;
import pipe.gui.Pipe;



/** Processes class implements petri net classification */
public class processes
        implements Module {

   private static final String MODULE_NAME = "process generator";

   private PetriNetChooserPanel sourceFilePanel;
   private ResultsHTMLPane results;

   private JTextField inputText;
   private JPanel inputPanel;

   private DataLayer destinationDataLayer = null;

   private int uniquePlaceId = 0;
   private int uniqueTransitionId = 0;


   public void run(DataLayer pnmlData) {

      // Build interface
      EscapableDialog guiDialog =
              new EscapableDialog(CreateGui.getApp(), MODULE_NAME, true);

      // 1 Set layout
      Container contentPane = guiDialog.getContentPane();
      contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.PAGE_AXIS));

      // 2 Add file browser
      sourceFilePanel = new PetriNetChooserPanel("Source net",pnmlData);
      contentPane.add(sourceFilePanel);

      // 2b add option area
      inputPanel = new JPanel();
      inputPanel.setLayout(new BoxLayout(inputPanel,BoxLayout.PAGE_AXIS));
      inputPanel.setBorder(new TitledBorder(new EtchedBorder(),"Transition fire order"));
      inputPanel.add(new JLabel("example: t1;t2;t1;t3"));
      inputPanel.add(inputText = new JTextField(5));

      //inputPanel.setMaximumSize(new Dimension(Integer.MAX_VALUE,inputPanel.getPreferredSize().height));
      contentPane.add(inputPanel);

      // 3 Add results pane
      results = new ResultsHTMLPane(pnmlData.getURI());
      contentPane.add(results);

      // 4 Add button
      contentPane.add(new ButtonBar("Build process",buildButtonClick,
              guiDialog.getRootPane()));

      // 5 Make window fit contents' preferred size
      guiDialog.pack();


      // 6 Move window to the middle of the screen
      guiDialog.setLocationRelativeTo(null);
```

```
      guiDialog.setVisible(true);
   }


   public String getName() {
      return MODULE_NAME;
   }


   /**
    * Build process button click handler
    */
   ActionListener buildButtonClick=new ActionListener() {

      public void actionPerformed(ActionEvent arg0) {
         DataLayer sourceDataLayer = sourceFilePanel.getDataLayer();

         if (sourceDataLayer == null) {
            JOptionPane.showMessageDialog( null, "Please, choose a source net",
                    "Error", JOptionPane.ERROR_MESSAGE);
            return;
         }

         String s = "<h2>Petri net Proces generator result:</h2>";

         //new Place(100,100).showEditor(); //!!!! useful dialog popup test

         if (!sourceDataLayer.hasPlaceTransitionObjects()) {
            s += "process result: Empty net";
         } else {
            s += generateProcess(sourceDataLayer);
            //s += "job done";
            results.setEnabled(true);
         }
         results.setText(s);
      }
   };



   // convert all places to places with cap 1
   private void makeAllStatesLimitOne(DataLayer sourceDataLayer){
        int placeCount = sourceDataLayer.getPlacesCount();

      for (int placeNo = 0; placeNo < placeCount; placeNo++) {
       sourceDataLayer.getPlace(placeNo).setCapacity(1);
      } // for
   } // makeAllStatesLimitOne

   // convert all places to places with no cap
   private void makeAllStatesLimitZero(DataLayer sourceDataLayer){
        int placeCount = sourceDataLayer.getPlacesCount();

      for (int placeNo = 0; placeNo < placeCount; placeNo++) {
       sourceDataLayer.getPlace(placeNo).setCapacity(0);
      } // for
   } // makeAllStatesLimitOne


   // generate a process based on given net and firing sequense
   private String generateProcess(DataLayer sourceDataLayer){
        String textOutput = "";
String transitionName[] = inputText.getText().split(";");

  boolean noError = true;

    // check all transitions availeble?
    // note that it does not check for multiple transitions with the same name!
  for(int tmp=0;tmp<transitionName.length;tmp++){
  if(sourceDataLayer.getTransitionByName(transitionName[tmp])== null){
      noError = false;
  textOutput += "<br>transition " + transitionName[tmp] + " not availeble!";
  return textOutput;
  } // if
  } // for

  // this function is restricted to Elementair Nets
  // first check if all places have capacity one
  for (int placeNo = 0; placeNo < sourceDataLayer.getPlacesCount();placeNo++) {
   if(sourceDataLayer.getPlace(placeNo).getCapacity()!=1){
       textOutput += "<br>Error: " + sourceDataLayer.getPlace(placeNo).getName() + " does not have capicity 1!";
   noError = false;
   return textOutput;
   } // if
    } // for

    // secondly check on the other conditions for a elementary net
    if(!checkNoCycles(sourceDataLayer)||!checkImm(sourceDataLayer)||!checkTokens(sourceDataLayer)||!checkWeight(sourceDataLayer)||!checkCompleteTrans(sourceDataLayer)){
    textOutput += "<br>Error: net is not a EN!<br> please use validator module to find out why";
```

```
noError = false;
return textOutput;
    } // if

    if(!checkSaveNet(sourceDataLayer)){
     textOutput += "<br>Error: net is not contact-free!";
     noError = false;
     return textOutput;
    } // if


     /*startCreating();
         Place a = createPlace("P0",20,20);
         Transition b = createTransition("T0",40,40);
         Arc c = createArc(a,b);
         Place d = createPlace("P0",50,50);
         createArc(b,d);
         createArc(a,d);
         stopCreating();*/




  //textOutput += "<br>places before" + transitionName[0] + " :" + getInputPlaces(sourceDataLayer,transitionName[0]) + "<br>";
  //textOutput += "<br>places after" + transitionName[0] + " :" + getOutputPlaces(sourceDataLayer,transitionName[0]) + "<br>";

  String Places[];
  Place tmpPlace;
  Transition tmpTrans;
  int numberOfNodesOnThisRow = 0;
  double lowestInputPlace = 0;
  double totalXfromPlaces = 0;
  double Xoffset;
  startCreating();

  // create initial places with tokens
  for(int tmp = 0;tmp<sourceDataLayer.getPlacesCount();tmp++)
  if(sourceDataLayer.getPlace(tmp).getCurrentMarking()>0)
  createPlace(sourceDataLayer.getPlace(tmp).getName(),50*numberOfNodesOnThisRow++ +20,0).setCurrentMarking(1);

      // main loop: walk trough the firing sequense
  sourceDataLayer.storeState();
  sourceDataLayer.setEnabledTransitions();
  for(int transNumber=0;transNumber<transitionName.length;transNumber++){
  // check if fire sequense is legal
  // if so, construct process
  if(sourceDataLayer.getTransitionByName(transitionName[transNumber]).isEnabled()){

  // create arcs from existing input places to transtition
  // ToDo: correct placement of transition
  //       correct placement of places
  Places = getInputPlaces(sourceDataLayer,transitionName[transNumber]).split(";");
  tmpTrans = createTransition(transitionName[transNumber],20,100*transNumber+50);
  for(int placeNumber=0;placeNumber<Places.length;placeNumber++){
  tmpPlace = destinationDataLayer.getPlaceByName(Places[placeNumber]);
  createArc(
  tmpPlace
      ,destinationDataLayer.getTransitionByName(transitionName[transNumber]));
      // keep track of place locations for decent transition position
      totalXfromPlaces += tmpPlace.getPositionX();
      if(lowestInputPlace<tmpPlace.getPositionY())
      lowestInputPlace = tmpPlace.getPositionY();
  } // for
  //tmpTrans.
  tmpTrans.setPositionX(totalXfromPlaces/Places.length);
   tmpTrans.setPositionY(lowestInputPlace + 50);
   tmpTrans.rotate(90);
   tmpTrans.update();
  //tmpTrans.setCentre(totalXfromPlaces/Places.length,lowestInputPlace + 50);
lowestInputPlace = 0;
  totalXfromPlaces = 0;

      // create arcs from transition to new places
      Xoffset = getXOffset(tmpTrans.getPositionY()+ 50);
                Places = getOutputPlaces(sourceDataLayer,transitionName[transNumber]).split(";");
                for(int placeNumber=0;placeNumber<Places.length;placeNumber++){
                 createArc(
                 tmpTrans,
                 createPlace(Places[placeNumber],50*placeNumber+Xoffset,tmpTrans.getPositionY() + 50));
                 // ToDo: correct place locations
                 //       something for same named places (seems to work automaticly correct...)
                } // for

// fires transition
// to simulate process
  sourceDataLayer.fireTransition(sourceDataLayer.getTransitionByName(transitionName[transNumber]));
  sourceDataLayer.setEnabledTransitions();
  }else{
  // iligal transition fire sequence!
```

```
          noError = false;
       textOutput += "<br>invalid fire sequence by: " + transitionName[transNumber];
  break;
      } // else
  } // for

  sourceDataLayer.resetEnabledTransitions();
  sourceDataLayer.restoreState();
  stopCreating();
  //CreateGui.getApp().repaint();

  // if for some reason the correct process net cant be computed, then the entire
  // process net will be removed
  if(!noError){
  //CreateGui.appGui.setObjectsNull(CreateGui.getTab().getSelectedIndex());
  CreateGui.removeTab(CreateGui.getTab().getSelectedIndex());
  CreateGui.getTab().remove(CreateGui.getTab().getSelectedIndex());
  } // if

    return textOutput;
   } // generateProcess

   // returns number asociated with the given transition object in the datalayer class
   private int getTransitionNumber(DataLayer sourceDataLayer,Transition trans){
    Transition transitionObject[] = sourceDataLayer.getTransitions();
    for(int tmp=0;tmp<transitionObject.length;tmp++)
    if(transitionObject[tmp]==trans)
    return tmp;
        return -1;
   } // getTransitionNumber

   // returns a string with all input place names from given transition
   // syntax: p1;p2;p4;p8;ect
   private String getInputPlaces(DataLayer sourceDataLayer,String transitionName){
    String places = "";
    int transitionNumber = getTransitionNumber(sourceDataLayer,sourceDataLayer.getTransitionByName(transitionName));
    int inputMatrix[][] = sourceDataLayer.getBackwardsIncidenceMatrix();

    for(int tmp=0;tmp<sourceDataLayer.getPlacesCount();tmp++){
    if(inputMatrix[tmp][transitionNumber]>0){
    if(!places.equals(""))
    places += ";";
    places += sourceDataLayer.getPlace(tmp).getName();
    } // if
    } // for

    return places;
   } // getInputPlaces

   // returns a string with all output place names from given transition
   // syntax: p1;p2;p4;p8;ect
   private String getOutputPlaces(DataLayer sourceDataLayer,String transitionName){
    String places = "";
    int transitionNumber = getTransitionNumber(sourceDataLayer,sourceDataLayer.getTransitionByName(transitionName));
    int inputMatrix[][] = sourceDataLayer.getIncidenceMatrix();

    for(int tmp=0;tmp<sourceDataLayer.getPlacesCount();tmp++){
    if(inputMatrix[tmp][transitionNumber]>0){
    if(!places.equals(""))
    places += ";";
    places += sourceDataLayer.getPlace(tmp).getName();
    } // if
    } // for

    return places;
   } // getOutputPlaces


   // creates a new tab in the GuiFrame envirioment
   // initialize destinationDataLayer
   // make sure Gui knows we are creating stuff
   void startCreating(){
CreateGui.appGui.createNewTab(null,false);
destinationDataLayer = CreateGui.getModel();
CreateGui.getApp().setMode(Pipe.CREATING);
   } // startCreating

   void stopCreating(){
    uniquePlaceId = 0;
        uniqueTransitionId = 0;
    CreateGui.getApp().restoreMode();
   } // stopCreating

   // creates a place
   Place createPlace(String name,double cordX,double cordY){
    Place newPlace = new Place(cordX,cordY);
        newPlace.setId("P"+ uniquePlaceId++);
    newPlace.setName(name);
    destinationDataLayer.addPetriNetObject(newPlace);
return newPlace;
```

```
   } // createPlace

   // creates a transition
   Transition createTransition(String name,double cordX,double cordY){
    Transition newTrans = new Transition(cordX,cordY);
newTrans.setId("T"+ uniqueTransitionId++);
    newTrans.setName(name);
    destinationDataLayer.addPetriNetObject(newTrans);
return newTrans;
   } // createPlace

   // creates a arc
   // always weight 1 and a straight line
   Arc createArc(PlaceTransitionObject source, PlaceTransitionObject destination){
    NormalArc newArc = new NormalArc(0,0,100,100,source,destination,1,"",false);
    source.addConnectFrom(newArc);
    destination.addConnectTo(newArc);
    destinationDataLayer.addPetriNetObject(newArc);
    //newArc.updateArcPosition(); //!!!
    return newArc;
   } // Arc

   // returns highest X value from all existing places on a specified Y cord
   private double getXOffset(double Y){
    Place tmpPlace[] = destinationDataLayer.getPlaces();
    double highestX = 0;
    for(int placeNumber=0;placeNumber<tmpPlace.length;placeNumber++)
    if(tmpPlace[placeNumber].getPositionY() == Y
    && highestX<tmpPlace[placeNumber].getPositionX())
    highestX = tmpPlace[placeNumber].getPositionX();
    if(highestX == 0)
    highestX += 20;
    else
    highestX += 50;
    return highestX;
   } // getXoffset

   // check if source net is save as a p/t
   // which means contact-free if it is a elementair net.
   public boolean checkSaveNet(DataLayer sourceDataLayer){
       boolean saveNet;
       makeAllStatesLimitZero(sourceDataLayer);
       try{
       myTree convTree = new myTree(sourceDataLayer,sourceDataLayer.getCurrentMarkingVector());
   // if tree is to big;
   // return true, so that big nets still can be generated,
   // however, chnce on wrong output if big net is not conatct-free
       saveNet = !convTree.moreThanOneToken || convTree.tooBig;
       }catch(Exception e){
           saveNet = true;
       } // catch
       makeAllStatesLimitOne(sourceDataLayer);
       return saveNet;
   } // checkSaveNet

   // ***EN validating functions***

   // detects place/transition loops
   private boolean checkNoCycles(DataLayer sourceDataLayer){
     int MatrixPlaceToTrans[][] = sourceDataLayer.getBackwardsIncidenceMatrix();
     int MatrixTransToPlace[][] = sourceDataLayer.getForwardsIncidenceMatrix();
     // both matrixes:
     // row number represent place
     // column number represent transition
     // diference:
     // Backwards representing arcs from places to transitions
     // Forwards representing arcs from transitions to places

     // this function checks for place transition loops
     // if there exist such a loop, then both matrices contain a value higher than zero
     // at the same position, since there is a arc from and too a place to the same transition
     for(int y = 0; y < MatrixPlaceToTrans.length; y++)
       for(int x = 0; x < MatrixPlaceToTrans[y].length; x++)
         if(MatrixPlaceToTrans[y][x] > 0 &&  MatrixTransToPlace[y][x] > 0)
          return false;
     return true;
   } // checkNoCycles

   // checks if net does not contain timed loops
   private boolean checkImm(DataLayer sourceDataLayer){
     int transitionCount = sourceDataLayer.getTransitionsCount();
     for (int transitionNo = 0; transitionNo < transitionCount; transitionNo++)
       if(sourceDataLayer.getTransition(transitionNo).isTimed())
           return false;
     return true;
   } // checkImm

   // checks if all palces do not contain more than one tooken
   private boolean checkTokens(DataLayer sourceDataLayer){
     int placeCount = sourceDataLayer.getPlacesCount();
```

```
    for (int placeNo = 0; placeNo < placeCount; placeNo++)
      if(sourceDataLayer.getPlace(placeNo).getCurrentMarking() > 1
         || sourceDataLayer.getPlace(placeNo).getInitialMarking() > 1)
          return false;
    return true;
  } // checkTokens

  // checks if all arcs conatian a weight of one
  private boolean checkWeight(DataLayer sourceDataLayer){
    int MatrixPlaceToTrans[][] = sourceDataLayer.getBackwardsIncidenceMatrix();
    int MatrixTransToPlace[][] = sourceDataLayer.getForwardsIncidenceMatrix();
    // all values in the matrix should either be 1 or 0, a other vaule
    // indecate a arc with a weight that is not 1
    for(int y = 0; y < MatrixPlaceToTrans.length; y++)
      for(int x = 0; x < MatrixPlaceToTrans[y].length; x++)
        if(MatrixPlaceToTrans[y][x] > 1 ||  MatrixTransToPlace[y][x] >1)
         return false;
    return true;
  } // checkWeight

  // check if al transitions have at least one input and output place
  private boolean checkCompleteTrans(DataLayer sourceDataLayer){
    int MatrixPlaceToTrans[][] = sourceDataLayer.getBackwardsIncidenceMatrix();
    int MatrixTransToPlace[][] = sourceDataLayer.getForwardsIncidenceMatrix();
    boolean hasInput = false;
    boolean hasOutput = false;
    // all values in the matrix should either be 1 or 0, a other vaule
    // indecate a arc with a weight that is not 1
    for(int x = 0; x < MatrixPlaceToTrans[0].length; x++){
      for(int y = 0; y < MatrixPlaceToTrans.length; y++){
        if(MatrixPlaceToTrans[y][x] > 0)
         hasInput = true;
        if(MatrixTransToPlace[y][x] >0)
         hasOutput = true;
      } //for
      if(!hasOutput || !hasInput)
        return false;
      hasInput = false;
      hasOutput = false;
    } // for
    return true;
  } // checkCompleateTrans

  /*// used for testing
  void testPlaceTransArc(){
   // creates output net in a new tap
        GuiFrame frame = CreateGui.appGui;
        frame.createNewTab(null,false);
        destinationDataLayer = CreateGui.getModel();
        Place testPlace = new Place(20.0,20.0);
        Transition testTrans = new Transition(50,50);
        NormalArc testArc = new NormalArc(0,0,100,100,testPlace,testTrans,1,"",false);
        testPlace.addConnectFrom(testArc);
        testTrans.addConnectTo(testArc);



        //edit mode;
        CreateGui.getApp().setMode(Pipe.CREATING);
        destinationDataLayer.addPetriNetObject(testPlace);
        destinationDataLayer.addPetriNetObject(testTrans);
        destinationDataLayer.addPetriNetObject(testArc);
        //sourceDataLayer.addPetriNetObject(testPlace);
        CreateGui.getApp().restoreMode();
        return null;
  }*/
}
```

8.2 bug fixes

siphon module This module is included with pipe2. However it has a small flaw. When a net has one or more isolated places, the module will give an incorrect result. This module was slightly changed to fix this bug.
original function:

```
/*
 * MinimalSiphons.java
 */
package pipe.modules.minimalSiphons;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Date;
```

```
import java.util.Vector;

import javax.swing.BoxLayout;

import pipe.dataLayer.DataLayer;
import pipe.dataLayer.PNMatrix;
import pipe.gui.CreateGui;
import pipe.gui.widgets.ButtonBar;
import pipe.gui.widgets.EscapableDialog;
import pipe.gui.widgets.PetriNetChooserPanel;
import pipe.gui.widgets.ResultsHTMLPane;
import pipe.modules.Module;


/**
 * MinimalSiphons computes minimal siphons and minimals traps of a Petri Net.
 * This module implements the algorithm presented in:
 * R. Cordone, L. Ferrarini, L. Piroddi, "Some Results on the Computation of
 * Minimal Siphons in Petri Nets"; Proceedings of the 42nd IEEE Conference on
 * Decision and Control, pp 3754-3759, Maui, Hawaii (USA), December 2003.
 * TODO:
 * a) Try to eliminate extra check for siphons to be mininal, without this
 *    extra check non minimal siphons are computed.
 * b) Optimize!
 * @author Pere Bonet
 */
public class MinimalSiphons
        implements Module {


   private DataLayer pnmlData;

   /* */
   private static final String MODULE_NAME = "Minimal Siphons And Minimal Traps";

   private PetriNetChooserPanel sourceFilePanel;
   private ResultsHTMLPane results;


   /**
    * @return        The module name
    */
   public String getName() {
      return MODULE_NAME;
   }


   /**
    * Call the methods that find the minimal traps and siphons.
    * @param dataObj   A dataLayer type object with all the information about
    *                  the petri net
    */
   public void run(DataLayer pnmlData) {
      // Keep a reference to the p-n for other methods in this class
      this.pnmlData = pnmlData;

      // Build interface
      EscapableDialog guiDialog =
               new EscapableDialog(CreateGui.getApp(), MODULE_NAME, true);

      // 1 Set layout
      Container contentPane = guiDialog.getContentPane();
      contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.PAGE_AXIS));

      // 2 Add file browser
      sourceFilePanel = new PetriNetChooserPanel("Source net",pnmlData);
      contentPane.add(sourceFilePanel);

      // 3 Add results pane
      results = new ResultsHTMLPane(pnmlData.getURI());
      contentPane.add(results);

      // 4 Add button
      contentPane.add(new ButtonBar("Generate",analyseButtonClick,
               guiDialog.getRootPane()));

      // 5 Make window fit contents' preferred size
      guiDialog.pack();

      // 6 Move window to the middle of the screen
      guiDialog.setLocationRelativeTo(null);

      guiDialog.setVisible(true);
   }


   /**
    * Generate button click handler
    */
   ActionListener analyseButtonClick = new ActionListener() {
```

```
   public void actionPerformed(ActionEvent arg0) {
      DataLayer sourceDataLayer=sourceFilePanel.getDataLayer();
      String s = "<h2>Minimal Siphons and Minimal Traps</h2>";

      if (sourceDataLayer == null) {
         return;
      }

      if (!sourceDataLayer.hasPlaceTransitionObjects()) {
         s += "No Petri net objects defined!";
      } else {
         s += analyse(sourceDataLayer);
         results.setEnabled(true);
      }
      results.setText(s);
   }
};


/**
 *
 */
private String analyse(DataLayer pnmlData) {

   Date start_time = new Date(); // start timer for program execution

   String output = "<h3>Minimal siphons</h3>";
   // compute siphons
   Vector <boolean[]> siphons = findAllMinimalSiphons(
           new PetriNet(pnmlData.getForwardsIncidenceMatrix(),
                        pnmlData.getBackwardsIncidenceMatrix()),
           new SetOfPlaces(pnmlData.getPlacesCount()));
   output += toString(siphons);

   output += "<h3>Minimal traps</h3>";
   // now, compute traps switching forwards and backwards incidence matrices
   Vector <boolean[]> traps = findAllMinimalSiphons(
           new PetriNet(pnmlData.getBackwardsIncidenceMatrix(),
                        pnmlData.getForwardsIncidenceMatrix()),
           new SetOfPlaces(pnmlData.getPlacesCount()));
   output += toString(traps);

   Date stop_time = new Date();
   double etime = (stop_time.getTime() - start_time.getTime())/1000.;
   return output + "<br>Analysis time: " + etime + "s";
}


private String toString(Vector <boolean[]> vector) {
   String s = "";

   if (vector.size() == 0) {
      return "None found.<br>";
   }

   for (boolean[] element : vector) {
      s += "{";
      for (int i = 0; i < element.length; i++) {
         s += element[i] ? pnmlData.getPlace(i).getName() + ", "
                         : "";
      }
      // replace the last occurance of ", "
      s = s.substring(0, (s.length()-2)) + "}<br>";
   }
   return s;
}


/**
 * findAllMinimalSiphons()
 * Finds all minimal siphons in a given Petri Net that contain a specific
 * set of places.
 * @param G      A Petri Net
 * @param Ptilde A set of places that each siphon must contain
 * returns       A vector containg all minimal siphons found
 */
private Vector <boolean[]> findAllMinimalSiphons(PetriNet G, SetOfPlaces Ptilde){
   Vector <boolean[]> E; // contains all minimal siphons found
   SetOfPlaces S = new SetOfPlaces(Ptilde.size()); // a siphon

   // Step 1
   E = new Vector();

   // Step 2
   // check if there is a place with an empty pre-set
   int p = G.placeWithEmptyInputSet();
   while (Ptilde.isEmpty() && (p != -1)){

      // Step 3
```

```
            S.add(p);
            E.add(S.getPlaces());
            // the Petri Net can be reduced by eliminating p
            G = reduceNet(G, Ptilde.getPlacesMinus(p));
            // check if there is another place with an empty pre-set
            p = G.placeWithEmptyInputSet();
         }

         // Step 4
         // Find a generic siphon
         SetOfPlaces Stilde = findSiphon(G, Ptilde);

         // Step 5
         if (Stilde.isEmpty()) {
            // No siphon has been found, return E
            return E;
         }

         // Step 6
         // Find a minimal siphon contained in the generic siphon
         S = findMinimalSiphon(G, Stilde, Ptilde);

         // problem!
         // here should be simply E.add(S.getPlaces());
         // but without this extra check, non minimal siphons were computed.
         // So, a we find a new minimal siphon (S2) contained in the generic siphon
         // with no place constraints and we check if S contains S2
         SetOfPlaces S2 = new SetOfPlaces(Ptilde.size());
         S2 = findMinimalSiphon(G, Stilde, new SetOfPlaces(Ptilde.size()));
         if (!S.containsSet(S2)) {
            // S is minimal!
            E.add(S.getPlaces());
         }

         // Step 7
         SetOfPlaces Pnew = S.minus(Ptilde); //Pnew = S - P~
         SetOfPlaces Pold = new SetOfPlaces(S.size()); // Pold = {}

         // Step 8
         // Decompose the problem, and for each sub-problem (corresponding to a
         // specific sub-net and some place constraints) apply the same procedure
         // from Step 1.
         PetriNet Gp;
         Vector <boolean[]> Ep;
         while (!Pnew.isEmpty()) {

            //Step 9
            int place = Pnew.removeTransition();
            Gp = reduceNet(G, G.P.getPlacesMinus(place));
            Ep = findAllMinimalSiphons(Gp, Ptilde.union(Pold));
            E.addAll(Ep);
            Pold.add(place);
         }
         return E;
      }


      /**
       * findSiphon()
       * Finds a generic siphon in a givenPetri Net G. This siphon will contain the
       * given set of places Ptilde.
       * @param G      A Petri Net
       * @param Ptilde A place constraint for the resultant siphon
       * @returns      A generic siphon which contains the given set of places
       */
      private SetOfPlaces findSiphon( PetriNet G, final SetOfPlaces Ptilde) {

         do {
            boolean[] placePreSet;

            // Step 1
            // check if exists a place that is element of the union of P and Ptilde
            // such that exists a transition t in its pre-set that is not an element
            // of P's post-set
            for (int place = 0; place < G.P.size(); place++) {
               if (G.P.contains(place) && Ptilde.contains(place)) {
                  // check each place that belongs to P and to ~P
                  placePreSet = G.getPlacePreSet(place);
                  for (int transition = 0; transition < placePreSet.length; transition++) {
                     if (placePreSet[transition] && !G.PPostSetcontains(transition)){
                        // There are no possible siphons that contain Ptilde, so
                        // findSiphon ends with empty outcome
//                        System.out.println("findSiphon returns an empty siphon");//dbg
                        return new SetOfPlaces(Ptilde.size(), false);
                     }
                  }
               }
            }

            // Step 2
```

```
         // check if there is a place which can be discarded
         int placeToEliminate = eliminablePlace_FS(G, Ptilde);
         if (placeToEliminate != -1) {
            // Step 3
            // perform the Petri Net reduction and go to Step 1
            G = reduceNet(G, G.P.getPlacesMinus(placeToEliminate));
         } else {
//            G.P.debug("findSiphon result is ");//dbg
            return new SetOfPlaces(G.P);
         }
      } while (true);
   }


   // Step 2) of algorithm "FindSiphon"
   private int eliminablePlace_FS(final PetriNet G, final SetOfPlaces Ptilde){
      boolean[] placePreSet;

      // check if exists a place that is element of P minus Ptilde
      // such that exists a transition t in its pre-set that is not an element
      // of P's post-set
      for (int place = 0; place < Ptilde.size(); place++) {
         if (G.P.contains(place) && !Ptilde.contains(place)) {
            placePreSet = G.getPlacePreSet(place);
            for (int transition = 0; transition < placePreSet.length; transition++) {
               if (placePreSet[transition] && !G.PPostSetcontains(transition)) {
                  return place; //
               }
            }
         }
      }
      return -1; // no place can be eliminated
   }


   /**
    * findMinimalSiphon()
    * Computes a mininal siphon in Petri Net G such that is contained in Stilde
    * and contains Ptilde, if exists.
    * @param G       A Petri Net
    * @param Stilde  A general siphon
    * @param Ptilde  A set of places that the minimal siphon must contain
    * @return        A minimal siphon
    */
   private SetOfPlaces findMinimalSiphon(PetriNet G, final SetOfPlaces Stilde,
         final SetOfPlaces Ptilde) {
      SetOfPlaces StildeCopy = new SetOfPlaces(Stilde);
      int placeToEliminate;
      boolean[] placePostSet;
      boolean[] transitionPostSet;
      boolean[] transitionPreSet;

      // Step 1
      do {
         placeToEliminate = eliminablePlace_FMS(G, StildeCopy, Ptilde);
         if (placeToEliminate != -1) {
            // Step 2
            // placeToEliminate can be removed from the given siphon
            StildeCopy.remove(placeToEliminate);
         }
      } while (placeToEliminate != -1);

      do {

         // Step 3
         if (G.P.containsSet(StildeCopy)) {
            G = reduceNet(G, StildeCopy.getPlaces());
         }

         // Step 4
         SetOfPlaces Pnew = G.P.minus(Ptilde);

         int newPlaceToEliminate;
         PetriNet Gp;
         SetOfPlaces Sp;

         do {
            // Step 5
            if (Pnew.isEmpty()) {
//               StildeCopy.debug("[FIND_MINIMAL_SIPHON] ~S"); //dbg
               return StildeCopy;
            }

            // Step 6
            newPlaceToEliminate = Pnew.removeTransition();
            Gp = reduceNet(G, G.P.getPlacesMinus(newPlaceToEliminate));
            Sp = findSiphon(Gp, Ptilde);
         } while (Sp.isEmpty());
         StildeCopy = Sp;
      } while (true);
```

```
}


// Step 1) of algorithm "FindMinimalSiphon"
// returns the index of a place that can be eliminated or -1 if there is no
// such place
private int eliminablePlace_FMS(final PetriNet G, final SetOfPlaces Stilde,
        final SetOfPlaces Ptilde) {

   int placeToEliminate = -1;
   boolean[] placePostSet;
   boolean[] transitionPostSet;
   boolean[] transitionPreSet;

   for (int place = 0; place < Ptilde.size(); place++) {
      if (G.P.contains(place) && !Ptilde.contains(place) &&
            Stilde.contains(place)) {
         // place 'place' is an element of the set P minus Ptilde
         placePostSet = G.getPlacePostSet(place);
         boolean eliminable = true;
         for (int transition = 0; transition < placePostSet.length; transition++) {
            if (placePostSet[transition]) {
               transitionPreSet = G.getTransitionPreSet(transition);
               transitionPostSet = G.getTransitionPostSet(transition);

               boolean containsCurrenPlace = false;
               if ((transitionPreSet[place] == true) && Stilde.contains(place)){
                  for (int currentPlace = 0; currentPlace < transitionPreSet.length; currentPlace++) {
                     if ((transitionPreSet[currentPlace] == true) &&
                           Stilde.contains(currentPlace) &&
                           currentPlace != place){
                        // transition pre-set intersection Stilde contains
                        // place 'place'
                        containsCurrenPlace = true;
                        break;
                     }
                  }
               }

               boolean tPostSetIntersectionStildeIsEmpty = true;
               for (int currentPlace = 0; currentPlace < transitionPostSet.length; currentPlace++) {
                  if (transitionPostSet[currentPlace] &&
                        Stilde.contains(currentPlace)){
                     tPostSetIntersectionStildeIsEmpty = false;
                     // transition post-set intersection Stilde is not empty
                     break;
                  }
               }

               if (!containsCurrenPlace &&
                     !tPostSetIntersectionStildeIsEmpty){
                  // place is not eliminable
                  eliminable = false;
                  break;
               }
            }
         }
         if (eliminable == true) {
            return place; // eliminable place
         }
      }
   }
   return -1; // there is no eliminable place
}


/**
 * reduceNet()
 * Simplifies a given PetriNet G discarding all places not in Ptilde and the
 * arcs connected with them.
 * @param G      A Petri Net
 * @param Ptilde A set of places
 * @returns      A simplified Petri Net
 */
private PetriNet reduceNet(final PetriNet G, final boolean[] Ptilde){
   PetriNet Gtilde = new PetriNet(G); // result

   int transitionCount = G.T.size();
   boolean[] transitionPreSet;
   boolean[] transitionPostSet;

   // for each transition in T, check if it can be discarded
   for (int transition = 0; transition < transitionCount; transition++) {
      if (G.T.contains(transition)) {
         transitionPreSet = G.getTransitionPreSet(transition);
         transitionPostSet = G.getTransitionPostSet(transition);
         boolean remove = true;
         for (int place = 0; place < Ptilde.length; place++) {
            if ((transitionPreSet[place] || transitionPostSet[place]) &&
                  Ptilde[place]){
```

```
                // the intersection Ptilde and the union of the transition's
                // pre-set and the transition's post-set is not empty, so this
                // transition can't be discarded
                remove = false;
                break;
             }
          }
          if (remove) {
             Gtilde.T.remove(transition);
          }
       }
    }

    // discard each place p that isn't an element of the set of places Ptilde
    // and its connecting arcs.
    for (int place = 0; place < Ptilde.length; place++) {
       if (Ptilde[place] == false){
          Gtilde.reduce(place);
       }
    }

//      System.out.println("\nreduceNet:"); //dbg
//      Gtilde.debug(); //dbg
    return Gtilde;
  }


  // used for debug
  private void print(String string, boolean[] b) {
    System.out.println(string);
    for (int i = 0; i < b.length; i++) {
      System.out.print(b[i] + " ");
    }
    System.out.println();
  }


  // helper class.
  // TODO: optimize it
  private class PetriNet{
    SetOfPlaces P;                    // set of places
    SetOfTransitions T;               // set of transitions
    PNMatrix forwardsIncidenceMatrix;  // input incidence matrix
    PNMatrix backwardsIncidenceMatrix; // output incidence matrix
    boolean[] PPostSet;               // union of the post-set of each place


    // constructor
    private PetriNet(PNMatrix _forwardsIncidenceMatrix,
            PNMatrix _backwardsIncidenceMatrix){
       forwardsIncidenceMatrix = _forwardsIncidenceMatrix.copy();
       backwardsIncidenceMatrix = _backwardsIncidenceMatrix.copy();
       P = new SetOfPlaces(forwardsIncidenceMatrix.getRowDimension(), true);
       T = new SetOfTransitions(forwardsIncidenceMatrix.getColumnDimension(), true);
       PPostSet = computePPostSet(P, T, forwardsIncidenceMatrix);
    }


    // constructor
    public PetriNet(int[][] _forwardsIncidenceMatrix,
            int[][] _backwardsIncidenceMatrix){
       forwardsIncidenceMatrix = new PNMatrix(_forwardsIncidenceMatrix);
       backwardsIncidenceMatrix = new PNMatrix(_backwardsIncidenceMatrix);
       P = new SetOfPlaces(forwardsIncidenceMatrix.getRowDimension(), true);
       T = new SetOfTransitions(forwardsIncidenceMatrix.getColumnDimension(), true);
       PPostSet = computePPostSet(P, T, forwardsIncidenceMatrix);
    }


    // constructor
    private PetriNet(PetriNet G){
       this(G.forwardsIncidenceMatrix, G.backwardsIncidenceMatrix);
    }


    // returns the index of a place which its pre-set is empty
    private int placeWithEmptyInputSet() {
       boolean [] placePreSet;
       boolean hasEmptyPreSet;

       for (int place = 0; place < P.size(); place++) {
          if (!P.contains(place)){
             continue;
          }
          placePreSet = this.getPlacePreSet(place);
          hasEmptyPreSet = true;
          for (int i = 0; i < placePreSet.length; i++) {
             if (placePreSet[i]) {
                hasEmptyPreSet = false;
                break;
```

```
         }
      }
      if (hasEmptyPreSet) {
         return place;
      }
   }
   return -1;
}


// returns the pre-set of a given place
private boolean[] getPlacePreSet(int place) {
   int[] column = forwardsIncidenceMatrix.getColumn(place);
   boolean[] result = new boolean[column.length];

   for (int i = 0; i < column.length; i++) {
      result[i] = (column[i] > 0);
   }
   return result;
}


// returns the post-set of a given place
private boolean[] getPlacePostSet(int place) {
   int[] column = backwardsIncidenceMatrix.getColumn(place);
   boolean[] result = new boolean[column.length];

   for (int i = 0; i < column.length; i++) {
      result[i] = (column[i] > 0);
   }
   return result;
}


// returns the pre-set of a given transition
private boolean[] getTransitionPreSet(int transition) {
   int[] row = backwardsIncidenceMatrix.getRow(transition);
   boolean[] result = new boolean[row.length];

   for (int i = 0; i < row.length; i++) {
      result[i] = (row[i] > 0);
   }
   return result;
}


// returns the post-set of a given transition
private boolean[] getTransitionPostSet(int transition) {
   int[] column = forwardsIncidenceMatrix.getRow(transition);
   boolean[] result = new boolean[column.length];

   for (int i = 0; i < column.length; i++) {
      result[i] = (column[i] > 0);
   }
   return result;
}


// return the union of each transition post-set
private boolean PPostSetcontains(int transition) {
   return PPostSet[transition];
}


// removes a place from P and clears its columns in forwards and backwards
// incidence matrices
private void reduce(int place) {
   P.remove(place);
   forwardsIncidenceMatrix.clearColumn(place);
   backwardsIncidenceMatrix.clearColumn(place);
   // P's post-set must be computed again!
   PPostSet = computePPostSet(P, T, backwardsIncidenceMatrix);
}


// computes the union of each transition post-set
private boolean[] computePPostSet(SetOfPlaces P, SetOfTransitions T,
        PNMatrix forwardsIncidenceMatrix) {
   boolean[] result = new boolean [T.size()];

   for (int i = 0; i < result.length; i++) {
      result[i] = false;
   }

   for (int transition = 0; transition < result.length; transition++) {
      result[transition] = false;
      for (int place = 0; place < P.size(); place++) {
         if (forwardsIncidenceMatrix.get(place, transition) > 0) {
            result[transition] = true;
            break;
```

```
            }
         }
      }
      return result;
   }


   // prints info for debug
   private void debug() {
      P.debug("P");
      T.debug("T");
      System.out.println("");
      System.out.print("Forwards Incidence Matrix");
      forwardsIncidenceMatrix.print(
              forwardsIncidenceMatrix.getColumnDimension(), 0);
      System.out.print("Backwards Incidence Matrix");
      backwardsIncidenceMatrix.print(
              backwardsIncidenceMatrix.getColumnDimension(), 0);

      System.out.print("P PostSet = { ");
      for (int i = 0; i < PPostSet.length; i++) {
         System.out.print(PPostSet[i]? i + " " : "");
      }
      System.out.println("}");
   }

}


// helper class.
// TODO: optimize it
private class SetOfPlaces{
   boolean[] P; // set of places


   // constructor
   SetOfPlaces(int length){
      this(length, false);
   }


   // constructor
   SetOfPlaces(int lenght, boolean flag){
      P = new boolean[lenght];

      for (int place = 0; place < lenght; place++) {
         P[place] = flag;
      }
   }


   // constructor
   SetOfPlaces(SetOfPlaces set){
      P = new boolean[set.size()];

      for (int place = 0; place < P.length; place++) {
         P[place] = set.P[place];
      }
   }


   // returns true if each element of P is false
   private boolean isEmpty() {
      for (int j = 0; j < P.length; j++) {
         if (P[j] != false){
            return false;
         }
      }
      return true;
   }


   // sets to true position t of P
   private void add(int t) {
      P[t] = true;
   }


   // returns a copy of P
   private boolean[] getPlaces() {
      return P.clone();
   }


   // returns a copy of P with position i set to false
   private boolean[] getPlacesMinus(int i) {
      boolean[] result = P.clone();
      result[i] = false;
      return result;
   }
```

```
   // returns a set of places S so that S = P - Ptilde
   private SetOfPlaces minus(SetOfPlaces Ptilde) {
      SetOfPlaces result = new SetOfPlaces(P.length);
      for (int i = 0; i < result.size(); i++) {
         result.P[i] = P[i];
         if (Ptilde.P[i]) {
            result.P[i] = false;
         }
      }
      return result;
   }


   // returns index of the first position of P that is true and removes it
   // from P; if P is empty, returns -1
   private int removeTransition() {
      for (int place = 0; place < P.length; place++) {
         if (P[place] == true) {
            P[place] = false;
            return place;
         }
      }
      return -1;
   }


   // returns a set of places S so that S = P U Pold
   private SetOfPlaces union(SetOfPlaces Pold) {
      SetOfPlaces result = new SetOfPlaces(Pold.size());
      for (int i = 0; i < result.size(); i++) {
         result.P[i] = P[i] || Pold.P[i];
      }
      return result;
   }


   // sets to false position i of P
   private void remove(int i) {
      P[i] = false;
   }


   // returns size of P
   private int size() {
      return P.length;
   }


   //returns true if position i of P is true
   private boolean contains(int i) {
      return P[i];
   }


   // returns true if P strictly contains Stilde
   private boolean containsSet(SetOfPlaces Stilde) {
      boolean containsSet = false;
      for (int place = 0; place < P.length; place++) {
         if (Stilde.contains(place) && !this.contains(place)){
            return false;
         }
         if (!Stilde.contains(place) && this.contains(place)){
            containsSet = true; //OK, Stilde and this are not equal
         }
      }
      return containsSet;
   }


   // used for debug purposes
   private void debug(String s) {
      System.out.print(s + " = { ");
      for (int j = 0; j < P.length; j++) {
         System.out.print(P[j]? j + " " : "");
      }
      System.out.println("}");
   }

}


// helper class.
// TODO: optimize it
private class SetOfTransitions{
   boolean[] T; // set of places


   // constructor
```

```
    SetOfTransitions(int length, boolean flag){
       T = new boolean[length];

       for (int j = 0; j < length; j++) {
          T[j] = flag;
       }
    }


    // returns size of T
    private int size() {
       return T.length;
    }


    //returns true if position i of T is true
    private boolean contains(int i) {
       return T[i];
    }


    // sets to false position i of T
    private void remove(int transition) {
       T[transition] = false;
    }


    // used for debug purposes
    private void debug(String s) {
       System.out.print(s + " = { ");
       for (int j = 0; j < T.length; j++) {
          System.out.print(T[j]? j + " " : "");
       }
       System.out.println("}");
    }

  }

}
```

Changed version has only a small change in the *findAllMinimalSyphons* function:

```
 /**
   * findAllMinimalSyphons()
   * Finds all minimal syphons in a given Petri Net that contain a specific
   * set of places.
   * @param G      A Petri Net
   * @param Ptilde A set of places that each syphon must contain
   * returns       A vector containg all minimal syphons found
   */
  private Vector <boolean[]> findAllMinimalSyphons(PetriNet G, SetOfPlaces Ptilde){
     Vector <boolean[]> E; // contains all minimal syphons found
     SetOfPlaces S = new SetOfPlaces(Ptilde.size()); // a syphon

     // Step 1
     E = new Vector();

     // Step 2
     // check if there is a place with an empty pre-set
     int p = G.placeWithEmptyInputSet();
     while (Ptilde.isEmpty() && (p != -1)){
        // Step 3
        S.add(p);
        E.add(S.getPlaces());

        // set S to allEmpty;
        // else the isolated places p1, p2 , p3
        // will return: {p1} {p1,P2} {p1,p2,p3}
        // instead of {p1} {2p} {p3}
        S.remove(p);
        // the Petri Net can be reduced by eliminating p

        // G.P.getPlaceMinus(p) instead of Ptilde.getPlacesMinus(p)
        // Ptilde is already empty (while loop condition)
        // you wish to subtract p from the entire net, G
        G = reduceNet(G,G.P.getPlacesMinus(p));
        // check if there is another place with an empty pre-set
        p = G.placeWithEmptyInputSet();
     }

     // Step 4
     // Find a generic syphon
     SetOfPlaces Stilde = findSyphon(G, Ptilde);
```

```
      // Step 5
      if (Stilde.isEmpty()) {
         // No syphon has been found, return E
         return E;
      }
```

load bug When a net was loaded and contained places with capacity, these places were not on their original positions. This ruined any nice layout of the net. With a very simple modification of the code this has been fixed.
Original code:

```
package pipe.dataLayer;

import java.awt.BasicStroke;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Insets;
import java.awt.RenderingHints;
import java.awt.Shape;
import java.awt.geom.Ellipse2D;
import javax.swing.BoxLayout;

import pipe.gui.CreateGui;
import pipe.gui.Pipe;
import pipe.gui.Grid;
import pipe.gui.undo.ChangeMarkingParameterEdit;
import pipe.gui.undo.ClearMarkingParameterEdit;
import pipe.gui.undo.SetMarkingParameterEdit;
import pipe.gui.undo.UndoableEdit;
import pipe.gui.undo.PlaceCapacityEdit;
import pipe.gui.undo.PlaceMarkingEdit;
import pipe.gui.widgets.PlaceEditorPanel;

import pipe.gui.Zoomer;
import pipe.gui.widgets.EscapableDialog;


/**
 * <b>Place</b> - Petri-Net Place Class
 *
 * @see <p><a href="..\PNMLSchema\index.html">PNML  -  Petri-Net XMLSchema (stNet.xsd)</a>
 * @see </p><p><a href="..\..\..\UML\dataLayer.html">UML  -  PNML Package </a></p>
 * @version 1.0
 * @author James D Bloom
 *
 * @author Edwin Chung corresponding states of matrixes has been set
 * to change when markings are altered. Users will be prompted to save their
 * work when the markings of places are altered. (6th Feb 2007)
 *
 * @author Edwin Chung 16 Mar 2007: modified the constructor and several
 * other functions so that DataLayer objects can be created outside the
 * GUI
 */
public class Place
        extends PlaceTransitionObject {

   public final static String type = "Place";
   /** Initial Marking */
   private Integer initialMarking = 0;

   /** Current Marking */
    private Integer currentMarking = 0;

   /** Initial Marking X-axis Offset */
   private Double markingOffsetX = 0d;

   /** Initial Marking Y-axis Offset */
   private Double markingOffsetY = 0d;

   /**  Value of the capacity restriction; 0 means no capacity restriction */
   private Integer capacity = 0;
   /*
   private boolean strongCapacity = false;
  */

   /** Initial Marking X-axis Offset */
   private Double capacityOffsetX = 0.0;

   /** Initial Marking Y-axis Offset */
   private Double capacityOffsetY = 22.0;
```

```
public static final int DIAMETER = Pipe.PLACE_TRANSITION_HEIGHT;

/** Token Width */
public static int tWidth = 4;

/** Token Height */
public static int tHeight = 4;

/** Ellipse2D.Double place */
private static Ellipse2D.Double place =
         new Ellipse2D.Double(0, 0, DIAMETER, DIAMETER);
private static Shape proximityPlace =
         (new BasicStroke(Pipe.PLACE_TRANSITION_PROXIMITY_RADIUS)).createStrokedShape(place);

private MarkingParameter markingParameter = null;

/**
 * Create Petri-Net Place object
 * @param positionXInput X-axis Position
 * @param positionYInput Y-axis Position
 * @param idInput Place id
 * @param nameInput Name
 * @param nameOffsetXInput Name X-axis Position
 * @param nameOffsetYInput Name Y-axis Position
 * @param initialMarkingInput Initial Marking
 * @param markingOffsetXInput Marking X-axis Position
 * @param markingOffsetYInput Marking Y-axis Position
 * @param capacityInput Capacity
 */
public Place(double positionXInput,  double positionYInput,
             String idInput,
             String nameInput,
             Double nameOffsetXInput, Double nameOffsetYInput,
             int initialMarkingInput,
             double markingOffsetXInput,  double markingOffsetYInput,
             int capacityInput){
   super(positionXInput, positionYInput,
         idInput,
         nameInput,
         nameOffsetXInput, nameOffsetYInput);
   initialMarking = new Integer(initialMarkingInput);
   currentMarking = new Integer(initialMarkingInput);
   markingOffsetX = new Double(markingOffsetXInput);
   markingOffsetY = new Double(markingOffsetYInput);
   componentWidth = DIAMETER;
   componentHeight = DIAMETER;
   setCapacity(capacityInput);
   setCentre((int)positionX, (int)positionY);
   //updateBounds();
}


/**
 * Create Petri-Net Place object
 * @param positionXInput X-axis Position
 * @param positionYInput Y-axis Position
 */
public Place(double positionXInput, double positionYInput){
   super(positionXInput, positionYInput);
   componentWidth = DIAMETER;
   componentHeight = DIAMETER;
   setCentre((int)positionX, (int)positionY);
   //updateBounds();
}


public Place paste(double x, double y, boolean fromAnotherView){
   Place copy = new Place (
           Grid.getModifiedX(x + this.getX() + Pipe.PLACE_TRANSITION_HEIGHT/2),
           Grid.getModifiedY(y + this.getY() + Pipe.PLACE_TRANSITION_HEIGHT/2));
   copy.pnName.setName(this.pnName.getName()
                       + "(" + this.getCopyNumber() +")");
   this.newCopy(copy);
   copy.nameOffsetX = this.nameOffsetX;
   copy.nameOffsetY = this.nameOffsetY;
   copy.capacity = this.capacity;
   copy.attributesVisible = this.attributesVisible;
   copy.initialMarking = this.initialMarking;
   copy.currentMarking = this.currentMarking;
   copy.markingOffsetX = this.markingOffsetX;
   copy.markingOffsetY = this.markingOffsetY;
   copy.markingParameter = this.markingParameter;
   copy.update();
   return copy;
}


public Place copy(){
   Place copy = new Place (Zoomer.getUnzoomedValue(this.getX(), zoom),
                           Zoomer.getUnzoomedValue(this.getY(), zoom));
```

```
   copy.pnName.setName(this.getName());
   copy.nameOffsetX = this.nameOffsetX;
   copy.nameOffsetY = this.nameOffsetY;
   copy.capacity = this.capacity;
   copy.attributesVisible = this.attributesVisible;
   copy.initialMarking = this.initialMarking;
   copy.currentMarking = this.currentMarking;
   copy.markingOffsetX = this.markingOffsetX;
   copy.markingOffsetY = this.markingOffsetY;
   copy.markingParameter = this.markingParameter;
   copy.setOriginal(this);
   return copy;
}


/**
 * Paints the Place component taking into account the number of tokens from
 * the currentMarking
 * @param g The Graphics object onto which the Place is drawn.
 */
public void paintComponent(Graphics g) {
   super.paintComponent(g);
   Graphics2D g2 = (Graphics2D)g;

   Insets insets = getInsets();
   int x = insets.left;
   int y = insets.top;

   if (hasCapacity()){
      g2.setStroke(new BasicStroke(2.0f));
   } else {
      g2.setStroke(new BasicStroke(1.0f));
   }
   g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                       RenderingHints.VALUE_ANTIALIAS_ON);

   if(selected && !ignoreSelection){
      g2.setColor(Pipe.SELECTION_FILL_COLOUR);
   } else{
      g2.setColor(Pipe.ELEMENT_FILL_COLOUR);
   }
   g2.fill(place);

   if (selected && !ignoreSelection){
      g2.setPaint(Pipe.SELECTION_LINE_COLOUR);
   } else{
      g2.setPaint(Pipe.ELEMENT_LINE_COLOUR);
   }
   g2.draw(place);

   g2.setStroke(new BasicStroke(1.0f));
   int marking = getCurrentMarking();

   // structure sees how many markings there are and fills the place in with
   // the appropriate number.
   switch(marking) {
      case 5:
         g.drawOval(x + 6, y + 6, tWidth, tHeight);
         g.fillOval(x + 6, y + 6, tWidth, tHeight);
         /* falls through */
      case 4:
         g.drawOval(x + 18, y + 20, tWidth, tHeight);
         g.fillOval(x + 18, y + 20, tWidth, tHeight);
         /* falls through */
      case 3:
         g.drawOval(x + 6, y + 20, tWidth, tHeight);
         g.fillOval(x + 6, y + 20, tWidth, tHeight);
         /* falls through */
      case 2:
         g.drawOval(x + 18, y + 6, tWidth, tHeight);
         g.fillOval(x + 18, y + 6, tWidth, tHeight);
         /* falls through */
      case 1:
         g.drawOval(x + 12, y + 13, tWidth, tHeight);
         g.fillOval(x + 12, y + 13, tWidth, tHeight);
         break;
      case 0:
         break;
      default:
         if (marking > 999){
            g.drawString(String.valueOf(marking), x, y + 20);
         } else if (marking > 99){
            g.drawString(String.valueOf(marking), x + 3, y + 20);
         } else if (marking > 9){
            g.drawString(String.valueOf(marking), x + 7, y + 20);
         } else {
            g.drawString(String.valueOf(marking), x + 12, y + 20);
         }
         break;
   }
```

```
}


/**
 * Set initial marking
 * @param initialMarkingInput Integer value for initial marking
 */
public void setInitialMarking(int initialMarkingInput) {
   initialMarking = new Integer(initialMarkingInput);
}


/**
 * Set current marking
 * @param currentMarkingInput Integer value for current marking
 */
public UndoableEdit setCurrentMarking(int currentMarkingInput) {
   int oldMarking = currentMarking;

   if (capacity == 0){
      currentMarking = currentMarkingInput;
   } else {
      if (currentMarkingInput > capacity) {
         currentMarking = capacity;
      } else{
         currentMarking = currentMarkingInput;
      }
   }
   repaint();
   return new PlaceMarkingEdit(this, oldMarking, currentMarking);
}


/**
 * Set capacity
 * This method doesn't check if marking fulfilles current capacity restriction
 * @param newCapacity Integer value for capacity restriction
 */
public UndoableEdit setCapacity(int newCapacity) {
   int oldCapacity = capacity;

   if (capacity != newCapacity) {
      capacity = newCapacity;
      update();
   }
   return new PlaceCapacityEdit(this, oldCapacity, newCapacity);
}

/**
 * Get initial marking
 * @return Integer value for initial marking
 */
public int getInitialMarking() {
   return ((initialMarking == null) ? 0 : initialMarking.intValue());
}


/**
 * Get current marking
 * @return Integer value for current marking
 */
public int getCurrentMarking() {
   return ((currentMarking == null) ? 0 : currentMarking.intValue());
}


/**
 * Get current capacity
 * @return Integer value for current capacity
 */
public int getCapacity() {
   return ((capacity == null) ? 0 : capacity.intValue());
}


/**
 * Get current marking
 * @return Integer value for current marking
 */
public Integer getCurrentMarkingObject() {
   return currentMarking;
}


/**
 * Get X-axis offset for initial marking
 * @return Double value for X-axis offset of initial marking
 */
public Double getMarkingOffsetXObject() {
   return markingOffsetX;
```

```
}


/**
 * Get Y-axis offset for initial marking
 * @return Double value for X-axis offset of initial marking
 */
public Double getMarkingOffsetYObject() {
   return markingOffsetY;
}


/**
 * Returns the diameter of this Place at the current zoom
 */
private int getDiameter() {
   return (int)(Zoomer.getZoomedValue(DIAMETER, zoom));
}


public boolean contains(int x, int y) {
   double unZoomedX =
            Zoomer.getUnzoomedValue(x - COMPONENT_DRAW_OFFSET, zoom);
   double unZoomedY =
            Zoomer.getUnzoomedValue(y - COMPONENT_DRAW_OFFSET, zoom);

   someArc = CreateGui.getView().createArc;
   if (someArc != null){ // Must be drawing a new Arc if non-NULL.
      if ((proximityPlace.contains((int)unZoomedX, (int)unZoomedY)
               || place.contains((int)unZoomedX, (int)unZoomedY))
               && areNotSameType(someArc.getSource())){
         // assume we are only snapping the target...
         if (someArc.getTarget() != this){
            someArc.setTarget(this);
         }
         someArc.updateArcPosition();
         return true;
      } else {
         if (someArc.getTarget() == this) {
            someArc.setTarget(null);
            updateConnected();
         }
         return false;
      }
   } else {
      return place.contains((int)unZoomedX, (int)unZoomedY);
   }
}


/* (non-Javadoc)
 * @see pipe.dataLayer.PlaceTransitionObject#updateEndPoint(pipe.dataLayer.Arc)
 */
public void updateEndPoint(Arc arc) {
   if (arc.getSource()==this) {
      // Make it calculate the angle from the centre of the place rather than
      // the current start point
      arc.setSourceLocation(positionX + (getDiameter() * 0.5),
                            positionY + (getDiameter() * 0.5));
      double angle = arc.getArcPath().getStartAngle();
      arc.setSourceLocation(positionX + centreOffsetLeft()
                                - (0.5 * getDiameter() * (Math.sin(angle))),
                            positionY + centreOffsetTop()
                                + (0.5 * getDiameter() * (Math.cos(angle))));
   } else {
      // Make it calculate the angle from the centre of the place rather than the current target point
      arc.setTargetLocation(positionX + (getDiameter() * 0.5),
                            positionY + (getDiameter() * 0.5));
      double angle = arc.getArcPath().getEndAngle();
      arc.setTargetLocation(positionX + centreOffsetLeft()
                                - (0.5 * getDiameter() * (Math.sin(angle))),
                            positionY + centreOffsetTop()
                                + (0.5 * getDiameter() * (Math.cos(angle))));
   }
}


public void toggleAttributesVisible(){
   attributesVisible = !attributesVisible;
   update();
}


public boolean hasCapacity(){
   return capacity > 0;
}


public void addedToGui(){
   super.addedToGui();
```

```
      update();
   }


   public void showEditor(){
      // Build interface
      EscapableDialog guiDialog =
               new EscapableDialog(CreateGui.getApp(), "PIPE2", true);

      Container contentPane = guiDialog.getContentPane();

      // 1 Set layout
      contentPane.setLayout(new BoxLayout(contentPane,BoxLayout.PAGE_AXIS));

      // 2 Add Place editor
      contentPane.add( new PlaceEditorPanel(guiDialog.getRootPane(),
               this, CreateGui.getModel(), CreateGui.getView()));

      guiDialog.setResizable(false);

      // Make window fit contents' preferred size
      guiDialog.pack();

      // Move window to the middle of the screen
      guiDialog.setLocationRelativeTo(null);
      guiDialog.setVisible(true);
   }


   public void update() {
      if (attributesVisible == true){
         pnName.setText("\nk=" + (capacity > 0 ? capacity :"\u221E") +
            (markingParameter != null ? "\nm=" + markingParameter.toString() : ""));
      } else {
         pnName.setText("");
      }
      pnName.zoomUpdate(zoom);
      super.update();
      repaint();
   }


   public void delete() {
      if (markingParameter != null) {
         markingParameter.remove(this);
         markingParameter = null;
      }
      super.delete();
   }


   public UndoableEdit setMarkingParameter(MarkingParameter _markingParameter) {
      int oldMarking = currentMarking;

      markingParameter = _markingParameter;
      markingParameter.add(this);
      currentMarking = markingParameter.getValue();
      update();
      return new SetMarkingParameterEdit (this, oldMarking, markingParameter);
   }


   public UndoableEdit clearMarkingParameter() {
      MarkingParameter oldMarkingParameter = markingParameter;

      markingParameter.remove(this);
      markingParameter = null;
      update();
      return new ClearMarkingParameterEdit (this, oldMarkingParameter);
   }


   public UndoableEdit changeMarkingParameter(MarkingParameter _markingParameter) {
      MarkingParameter oldMarkingParameter = markingParameter;

      markingParameter.remove(this);
      markingParameter = _markingParameter;
      markingParameter.add(this);
      currentMarking = markingParameter.getValue();
      update();
      return new ChangeMarkingParameterEdit(
               this, oldMarkingParameter, markingParameter);
   }


   public MarkingParameter getMarkingParameter() {
      return markingParameter;
   }

}
```

Modification in the constructor of the *Place* class:

```
public Place(double positionXInput,  double positionYInput,
             String idInput,
             String nameInput,
             Double nameOffsetXInput, Double nameOffsetYInput,
             int initialMarkingInput,
             double markingOffsetXInput,  double markingOffsetYInput,
             int capacityInput){
   super(positionXInput, positionYInput,
         idInput,
         nameInput,
         nameOffsetXInput, nameOffsetYInput);
   initialMarking = new Integer(initialMarkingInput);
   currentMarking = new Integer(initialMarkingInput);
   markingOffsetX = new Double(markingOffsetXInput);
   markingOffsetY = new Double(markingOffsetYInput);
   componentWidth = DIAMETER;
   componentHeight = DIAMETER;

   // changed order
   // if setCapacity is before setCentre then the
   // places are placed wrong when loading a net
   // with places that have capacity higher than 0
   setCentre((int)positionX, (int)positionY);
   setCapacity(capacityInput);
   //updateBounds();
}
```