Thesis Diploma

# A SIMULATOR FOR REVERSING PETRI NETS

**Pantelina Ioannou**

## UNIVERSITY OF CYPRUS

## DEPARTMENT OF COMPUTER SCIENCE

**May 2019**

# UNIVERSITY OF CYPRUS

## DEPARTMENT OF COMPUTER SCIENCE


# A Simulator for Reversing Petri Nets


**Pantelina Ioannou**

Supervisor

Dr. Anna Philippou


Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

May 2019

# Acknowledgments

Foremost I would like to express my gratitude to my supervisor Dr. Anna Philippou for the continuous support of my Diploma thesis study and research, for her patience, motivation and the excellent cooperation we have had throughout this project. I want to thank her for all the knowledge she provided to me, which will be very lucrative for my future career, and the opportunity to explore new intriguing areas of Computer Science through my Diploma Thesis.

Besides my supervisor, I would like to thank the PhD student, Kyriaki Psara for her continuous help and support during my Diploma thesis. She was always available for me for any technical issues and for providing me with further knowledge about the topic.

I would also want to thank many of my fellow students and friends, for their help and support at every step of the way, and for making this experience more creative.

Last but not least, I would like to thank my family, because they were next to me and were supporting me all this time, at any problem that I had to deal with.

# Abstract

This diploma thesis deals with the issue of reversible computation and the application of it in Petri Nets. Reversible computation is a different form of computation, where any executed sequence of operations can be executed in reverse at any point during computation. Thus, this form of computation has the ability to retrieve any previous visited states or even attain new states, which are not reachable with any forward execution. Reversibility is embedded in various natural or artificial processes, such as biological processes, reliable systems, and quantum computations. Petri Nets, on the other hand, are a graphical mathematical language, that can be used for specification and analysis of discrete event systems, like those mentioned before.

Petri Nets have been used broadly to modelling and reasoning about a wide range of applications. A property studied in the context of Petri Nets is reversibility, and there are three methodologies of reversibility in the framework of Petri Nets, besides the Forward execution. The reversible mechanisms are Backtracking, Causal reversibility, and Out-of-Causal reversibility.

The definition and semantics of the four mechanisms have been used for the implementation of a Simulator for Reversible Petri Nets. The simulator gives the opportunity to the user to import the initial marking of a Reversible Petri Net, and then analyze the possible forward or backward executions that the Petri Net can reach. The user can select which forward or backward enabled transition, he/she desires to execute, and then watch how the marking of the net evolves in a computation. Furthermore, the simulator has the ability to represent the Petri Net graphically, through the Obeo designer software, which constitutes an Eclipse plug-in software. The simulator has been implemented in the Java programming language, and when the user selects to use graphical representation, the simulator opens the environment of the Obeo designer. Through the Obeo designer environment the user can directly watch the changes at the Petri Net, when an action is caused from the simulator.

# Table of Contents

# Table of Figures

# Chapter 1

## Introduction

---



---

## 1.1 Motivation

Petri Nets are capable to model parallel and distributed systems, in order to assist users and developers to better understand what a system is supposed to do, and how it can be implemented, used, varied, and improved. Moreover they help the user to analyze such systems via model checking.

Reversible Computation, on the other hand, it is a concept that has studied in the context of Petri Nets. This form of computation enables systems to perform actions in reverse, with the same effort as performing forward-only actions. Generally, reversibility is inherent in various applications and sciences, such as Biology, Mathematics and Physics. Thus, the notion of reversibility in Petri nets gives the opportunity to return to the initial state from any reachable state, by reversing the effects of already, executed transitions.

Although, the modelling of a system using Reversible Petri Nets may be too complicated, for systems with a lot of components and possible actions. The model checking process of these systems , it is difficult even without reversibility. So, it can be more complicated with the addition of reversibility and without an automated tool. So, a tool that will be able to simulate the actions of a given Reversible Petri Net is necessary. It is important that the tool will simulate the behavior of these systems, and be able to capture all the mechanisms of reversibility.

In addition with the need of a simulator for Reversible Petri Nets, there is also the need for a graphical representation of the model, both in the creation and update process of the Petri Net, so the user can see the changes visually in the model.

## 1.2 Work Purpose

The main aim of this diploma thesis is the creation of an Object Oriented approach to Reversible Petri Nets [8], and the implementation of this approach in a simulator. The simulator supports four mechanisms of reversible computation as defined in [8], the forward execution, and the three main strategies of reversible computation, namely backtracking, causal, and out-of-causal execution.

Furthermore the simulator contains a component that gives to the user the opportunity to create the initial marking with a graphical representation, and then during the execution should watch the changes that occur in the Petri Net.

## 1.3 Work Methodology

At the first part of the thesis, a theoretical research was conducted, about Petri Nets, Reversible Computation, the Forms of Reversibility, and Reversible Petri Nets, in order to fully understand the appropriate knowledge about the semantics of the topic. In order to achieve the mentioned above knowledge, an extensive study took place in various scientific papers, related to the field.

When the necessary knowledge was gained, the first step was the specification of requirements and the design of the simulator. At the design phase of the simulator two diagrams have created, a component diagram and a class diagram.

Then, based on the diagrams an object oriented approach of Reversible Petri Nets has been created. Initially, the necessary objects of the representation were found, and then the attributes of each object. Thereafter, the connections and relationships between the objects were decided. So, at this phase the object of a RPN was built.

The next phase of the thesis was the creation of code for the simulator. The implementation of the algorithms and the graphical user interface were done in Java programming language. Each Java Class is an Object, which consists of the appropriate attributes, and then a Petri Net is a Java Class which consists of all other Classes, as

attributes, and some extra fields. Afterwards, the algorithms were split to four classes, and each class contains the different functions of each algorithm.

After the implementation of the algorithms, the form of input that the user has to give to the simulator in order to do actions was studied. The final decision was to give to the user three choices: give a file as input, give the input manually, or draw the petri net.

At the last part of the thesis, a research about tools that can help with the development of Petri Nets' graphical representation was done. The extensive research ended up with the use of Obeo Designer software. Obeo Designer software helped with the formation of the metamodel and the code generation of the representation. The next step was to create the components of the palette, based on the metamodel relationships. Finally, when all the components were completed, there was a need for a Parser. Thus, a Parser that was done with Java programming language was developed, in order to parse the xml output of Obeo designer, and convert it to the appropriate input for the simulator. At a later phase a Reverse Parser, which updates the xml file when a change occurs to the model, was developed. The last step was to connect the graphical user interface with the graphical representation environment.

When all the necessary components of the simulator had been developed, some testings took place. The simulator has been tested for simple examples at first, and then for more complicated in order to ensure the correctness on its results and its effectiveness.

## 1.4 Thesis Structure

This thesis document is composed of another four chapters.

More specifically Chapter 2 contains the scientific background. There are subchapters explaining Reversible Computation, Forms of Reversibility, Petri Nets, and Reversible Petri Nets. Afterwards, there are the forms of reversibility on reversible Petri Nets.

Chapter 3 presents tools, software, and programming languages used for the completion of all components of the simulator. Specifically, there are subchapters for Java Programming language, and afterwards the reasons of using it and how the object oriented approach created for Petri Nets. Afterwards there is a subchapter that contains UML language and it follows a subchapter for Obeo Designer Software. The last

element of this chapter is the Requirements specification that is an introduction to the simulator.

Chapter 4 analyses the main components of the simulator. Initially, there is an architecture diagram of the tool, which explains the structure of the simulator and how components are connected with each other. Afterwards, subchapters for each component exist, namely Simulator functions, Graphical representation. Each subchapter consists of its own sections that explain the implementation of the algorithms and user interface, and the graphical representation of Petri Nets, as well as, the connection between them.

Finally, in Chapter 5 a general conclusion is drawn. That chapter mentions the problems encountered and how we deal with them, some limitations that exist, along with future extensions that can be done in the simulator, in a later research, to upgrade it.

Pseudocode for each algorithm can be found in Chapter 4, below the description of each algorithm. The complete Java code implementations of the algorithms are listed in Appendix A. Moreover the Java code of Simulator's Graphical User Interface and for parsers, which connects the simulator's environment with the graphical representation tool, is listed in Appendix B and Appendix C, respectively. Appendix E contains the Simulator's Manual for users.

# Chapter 2

## Scientific Background and Related Work



## 2.1 Reversible Computation

Reversible computation [8] [7] is a fundamental concept in sciences that extends the standard forward-only mode of computation. It is the study of computation models that exhibit both forward and backward determinism. Reversibility is the ability to execute sequences of operations in reverse at any point during the computation. With this form of computation individual operations can be carried out reversibly, as effortlessly as they can be executed in the standard forward direction. Thus, this form of computation has the ability to retrieve any previous visited states or even attain new states, which are not reachable with any forward execution. It is possible to attain new states because reversibility enables the undoing of an event before its effects have been undone.

The concept of reversibility [9] is inherent in many other sciences apart for Computer Science, such as Mathematics, Physics, and Biology. In many biological processes, for instance, computation may be carried out in forward or backward direction, to have the expected reaction. Furthermore reversible computing attracts interest in a burgeoning number of application areas, namely cellular automata, software architectures, reversible programming languages, digital circuit design, and quantum computing.

Reversible computation [9] integrates thermodynamics and information theory, in order to reflect one of the dominant physical properties of Nature. The motivation for reversible computing began with Landauer's principle, in 1961 [9]. Rolf Landauer was

the first to argue the relation between thermodynamics and the irreversible character of conventional computers. Specifically, Landauer observed that only irreversible computation generates heat and he noted that, while all of the fundamental laws of physical dynamics are reversible, such as classical and quantum mechanics, then conceptually any machine should be capable to run backwards, as well as forwards. Thus, Landauer's principle says that in order for a computational process to be physically reversible, it must also be logically reversible.

After Landauer's discovery, Lecerf in 1963 [9] was the first to explain reversible computations executed on reversible Turing Machines, but he was unaware of Landauer's thermodynamic applications, and his machines did not save their outputs. A decade after, Bennet reconceived Lecerf's reversible Turing machines based on Landauer's principle, and proved that it is achievable to build fully reversible Turing machines.[9]

There is still ongoing research for Reversible computation, as it promises the design implementation of reversible circuits and logic gates [12], which will increase the speed and capacity of circuits. Moreover reversible computation has lower-energy consumption, and indicates low-power computation.

Reversibility should also be used for the development of reliable systems. The reason is because backward recovery is an instance of reversible computation. When errors occur, it means that the behavior of the system is different than the expected one, and fault tolerance has to eliminate the presence of faults and provide the required level of service. This can be achieved with reversible computing because it gives the ability to return to previous states, which are safe, if a problem occurs after an action. This will give the opportunity to the system to find a new direction, and avoid the problematic actions [9].

Another area that reversibility has been applied is Robotics. Robotics represents a real-world application area where computational reversibility has a physical counterpart. Several kinds of actions performed by robots can be physically reversed, as for instance the change of direction in which a mobile robot is driving, or the reverse of an industrial assembly process [5].

Comprehensively, reversible computation adds an entirely new, "orthogonal" dimension to almost all aspects of traditional computing [2].

## 2.2 Forms of Reversibility

As mentioned above, a computation is called reversible when it has the ability to execute in reverse, and thus it can go back to previously visited states, or even reach new states, that were not reachable with the forward-only computation. In order for reversible computation to be applied to models, there are three forms of reversibility, which are backtracking, causal and out-of-causal reversibility.

Backtracking is the simplest form of reversibility as it is the process of undoing computational steps in the exact inverse order that they have occurred, as shown in Figure 2.1. This form of reversibility assures that at any state of computation there is at most one predecessor state and in, the context of concurrent systems, this phenomenon can cause fake dependencies on backward sequences of actions. Thus, backtracking reversing can be thought of as an overly restrictive form due to the fake dependencies they may cause on the backward sequences of actions, an action that could be reversed in any order, is forced to be undone in the precise order that are occurred.
Considering the Figure below (2.1) we can observe that the only choice in backtracking is to execute the events in the same order as they have occurred in the forward execution.

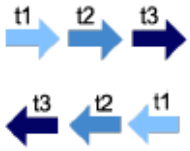



**Figure 2. 1: Backtrack reversibility [9]**

The second approach of reversibility is causal reversing and it allows a more flexible form of reversibility. Causal reversibility allows events to reverse in any order, assuming that they are independent. Thus, in causal reversing it is not necessary to reverse actions in the same order as they have occurred. Thereafter, as long as caused actions are reversed before the actions that have caused them, causal reversibility can undo actions in any order and be legitimate.

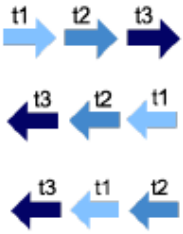



**Figure 2. 2 : Causal reversibility [9]**

Considering the figure above (Figure 2.2), since t1 occurs independently of action t2 we can reverse t1 and t2 in any order we want, although we can never reverse them before t3.

Both forms of backtracking and causal reversibility are cause-respecting, meaning that they respect the causal dependencies that exist between the events of a model. However, in many real-life examples, actions are reversed before their effects are undone. This is inherent in many real-life applications, such as biochemical reactions and long-running transactions. In such applications, the third form of reversibility that is called out-of-causal reversibility is used. Out-of-causal reversibility allows actions to be reversed before the actions that caused them are reversed. The main advantage of out-of-causal reversibility over the other forms of reversibility is that it gives the opportunity to reach new states that were formerly inaccessible by any forward-only execution path, or even with a backtrack or causal execution path. This happens because the causally-respecting forms of reversibility give only the chance to move backward and forward, through previously visited states only.

The figure below (Figure 2.3) depicts the possible out-of-causal executions for the same forward execution as in the previous Figures (2.1, 2.2). However with out-of-causal reversibility every possible sequence of executions is legitimate, because does not respect the causal dependencies between the events.

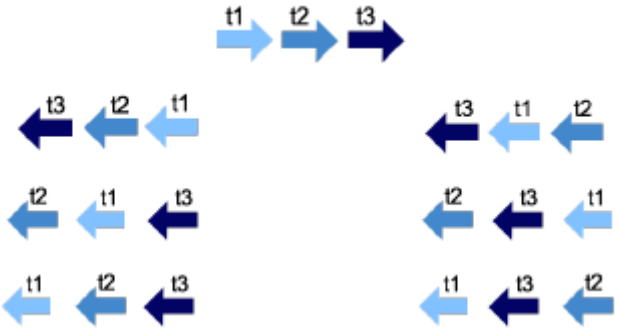



**Figure 2. 3: Out of causal reversibility [9]**

A standard example from biochemistry where out-of-causal reversibility is indispensable is the catalysis process.

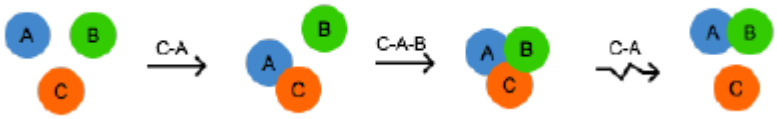



**Figure 2. 4: Catalysis reaction [9]**

The Figure above (2.4) shows the catalysis process where a catalyst *c* helps the otherwise inactive modules *a* and *b* to bond. So, initially element c bonds with a which enables the bonding between *a* and *b*. Then, the catalyst is no longer needed and its bond to the other two molecules is released. It can be observed that the last step of releasing catalyst *c* is an out-of-causal reversing action.

## 2.3 Petri nets

Petri nets [10] are a graphical mathematical language that is used for the specification and analysis of parallel and distributed systems, which were developed by Carl Adam Petri in 1962 as the subject of his dissertation. It is a class of discrete event dynamic system, which supports both action-based and state-based modelling, where the basic idea is to describe state changes in a system with transitions.

A Petri net model [4] is a structure that contains two kinds of elements. The first kind of elements is the node. A node can be either a *place*, or a *transition*. A place *p,* always models a passive component of the system, which should be the states, conditions, or resources that are necessary before an action can be carried out. Transitions always model an active component of the system, which is a certain action that may occur. The second element of a Petri Net model is *arcs*, which connect places and transitions to each other, and indicate the relation between the components, such as logical connections, access rights, or immediate linkings. The arcs addressed from a place to a transition describe that the places connected in that arc are pre-conditions for the corresponding transition, and are called incoming arcs. The arcs addressed from a transition to a place describe that the places connected in that arc are post-conditions of the corresponding transition, and are called outgoing arcs. Furthermore, arcs never run

between transitions or between places. Places are graphically represented by a circle; transitions are represented by a square/bar and arcs are represented by a directed arrow. Places may contain *tokens* that may move to other places by executing actions. A token on a place indicates the current state of the execution and the overall distribution of tokens across places is known as the marking M. The input state is indicated by the initial marking $M_{0.}$

The figure below (Figure 2.5) represents the graphical representation of the components that compose a Petri net.

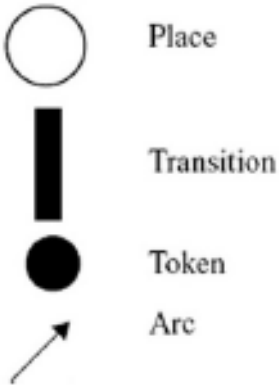



**Figure 2. 5: Components of a Petri net**

A transition is enabled for execution only when the number of tokens in each of its input places is at least equal to the number of arcs going to the transition. When a transition is fired, tokens from its pre-places are distributed to its output places, and a new marking of the net is created, where the transition that has been executed is no longer enabled, while others become enabled in that marking.

Thus, a net structure [9] is a directed, finite, weighted, – bipartite diagram specified as:

$$N= (P, T, F)$$

Where:

1. P is a finite set of places
2. T is a finite set of transitions such that $P \cap T= \emptyset$.
3. F is a set of arcs $F \subset (P \times T) \cup (T \times P)$.

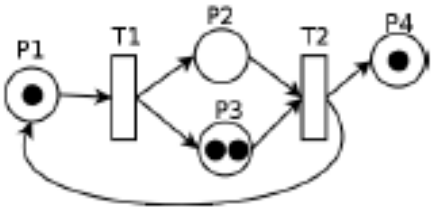



**Figure 2. 6: An example of a Petri net diagram**

Let us for instance consider the Petri net in Figure 2.6. It is composed of four places (P1, P2, P3, P4), and two transitions (T1, T2). Place P1 is filled with a token, as well as P4, while P3 is filled with two tokens. In this Petri net we have seven arcs from which the three of them are incoming arcs, and the other four are outgoing arcs. Moreover, T1 is an enabled transition, as all its input places are filled with tokens, while transition T2 is not enabled for firing because place P2, that is one of its input places is not filled with tokens.

The weight is graphically represented on the arcs as a label, and indicates the number of tokens that will be transferred through that arc.

Below there is an example of when a transition is enabled, and then the representation of the marking after the firing of the enabled-transition, showing how the tokens are distributed from the input places to the output places.

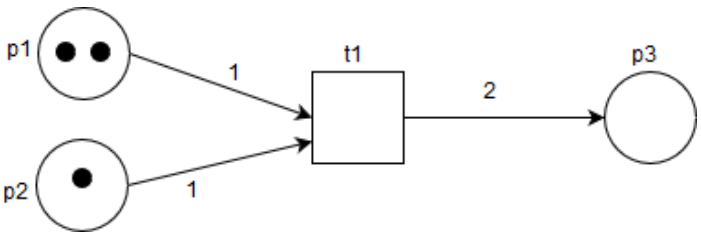



**Figure 2. 7: A Petri Net example where $t_1$ is an enabled transition**

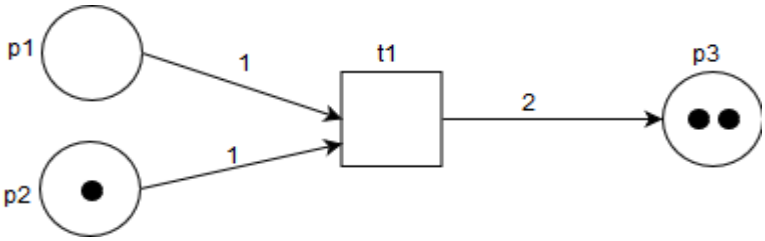



**Figure 2. 8 : A Petri Net example after transition $t_1$ is executed**

Petri nets can be applied to any area or system consisting of parallel or concurrent activities, as well as systems that can be described graphically like flow charts, due to their simple user interface, and generalit. Petri nets is an interesting conception because they can automatically determine a lot of information about a concurrent system, and this is the reason that they have been extensively used for modelling and reasoning in a wide range of applications.

## 2.4 Reversing Petri Nets

An approach that has been studied for Petri nets it is the reversibility in Petri Nets. This approach [8] introduces the concept of reversibility in the context of Petri Nets. Reversing Petri nets is an alternative model of the traditional Petri nets that deals with the three forms of reversibility, namely backtracking, causal and out-of-causal reversibility. Adding reversibility in Petri nets means that at any reverse execution, some tokens may return to the places that came from, while others not. So this property created the need to distinguish each token with an identity. In the context of reversing PNs there are two types of tokens, namely base and bond . A base is a persistent type of token which cannot be consumed, while a bond is a coalition of two bases. Specifically, the effect of a transition is the creation of new bonds between tokens or the transfer of already existing bonds/tokens along the places of a Petri net. Another feature of Reversing Petri nets is that incoming arcs are labeled in order to indicate the necessary tokens/bonds needed to fire the transition. If tokens/bonds exist in a place beyond those denoted to the incoming arcs, they do not necessarily need to exist to fire the transition. Reversing Petri Nets also introduce the notion of history in transitions, which is increased every time a transition is executed.

Thus, a Reversing Petri net (RPN) is defined as a tuple [8]:

$$N= (A, P, B, T, F)$$

Where:

1. A is a finite set of bases or tokens ranged over by a,b,... and $\overline{A} = \{ \overline{a} \mid a \in A\}$ contains a “negative” instance for each token and we write $\mathscr{A} = A \cup \overline{A}$ .
2. P is a finite set of places.

3. B ⊆A x A is a set of bonds ranged over by β, γ... We use the notation a−b for a bond (a, b) ∈ B. $\overline{B}$ = { $\overline{\beta}$ | β ∈ B} contains a "negative" instance for each bond and we write $\mathscr{B}$=B ∪ $\overline{B}$ .
4. T is a finite set of transitions.
5. F: (P x T ∪ T x P) →2 $^{\mathscr{A}\cup\mathscr{B}}$ is a set of directed arcs.

Places and transitions have the standard meaning as in Petri Nets, and they are graphically represented by circles and squares respectively. Bases are indicated by ● and bonds by lines between tokens. Furthermore arcs $\ell$ = F (p, t) or $\ell$ = F (t, p), which could be addressed from place to transition, or from transition to place, contain each token at most once, either the positive or the negative one. If a bond (a, b) ∈ $\ell$ then a, b ∈ $\ell$ and for $\ell$ = F (t, p) then the following holds $\ell$ ∩ (A ∪ B) = ∅. Thus, the labels placed on incoming arcs are expressing the requirements to fire a transition, and in the outgoing arcs are expressing the effects of the transition. For t ∈ T we write $^{\circ}$t = {x ∈ P | F(x, t) ≠ ∅} and t$^{\circ}$={x ∈ P | F (t, x) ≠ ∅} for the incoming and outgoing places of transition *t*, respectively. Furthermore, we write pre (t) =∪ $_{x\in P}$ F(x, t) for the union of all labels on the incoming arcs of transition t, and post (t) =∪ $_{x\in P}$ F (t, x) for the union of all labels on the outgoing arcs of transition t.

In the model of Reversing Petri nets a marking is a distribution of tokens and bonds across places, M: P → A ∪ B where if a-b ∈ M(x), for some x ∈ P, then a, b ∈ M(x). As mentioned above a history is assigned to each transition, H: T → ε ∪ ℕ. If a history of a transition is ε, it means that the transition has not been executed yet, or it has been reversed, otherwise any other value, n ∈ ℕ, indicates that the transition has been executed and not reversed yet, where n shows the order of execution among non-reversed actions. The initial history of a transition is denoted by $H_0$ and is equal to ε for all transitions of the initial marking. A state of a Petri net is described as a pair of a marking and a history <M, H>.

Any Reversing Petri net is called *well-formed* if it satisfies the following conditions for every transition t ∈ T:

1. A ∩ pre(t) = A ∩ post(t)
2. If a−b ∈ pre(t) , then a−b ∈ post(t),

3. $F(t, x) \cap F(t, y) = \emptyset$ for all $x, y \in P, x \neq y$.

The above conditions define that the transitions:

1. Do not erase tokens,
2. Do not destroy bonds, for instance if a-b exists in an input place of a transition, then after the execution of the transition it is maintained in some output place,
3. Tokens/bonds cannot be cloned into more than one outgoing places.

The last piece that is completing the definition of Reversing Petri nets machinery is con (a, C), where *a* is base and $C \subseteq A \cup B$ is the set of tokens and bonds connected with *a*.

$$con(a, C) = (\{a\} \cap C) \cup \{\beta, b, c \mid \exists w \text{ s.t. } path(a, w, C), \beta \in w, \text{ and } \beta = (b, c)\}$$

Where path (a, w, C) if $w = <\beta_1 \ldots, \beta n>$, and for all $1 \leq i \leq n$, $\beta i = (a_{i-1}, a_i) \in C \cap B$, $a_i \in C \cap A$, and $a_0 = a$.

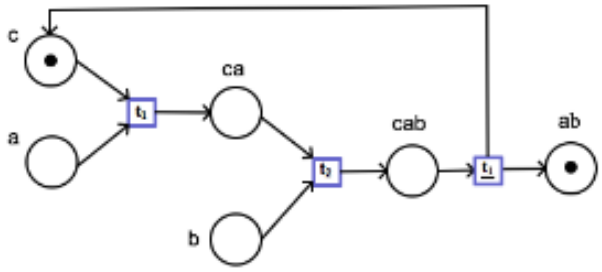



**Figure 2. 9: Catalysis example in classic Petri Nets.**

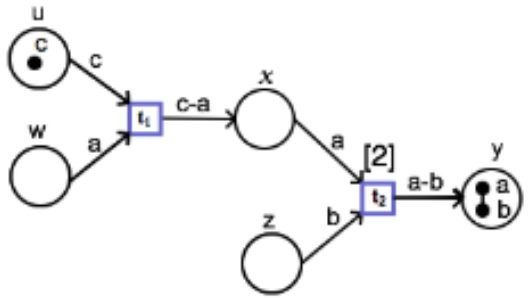



**Figure 2. 10: Catalysis example in Reversing Petri Nets.**

The figures above (Figure 2.9, 2.10) depicts the same example of catalysis from biochemistry in traditional Petri nets and in RPNs, respectively. We can observe the

changes from that have occurred in the RPN diagram. Firstly, in the cases that a token has represented a connection between two tokens, like for example *ca* in the first diagram, has been replaces in the second diagram with the insertion of bonds. Thus, any token that represented a connected components, now is represented with a bond. Moreover, in the RPN example the arcs are labelled, and there are no arcs with inverse direction, because in RPNs there is always feasible to execute transitions in reverse. Thus, the reverse execution of the transitions covers all possible arcs with inverse direction.

### 2.4.1 Forward execution

Considering a reversing Petri net (A, P, B, T, F), a transition *t*, and a state <M, H>, we say that *t* is forward enabled in <M, H> if the following hold:

1. If $a \in F(x, t)$, for some $x \in {}^\circ t$, then $a \in M(x)$, and if $\overline{a} \in F(x, t)$ for some $x \in {}^\circ t$, then $a \notin M(x)$,
2. If $\beta \in F(x, t)$, for some $x \in {}^\circ t$, then $\beta \in M(x)$, and if $\overline{\beta} \in F(x, t)$ for some $x \in {}^\circ t$, then $\beta \notin M(x)$,
3. If $a \in F(t, y_1)$ and $b \in F(t, y_2)$ where $y_1 \neq y_2$ then $b \notin con(a, M(x))$ where $x \in {}^\circ t$, and
4. If $\beta \in F(t, x)$ for some $x \in t^\circ$ and $\beta \in M(y)$ for some $y \in {}^\circ t$ then $\beta \in F(y, t)$.

So, a transition *t*, is forward enabled in state <M, H> if all tokens and bonds labelled on incoming arcs are available in the incoming places of *t*, and none of the tokens/bonds whose absence is required exists in any of the incoming places of *t*. Moreover, forks do not duplicate tokens and if a pre-existing bond appears in an outgoing arc of a transition, then it is also a precondition of the transition to fire.

Given a reversing Petri Net (A, P, B, T, F), a state <M, H>, and a transition *t* which is forward-enabled in <M, H>, we write $\langle M, H\rangle \xrightarrow{t} \langle M', H'\rangle$ where:

$$M'(x) = \begin{cases} M(x) - \bigcup_{a \in F(x, t)} con(a, M(x)), & \text{if } x \in {}^\circ t \\ M(x) \cup F(t, x) \cup \bigcup_{a \in F(t, x),\, y \in {}^\circ t} con(a, M(y)), & \text{if } x \in t^\circ \\ M(x), & \text{otherwise} \end{cases}$$

and

$$H'(t') = \begin{cases} \max\{k \mid k=H(t''), t'' \in T\} + 1, & \text{if } t'=t \\ H(t'), & \text{otherwise} \end{cases}$$

According to the definition, when a transition *t* is executed, all tokens, and bonds existing in its incoming arcs are transferred from the input places to the output places along with their connected components. Furthermore, history function H is extended to H' by assigning to transition *t* the next available integer key. [8]

### 2.4.2 Backtracking execution

Considering a reversing Petri net N = (A, P, B, T, F), a state <M, H> and a transition $t \in T$, we say that *t* is *bt-enabled* if it is the last executed transition, namely it has the highest H value.

When we reverse a transition with a backtracking fashion in a reversing Petri net, the effect is as follows:

Given a reversing Petri Net (A, P, B, T, F), a state <M, H>, and a transition *t* which is b-enabled in <M, H>, we write $\langle M, H\rangle \overset{t}{\underset{b}{\rightsquigarrow}} \langle M', H'\rangle$ where:

$$M'(x) = \begin{cases} M(x) \cup \bigcup_{y \in t^\circ, a \in F(x,t) \cap F(t,y)} con(a, M(y) - eff(t)), & \text{if } x \in {}^\circ t \\ M(x) - \bigcup_{a \in F(t,x)} con(a, M(x)), & \text{if } x \in t^\circ \\ M(x), & \text{otherwise} \end{cases}$$

and

$$H'(t') = \begin{cases} \varepsilon, & \text{if } t'=t \\ H(t), & \text{otherwise} \end{cases}$$

So, after reversing transition t, all tokens, and bonds in the outgoing places of t, as well as their connected components will be transferred to the incoming places of the transition. In addition any newly-created bonds will be broken and the history of t will be refined to 0. [8]

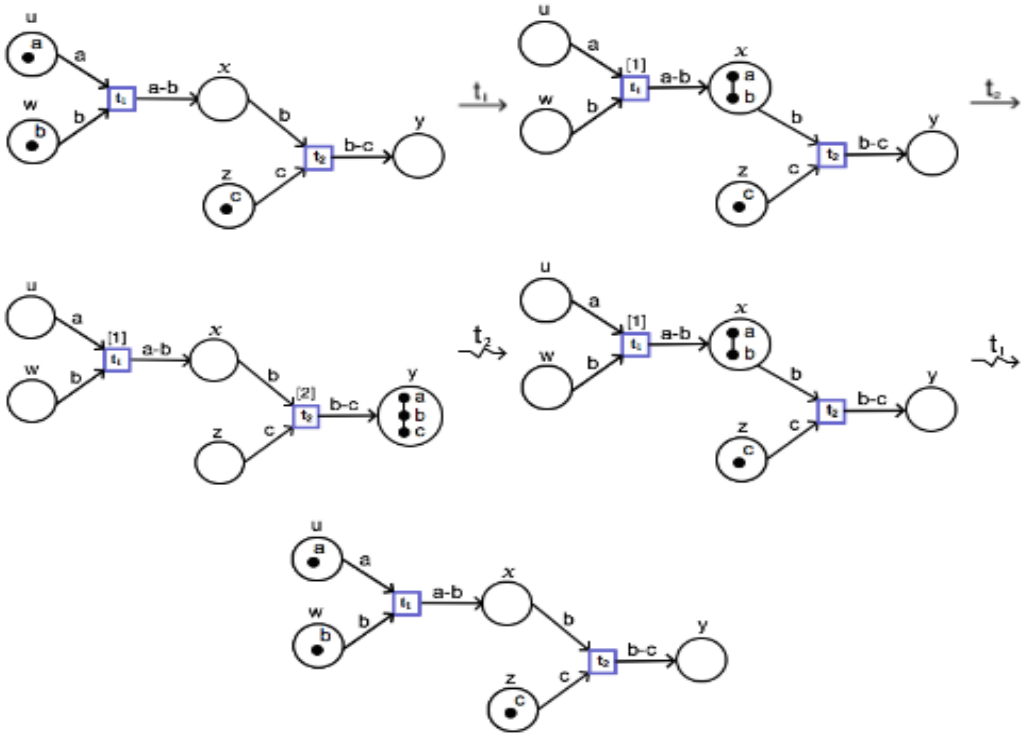



**Figure 2. 11: An example of Forward and Backtracking execution**

The figure above (Figure 2.11) represents a sequence of forward executions in the following order: t1 -> t2 -> t3, and then the backtrack execution of these transitions, which is the inverse execution of the forward execution, according to the backracking rules.

### 2.4.3 Causal execution

Consider a reversing Petri net N= (A, P, B, T, F), a state <M, H> of N, and a transition t ∈ T, we say that t is co-enabled in <M, H> if all its effects are reversed, or not executed yet, i.e. all tokens labelled in the outgoing arcs of the transition, are available in the output places of it.

After reversing a transition t, with causal-order fashion, the effects are exactly the same as in backtracking.

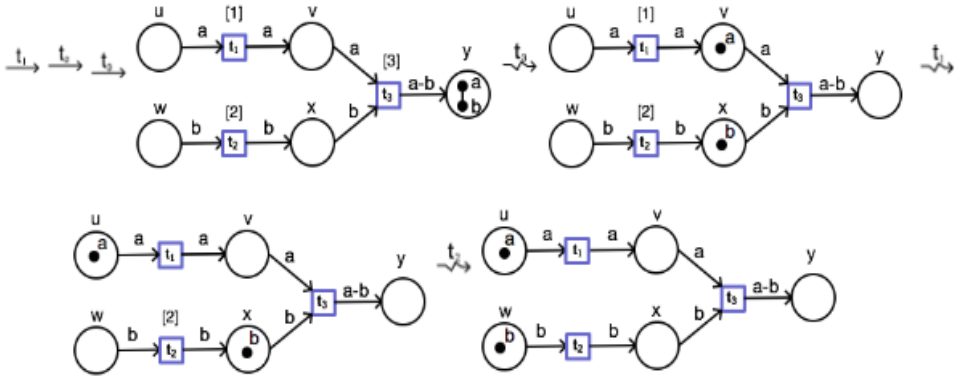



**Figure 2. 12: An example of Causal-order execution**

The figure above (Figure 2.12) shows one possible causal execution, after the forward execution with the following order: t1 -> t2 -> t3. We can observe that the transitions t2 and t1 are independent because there are no dependencies for their executions, and we can reverse them with any order, but not before the transition t3. We have to reverse the transition t3 first, because t3 is depended on the t1's ans t2's execution, so the transitions t1 and t2 need the reversed effects of t3 to be able to reversed.

### 2.4.4 Out-of-causal-order execution

Consider a reversing Petri net N= (A, P, B, T, F), a state <M, H> of N, and a transition t ∈ T, we say that t is o-enabled in <M, H> if it has been executed before. So, all transitions with history H (t) ≠ ε are o-enabled.

The effect of reversing a transition in an out-of-causal fashion is that all bonds created by the transition are destroyed. If the destruction of a bond divides a component into smaller components, then those components should flow back, as far back as possible, to the last transition in which they participated.

The notion "as far back as possible" is defined below:

Given RPN N= (A, P, B, T, F), an initial marking $M_0$, a current marking M, a history H, and a set of bases and bonds C we write:

$$\text{last}(C, H) = \begin{cases} t, & \text{If } \exists_t \text{ post}(t) \cap C \neq \emptyset,\ H(t) \in \mathbb{N} \\ & \nexists_{t'} \text{ post}(t') \cap C \neq \emptyset,\ H(t') \in \mathbb{N},\ H(t') > H(t) \\ \perp, & \text{otherwise} \end{cases}$$

Thus, last(C, H) is defined as follows: If the component C has been manipulated by some previously-executed transition, then last(C, H) is the last executed such transition. Otherwise, if no such transition exists (e.g. because all transitions involving C have been reversed), then last(C, H) is undefined (⊥). Transition reversal in an out-of-causal order can thus be defined as follows:

Given a RPN (A, P, B, T, F), an initial marking $M_0$, a state ⟨M, H⟩ and a transition that I so-enabled in ⟨M, H⟩ , we write $\langle M, H\rangle \overset{t}{\underset{o}{\rightsquigarrow}} \langle M', H'\rangle$ where H ′is defined as in backtracking execution and we have:

$$M'(x)= M(x)-eff(t)-\{C_{a,x} \mid \exists a \in M(x), x \in t'^{\circ}, t' \neq last(C_{a,x}, H')\}$$
$$\cup\{C_{a,y} \mid \exists a, y, a \in M(y), last(Ca, y, H') = t', F(t', x) \cap C_{a,y} \neq \emptyset\}$$
$$\cup\{C_{a,y} \mid \exists a, y, a \in M(y), last(Ca, y, H') = \perp, C_{a,y} \subseteq M_0(x)\}$$

where we use the shorthand $C_{b,z}$ = con (b, M (z) –eff (t)) for b ∈ A, z ∈ P, and eff(t) for the effects that has been created from the execution of transition t.

Thus, according to the above definitions, if the destruction of a bond divides a component into smaller connected components then each of these components should be relocated back to the outgoing places of their last transition, i.e. to the places that should exist if transition t never took place, or to the places in their initial marking.[8]

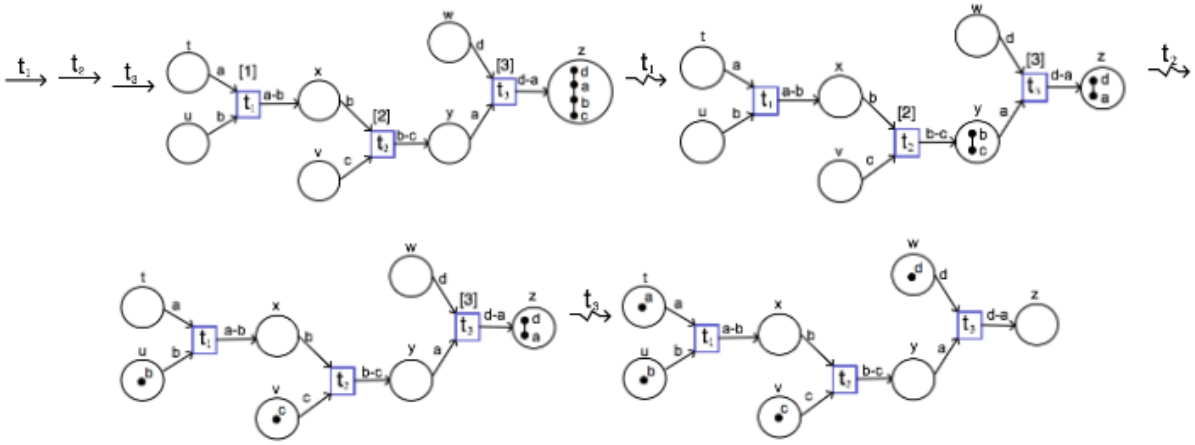


**Figure 2. 13: An example of Out-of-causal-order execution**

The figure above (Figure 2.13) depicts an out-of-causal execution, after a forward execution with the following order: t1-> t2 -> t3. We can observe that in out-of-causal reversibility it is possible to reverse a transition before we reverse the effects of this transition. For instance in the figure above the transition t1 is reversed before its effects.

# Chapter 3

## Requiremnts specification and Software used



Since there is no software that simulates the behavior of Reversing Petri nets, after the requirements specification, the software and programming languages that would be used for the implementation of that simulator should be decided.

## 3.1 Requirements specification

The first phase of this project was the specification of the requirements.

The requirements specification is a key process of any well organized and coherent project, because requirements will drive the software design and implementation process. The software requirements specification lays out functional and non-functional requirements, which will define the functions that the software is supposed to provide and the behavior of the software in several situations.

### 3.1.1 Aims

The aim of this Diploma thesis project is the development of a tool that will simulate the theoretical semantics of Reversing Petri nets, in order to find practical application in

user-friendly software. The software will give the opportunity to users to give a reversing Petri net as text file for input or define it from the interface, as well as to draw a diagram for textual RPN representation. In addition it will provide choices for finding enabled transitions, and execute them, based on user choices and on the implemented algorithms. Then users will be able to see the changes in the marking, after the execution, and if the graphical representation was chosen, the changes will be available at the diagram as well.

### 3.1.2 Objectives

In order to apply the principles of Reversing Petri nets in practice and give to users the opportunity to simulate their behavior on software, the following objectives have to be met:

i. To create an Object Oriented approach which will translate the Reversing Petri nets' information, and then from that form into the corresponding algorithms.
ii. To prepare the steps of the software development by identifying the technical and engineering background of the simulator.
iii. To implement the algorithms in the Java programming language
iv. To construct a Graphical User Interface that will allow users to interact with the system to understand Reversing Petri nets and observe forward and reverse execution of a chosen transition. Then, connect the Graphical User Interface with the algorithms implementation to simulate the behavior of the algorithms in a marking.
v. To develop a Graphical representation tool for Reversing Petri nets to model them and give to users to choice to create a RPN diagram and give it as input in the simulator.
vi. To translate the Petri net's information from the Graphical representation tool into Java code, and be able to give it as input for the algorithms.
vii. To connect the Graphical representation tool with the Graphical User Interface, so users can see the changes in a marking, after the execution of a transition directly on the diagram, as well as on the GUI.

### 3.1.3 Specifications

The project objectives must be achieved by the implementation of specific steps. Firstly, a way to express Reversing Petri net information in an Object oriented approach in the simulator environment has to be set. Thence, the four mechanisms that have been defined for RPNs must be implemented, based on the object oriented approach that has been defined before. The key consideration of the implementations at first will be correctness, and at a subsequent stage memory resources and efficiency will be under consideration.

Generally, we have to find the most efficient structure for the Reversing petri nets representation, in order to easily change tokens or bonds within a set, with the minimum memory resources and time.

Moreover, the graphical representation tool has to be simple in use for any user, and easily accessible from the simulator environment. Thus, a way to connect directly the simulator environment with the graphical representation tool has to be set.

## 3.2 The Java programming language

The developed software system simulates both forward and reverse direction execution in Reversing Petri nets, in the Java programming language. Java is a Class-based, object oriented language that allows programmers to develop user friendly interfaces, with the use of Swing API. In addition Java has designed to have as few implementation dependencies as possible, and thus Java code runs on all platforms that support Java without the need for recompilation. Moreover Java [6] provides some features that manage automatically memory space, so temporary results, and variables are not a concern for the programmer anymore. After a comparison with other programming languages, Java was considered the most appropriate, because of the features that it provides, and alongside the opportunity to develop a friendly user interface, with the use of Swing API, which would facilitate users to understand the functions of the simulator quickly and use it without much effort. Moreover, Java’s object oriented feature, is appropriate, in order to create a flexible, space-free approach, for Reversible Petri Nets.

### 3.2.1 Object-oriented approach of Reversible Petri Nets

Having in mind that the simulator must execute commands rapidly, and use as less as space possible, the design of an Object-oriented approach of RPNs was necessary, at the first steps of this diploma thesis.

Every element of a RPN had to be represented by an object, in order to be independent and relationships could be described between the objects. So, transition, place, token/bond, and arcs are represented by an object in Java, respectively.

Each of these objects includes features that compose each element of a RPN. Every object is characterized of a unique number, the ID, which defines the identity of the object. Further field of each object are the name, a list of tokens/bonds in the case of places and arcs, a variable that determines whether a transition is forward-enabled, bt-enabled, co-enabled or o-enabled.

Some objects have to have a relationship with other objects, such as arcs. Arcs must contain fields that determine the place and the transition that make up the arc. Also further fields that are essential for the arc-object are fields that define the direction of the arc (‘from’, ‘to’).

At that stage we had the representation of the basic elements that compose a RPN in Java, and the relationships between them. The next step was to consider how all these elements would be connected and synthesize the object of the RPN. Another object had to be added in our approach, the object of the Petri net. Petri net’s object contains a list for each item, namely places, transitions, arcs, and tokens/bonds. Furthermore there is an instance of the last executed transition. Another element that had to take into account was the History. In order to represent the history in Java, an extra object was created. This object, named ‘Cell’, includes two fields, an instance of a transition and the history value of each transition. Thus, Petri net object comprises a list of ‘Cells’, with size equal to the number of transitions in the RPN, and it represents the history of each transition in the RPN.

Then, we had a complete object-oriented approach for RPNs, and we could continue with the implementation of that approach in Java, in order to simulate the algorithms.

## 3.3 Obeo Designer software

Obeo Designer software has been used for the development of the graphical representation tool for Reversible Petri nets, because it has features that were needed, according to the requirements of graphical representation tool.

Obeo Designer [1] is plug-in software of Eclipse Foundation, which provides a way to designers to create and edit a domain model first, corresponding to their needs, and then gradually creates a modelling workbench, which will handle the actions for the representations of the domain model. After the completion of all steps, a modelling tool will be developed.

In order to create the modelling tool, there is a need of a class diagram, which will define how data of the domain model will be saved. That class diagram, is an Ecore model at Obeo Designer, and works like the metamodel of the project.

The next step is to use the Ecore model that has been created, in order to generate the code of the project, which is generated automated by the tool. Afterwards, there is the development of a design approach of the domain model, and a workbench to give the opportunity to users to create and/or edit their representations.

Generally, the developer defines the architecture of the model, and the software generates the code automatically.

Obeo Designer uses Eclipse Sirius [1], to enable users graphically design complex systems, while keeping the corresponding data consistent. Eclipse Sirius is an Open Source Eclipse software project, which is integrated into annual versions on the Eclipse platform. Sirius technology allows users to create custom graphical modelling workbenches by leveraging the Eclipse Modelling technologies of EMF and GMF. The modeling workbench that will be created will be composed of a set of Eclipse editors, such as diagrams, trees, and tables.

## 3.4 Unified Modeling Language (UML)

Before the implementation of the algorithms began in Java, a Component diagram had to be designed, in order to create a clear picture of the functions of the simulator and the connections with the Graphical representation tool. Furthermore, in order to develop the tool for the graphical representation of the Petri net, it was essential a Class diagram, as a metamodel of the system. The most appropriate language to create these diagrams is Unified Modeling Language, known as UML. UML is a general-purpose language for

specifying, visualizing, constructing and documenting the artifacts of software systems. It is adapted as standard by Object Management Group (OMG), and constitutes an approved ISO standard. UML provides different types of diagrams, so they can visualize a system's architectural blueprints, such as any activities of the system, individual components and how they can interact with the system, how entities of the system interact with each other, the design of user interface, and how the system will run. Most of the times, in order to have a complete view of a system several diagrams are needed, in the design phase. UML diagrams are divided in two groups based on the view of the system model that they are representing.

The first category of UML diagrams [11] is static or structural diagrams, which they emphasize in the static structure of a system, and not model any dynamic behavior of the system. Since they are representing the structure, they are used extensively in documenting the software architecture of software systems. This kind of diagrams is composed of objects, attributes, operations, and relationships. Secondly, there are the dynamic or behavioral diagrams [11], which they emphasize in the dynamic behavior of a system, by showing collaborations among objects and the changes to internal states of objects. They are used to describe the functionality of software systems

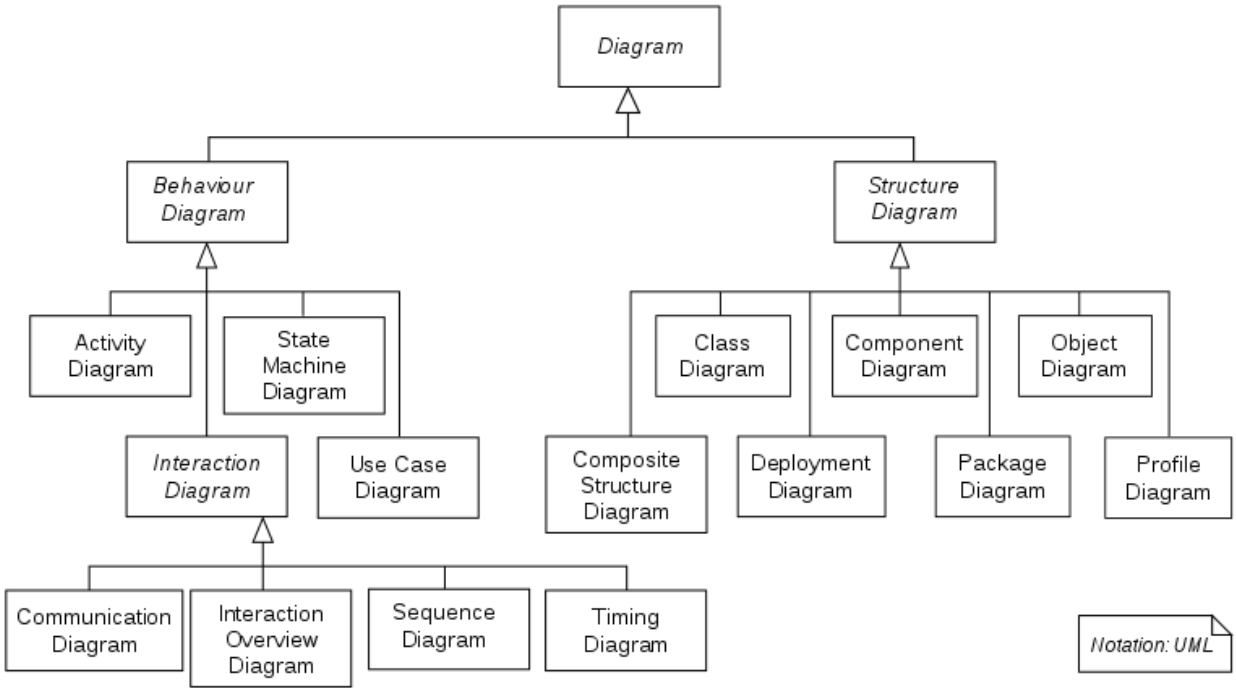



**Figure 3. 1: UML Diagrams overview**

The figure above (Figure 3.1) displays the overview of UML diagrams, separated in the two main categories, and then in subcategories.

For the needs of this diploma thesis, two structure diagrams were necessary; a component diagram and a class diagram. The component diagram was created to describe the components of the system and the dependencies among them. Afterwards, in the development of the graphical representation tool a Class diagram was essential, to operate as a metamodel of the tool.

### 3.4.1 Component diagram

Component diagrams [11] are used to depict how components of a system are wired together to form larger components or software systems, and they are used to illustrate the general architecture of arbitrary complex systems. These diagrams consist of squares, which represents the components, half and whole circles which represent the required and provided interfaces for components respectively, and interrupted lines that represent dependencies between components.

### 3.4.2 Component diagram for Reversing Petri nets

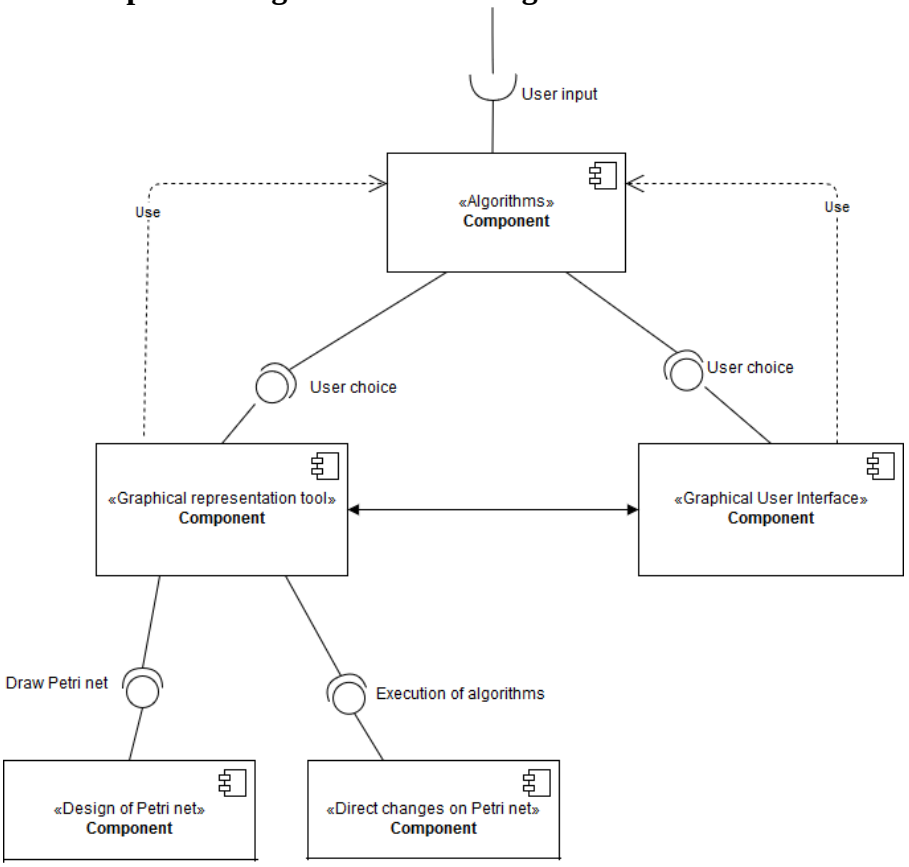



**Figure 3. 2: Component diagram for Simulator for RPNs**

The diagram above (Figure 3.2) represents the Component diagram of the Simulator for Reversible Petri nets system. There are three main components in the systems, namely Algorithms, Graphical User interface, and Graphical representation tool. “Algorithms” component indicates the part of the software that simulates the behavior of the four algorithms. “Graphical representation tool” component represents the graphical representation software of the simulator, which is connected with the algorithms part, and finally “Graphical User Interface” represents the interface of the simulator, that is connected with the “Algorithms” component too.

Algorithms need the user input, in order to function, according to the diagram. Moreover, “Graphical User Interface” and “Graphical representation tool” are provided interfaces of the “Algorithms” component, because when algorithms have the user input, they can give the opportunity to user to choose graphical representation as input or give the input from the graphical user interface.

Based on the user choice either Graphical User Interface” or “Graphical representation tool” component can function. “Graphical representation tool” component has two more provided interfaces, namely “Design of Petri net” and “Direct changes on Petri net.” “Design of Petri net” component indicates the function of tool that allows the user to draw the Petri net diagram. In addition “Direct changes on Petri net” component represents the function of the tool that shows to the user the changes on the Petri net diagram, based on the execution of the algorithms.

The Graphical User Interface” and the “Graphical representation tool” are associated with each other because the user executes the desired transition from GUI and the changes can be shown on the graphical representation directly.

### 3.4.3 Class diagram

A Class diagram [11] is a type of static structure diagram which depicts the structure of a system, by showing the system classes. Classes consist of attributes and methods, which represents the possible operation of the class. In a class diagram classes have relationships among them.

### 3.4.4 Class diagram for Reversing Petri nets

At the first steps of the development of the graphical representation tool there was a need for a class diagram. Class diagram indicates the components of a reversible petri net and the relationships among them, so when a user draws the petri net only permissible relationships would be available. Moreover, class diagram will determine the components of a RPN, and the attributes of each component, based on the relationships of the diagram.

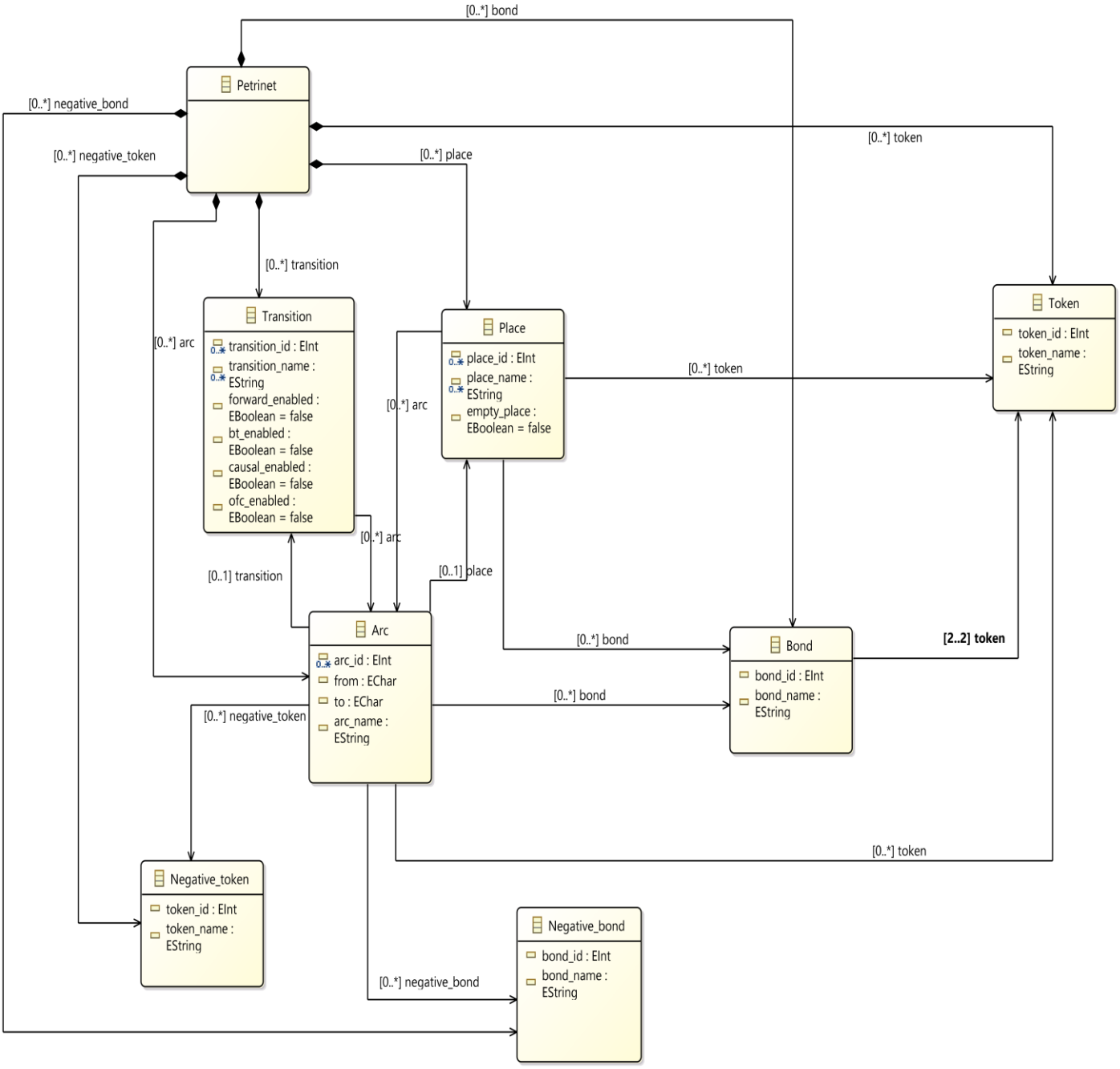



**Figure 3. 3: Class diagram for Simulator for RPNs**

The Class diagram above (Figure 3.3) shows the class diagram for the Reversible Petri net model. There is a central class that represents the Petri net class, which is composed of all other components of a reversing Petri net, namely place, transition, arc, token, bond, negative token, and negative bond. An arc consists of a place and a transition, and can be either from a place to transition or from a transition to place. It can be observed that a place can have several tokens and/or several bonds, while a bond consists of exactly two tokens. An arc can be labeled with a set of bonds, and/or a set of tokens, and/or a set of negative tokens, and/or a set of negative bonds, and there are two fields that indicate whether the arc comes from a transiton or a place. Moreover all elements can be identified from a name and an id number that are unique for its instance. Transition class consists of four boolean fields that indicate if a transition is forward-enabled, backtracking-enabled, causal-enabled and/or out-of-causal enabled, respectively.

# Chapter 4

## Simulator

---



---

## 4.1 Representation of basic components

After defining the most appropriate programming language for the implementation of the simulator, a research on UML and the creation of the necessary diagrams for the project, which are a component and a class diagram, as well as the specification of the

project requirements, the next phase was to begin the actual implementation of the simulator, in the Java programming language.

The initial step of the implementation was the representation of all basic components that constitute the Reversing Petri nets, in the programming language environment. Thus, based on the object oriented approach described in subchapter 3.1.1, each component is depicted from a Java object, with its own fields.

### 4.1.1 Place

Places is one of the main components in Reversing Petri nets, and based on the object oriented approach that has been created, it will be composed of the following fields: place id, place name , which are unique for each place, a list with the possible tokens and a list with the possible bonds held in the place. Moreover it will have a field that will indicate if the place is empty, and a list that will indicate with which arcs the place is connected. The ArrayList structure was selected for the representation of lists in Java. The figure below (Figure 4.1) shows what a Place object contains in Java and how these fields are initialized, when a Place object is created.

```
public class Place {
    String place_name;
    int place_id;
    ArrayList<Token> tokens;
    ArrayList<Token>bonds;
    ArrayList<Integer> arc_id;
    boolean empty;

Place(){
    this.place_name="";
    this.place_id=0;
    this.tokens=new ArrayList<Token>();
    this.bonds=new ArrayList<Token>();
    this.empty=false;
    this.arc_id=new ArrayList<Integer>();
}

}
```

**Figure 4. 1: Place representation in Java**

### 4.1.2 Transition

The other type of node in Reversing Petri nets are transitions, which describe the actions either in forward or reverse order that can take place in a Petri net. Transition is another object in Java representation, which consists of transition id and transition name, which are unique for each instance, four Boolean variables which indicate whether a transition is forward enabled, bt-enabled, co-enabled and/or o-enabled. Furthermore, the transition's object has a list with the arc ids that a transition is connected with, and two variables that tells the number of input arcs and the number of output arcs for that transition.

The figure below (Figure 4.2) shows the fields that compose a transition object and how they are initialized, when a transition object is created.

```
public class Transition {
String transition_name;
int transition_id;
boolean enabled_for_execution;
boolean backtrack_enable;
boolean co_enabled;
boolean o_enabled;
int num_of_input;
int num_of_output;
ArrayList<Integer> arc_id;


Transition(){
    this.transition_name="";
    this.transition_id=0;
    this.enabled_for_execution=false;
    this.backtrack_enable=false;
    this.co_enabled=false;
    this.num_of_input=0;
    this.num_of_output=0;
    this.o_enabled=false;
    this.arc_id=new ArrayList<Integer>();

}
}
```

**Figure 4. 2: Transition representation in Java**

### 4.1.3 Arc

An arc is the connection between a place and a transition, or a transition and a place in a reversing Petri net, and describes the transfer of tokens and/or bonds through places, when a transition is fired. An arc object contains an instance of the place and of the transition that are connected and two variables that point out if it is an arc from place to transition, or an arc from transition to place. In addition it includes four lists which comprise the tokens and/or bonds required for that arc, in order to fire the transition, and/or the absence of tokens/bonds, in order to fire the corresponding transition (pre-conditions), if it is an arc from place to transition. On the other hand, if it is an arc from transition to place these lists comprise the tokens and/or bonds that the transition has as effect after its execution (post-conditions).

The figure below (Figure 4.3) shows the elements of an arc object in Java, and how these elements are initialized when an Arc object is created.

```
public class Connection {
    int connection_id;
    Place place;
    Transition transition;
    ArrayList <Token>tokens;
    ArrayList <Token> bonds;
    ArrayList<Token> negative_tokens;
    ArrayList<Token> negative_bonds;
    char from;
    char to;

Connection(){
    this.connection_id=0;
    this.place=new Place();
    this.transition=new Transition();
    this.tokens=new ArrayList<Token>();
    this.from='\0';
    this.to='\0';
    this.bonds=new ArrayList<Token>();
    this.negative_bonds=new ArrayList<Token>();
    this.negative_tokens=new ArrayList<Token>();

}
}
```

**Figure 4. 3: Arc representation in Java**

### 4.1.4 Token

Tokens are the elements that are distributed in places on a reversing Petri net, and indicate the pre- and post-conditions of a transition. A token can be a base or bond, which is a connection between two bases. So, in the Java representation, the token

structure describes either a base or a bond, and contains token id and token name fields, which are unique for each token.

The figure below (Figure 4.4) shows the token's fields in Java and their initialization, when an instance of that object is created.

```
public class Token {
    String name;
    int id=0;
    public static int e=0;

Token(){
this.name="";
e++;
id=e;

}
}
```

**Figure 4. 4: Token representation in Java**

### 4.1.5 History

History is a piece of data for each transition that indicates whether the corresponding transition has been reversed, or has been forwardly executed and in what order. Thus, in order to represent the history for each transition a new structure had been created in Java composed of the transition and the history value. The new structure is called Cell, and there is a Cell instance for each transition in the given Reversing Petri net, saved in an ArrayList.

The figure below (Figure 4.5) shows the Cell structure in Java and its fields.

```
public class Cell {
    Transition tr;
    int history;

    Cell() {
        this.history = 0;
    }
}
```

**Figure 4. 5: Cell structure in Java, which represents history for each transition**

### 4.1.6 Petri net

After the creation of all basic components of RPNs in Java, the next step was to create an object, which contains all other elements and represents the reversing Petri net. So, the PetriNet structure was created that is composed of ArrayLists for places, transitions, tokens, arcs, and history of the reversing Petri net. In addition it contains an instance of the last executed transition.

The figure below (Figure 4.6) shows the Petri net structure in Java, and the fields from which it is composed.

```
public class PetriNet {
    ArrayList<Place> places;
    ArrayList<Transition> transitions;
    ArrayList<Connection> arcs;
    ArrayList<Token> tokens;
    ArrayList<Cell> history;
    Transition last_executed;

    PetriNet() {
        places = new ArrayList<Place>();
        transitions = new ArrayList<Transition>();
        arcs = new ArrayList<Connection>();
        tokens = new ArrayList<Token>();
        history = new ArrayList<Cell>();
        last_executed = new Transition();
    }

}
```

**Figure 4. 6: Petri net representation in Java**

## 4.2 Algorithms

Since all structures were created in Java, the four algorithms that can take place in Reversing Petri nets, namely Forward, Backtrack, Causal-order, and Out-of-causal-order, had to be implemented in Java, using the structures mentioned above.

### 4.2.1 Forward algorithm

Forward execution can take place when all pre-conditions of a transition are met, namely all tokens and/or bonds exist in the required places and all the tokens, and/or bonds whose absence is necessary do not exist in any corresponding place. When all pre-conditions are met, the transition is forward-enabled and can be fired. After the execution of the transition tokens and/or bonds are distributed to the corresponding output places of the transition, or new bonds are created and then distributed to the output places. Thus, for the forward algorithm of RPNs, there were implemented two methods, and specifically the forward_enabled and forward_execution method.

**Forward-enabled method**

**Algorithm 1:** Forward-enabled method

```
Input: The method takes a Reversible Petri net as attribute
Result: Find all forward-enabled transitions for a given Petri net
initialization;
foreach transition t_∈ Petrinet do
    foreach arc a_∈ Petrinet do
        if t ==a.transition and a.to == "t" then
            toTransitionArcs add a;
        end
    end
end
foreach arc a_∈ toTransitionArcs do
    if a has bonds as preConditions then
        foreach place p_∈ InputArcsOfTransition do
            check if preCondition bonds exist;
        end
    end
    if a has tokens as preConditions then
        foreach place p_∈ InputArcsOfTransition do
            check if preCondition tokens exist in the corresponding input places ;
        end
    end
    if a has negative tokens as preConditions then
        foreach place p_∈ InputArcsOfTransition do
            check if preCondition negative tokens do not exist in the corresponding input
              places;
        end
    end
    if a has negative bonds as preConditions then
        foreach place p_∈ InputArcsOfTransition do
            check if preCondition negative bonds do not exist in the corresponding input
              places;
        end
    end
    if all preConditions are met then
        a.transition.forwardEnabled=true;
    end
end
```

**Figure 4. 7: Forward-enabled method written in pseudocode.**

The figure above (Figure 4.7) represents the actions that the forward-enabled method does. Firstly, for a given RPN it finds all the arcs of the transition, which they are arcs from place to transition and it saves them in a list. The list in Java is represented by ArrayList structure, so *toTransitionArc* is an ArrayList that contains <Arc> instances. Thereafter, it takes each of these arcs and checks its pre-conditions. If pre-conditions consist of tokens, it checks all input places of that arcs, and find out if the required tokens hold in the corresponding places. Similarly, behaves if pre-conditions contain bonds. In the case that pre-conditions contain negative tokens or bonds, it means that the absence of these tokens/bonds is required in the corresponding places, so a for loop is done in the input places of the arcs, and checks if these tokens/bonds do not exist in the input places, respectively. Finally, if all conditions are met, namely all tokens/bonds that must exist are in the right input places, and all token/bonds that must not exist are not in any input place, the transition of that arc is identified as forward-enabled.

**Forward execution method**

**Algorithm 2** Forward-execution method

**Input:** The method takes a specific transition t of a Petri net as attribute.
**Result:** If the transition is forward enabled,it is executed.

```
find inputArcs for transition t and save them to a list();
find outputArcs for transition t and save them to a list();
find inputPlaces for transition t and save them to a list();
find outputPlaces for transition t and save them to a list();

if t is forwardEnabled then
    foreach outArc outa_∈outArcs do
        foreach element e_∈outa do
            if e is token then
                tranfer e from corresponding inputPlace to corressponding outputPlace();
            else
                if e is bond and exists in an inputPlace then
                    tranfer e to corressponding outputPlace();
                else
                    if e is bond and does not exist to an inputPlace then
                        createTheBond b();
                        tranfer b to corressponding outputPlace();
                    end
                end
            end
            if e is a token and there is a bond or connected component c in inputPlace then
                con=createTheBond b() between e and the right base of c;
                transfer con to corresponding outputPlace();
            end
        end
    end
    t.historyValue=lastExecutedTransition.historyValue+1;
end
```

**Figure 4. 8: Forward-execution method written in pseudocode**

The figure above (Figure 4.8) displays the forward-execution method, written in pseudocode. The idea is to first find all input and output places of the transition we desire to execute, as well as all input and output arcs of it. Then, we check if the transition is forward_enabled, and in case it is, we pass one by one all output arcs. For each output arc we check its label, and if it is a token, we transfer the token to the corresponding output place, or if it is a bond, and does not exist to an input place we create the bond and transfer it to the corresponding output place. Otherwise, if the label of the output arc is a bond, which exists in an input place, we just transfer to the output place. If the element is a token, and has to be connected with a component, which is already connected, a bond will be created between e and the right base of the component. Finally, we assign the history value of the transition to the history value of previous last executed transition plus 1.

### 4.2.2 Backtrack algorithm

Backtrack-order execution can take place, only to the transition in the Petri net that has the biggest history value, that is the one that has been executed last, and the history value is not ε. If a bt-enabled transition exists in the Petri net, then backtrack execution can happen. When backtrack execution takes place the tokens labeled in output arcs are relocated back from output places to their corresponding input places, and any newly created bonds break, and then their bases are relocated back to their corresponding input places.

**Backtrack-enabled method**

**Algorithm 3** Backtrack-enabled method

**Input:** The method takes a Reversible Petri net as attribute

**Result:** Find the backtrack-enabled transition for a given Petri net,because at any time we can have only one bt-enabled transition, the one with the biggest history value.

```
int max=0;
 Cell maxCell=new Cell();
 foreach historyInstance h_∈PetriNet do
    if h.historyValue > max then
       max=h.historyValue;
        maxCell=h;
    end
end
if maxCell != null then
    foreach transition t_∈PetriNet do
       if t==maxCell.tr then
          t.btEnabled=true;
       end
    end
end
```

**Figure 4. 9: Backtrack-enabled method written in pseudocode**

The figure above (Figure 4.9) represents the backtrack-enabled method written in pseudocode. Firstly, for a given Reversible Petri net, and for each history instance in that Petri net, it finds the bigger history value, and saves it to a variable, as well as and the Cell instance that holds this value, with its transition instance. Afterwards, if a maximum value is found, it passes from all the transitions in the Reversible Petri net, and compare it with the transition of the maxCell. When it finds the transition that matches, it means that this is the transition with the biggest history value, and it assigns its bt_enabled field to true.

**Backtrack execution method**

**Algorithm 4** Backtrack-execution method

**Input:** The method takes a Reversible Petri net as attribute.

**Result:** If there is transition t that is backtrack-enabled,it is executed in reverse order, using backtracking algorithm's rules.

```
find inputArcs for transition t and save them to a list();
find outputArcs for transition t and save them to a list();
find inputPlaces for transition t and save them to a list();
find outputPlaces for transition t and save them to a list();

if t is btEnabled then
    foreach outArc outa∈outArcs do
        foreach element e∈outa do
            if e is token then
                tranfer e from corresponding outputPlace to corressponding inputPlace ();
            end
            if e is newly created bond b then
                breakTheBond b;
                relocate elements of b in corresponding inputPlaces(); /*elements of b, can
                be either tokens or other bonds.*/
                if e is bond, and bases from that bond are connected with other bases then
                    breakTheConnectionof(e); if there are connected components that they are
                    consists of a base of e then
                        relocate connected component in the corresponding inputPlaces(); re-
                        locate the base left from e in corresponding inputPlaces();
                    end
                end
            end
        end
    end
    t.historyValue=0;
end
```

**Figure 4. 10: Backtrack execution method written in pseudocode.**

The figure above (Figure 4.10) shows the backtrack execution method, written in pseudocode. Firstly, it finds all input and output places and input and output arcs of transition t, and they are saved in four tables respectively. Then for each output arc, and for each element of that output arc, it relocates the elements from the output places to their corresponding input places, if the element is token. If the element is a newly created bond, it breaks the connection between the bases, and then relocates its elements back to their corresponding input places. The elements of a bond can be either bases or other tokens, and in both cases they are transferred back to their input places. Finally, it sets history value of the transition to 0, because it has been reversed.

### 4.2.3 Causal-order algorithm

Causal-order reversibility enables the execution of a transition in different order than it was forward executed, a feature that backtracking reversibility does not have. Although, in order for a transition to be co-enabled, its history must be any value different than ε that is any number bigger than zero, and all tokens and/or bonds identified in its output arcs, are present in its output places.

**Causal-enabled method**

**Algorithm 5** Causal-enabled method

**Input:** The method takes a Reversible Petri net as attribute

**Result:** Find all causal-enabled transitions for a given Petri net

```
find all executedTransitions() and save them in a list named executed;
 boolean flag=false;

 foreach transition t∈executed do
    find all inputArcs for t();
     find all outArcs for t();

    foreach arc a∈outArcs do
        foreach element e∈a do
            foreach arc in∈inputArcs do
                if in.place contains e then
                    flag=true;
                end
            end
        end
    end

    if flag==true then
        executed.coEnabled=true;
    end
end
```

**Figure 4. 11: Causal-enabled method written in pseudocode.**

The figure above (Figure 4.11) shows the causal-enabled method written in pseudocode. Firstly, it finds all the executed transitions, for a given Reversible Petri net, and it saves them to a list, named executed. Then, for each executed transition it finds all its input and output arcs and saves them to two lists. Afterwards, for each output arc *a* and for each element of *a* , it passes all places of input arcs and checks if the element is present in any of them. If is presented, it assigns the value true to a boolean variable. Finally, if that boolean value is true, it assigns the co_enabled field of that transition to true.

**Causal execution method**

**Algorithm 6** Causal-execution method

**Input:** The method takes a transition t as parameter.

**Result:** If transition t is co-enabled,it is executed in reverse order, using causal algorithm's rules.

```
find inputArcs for transition t and save them to a list();
find outputArcs for transition t and save them to a list();
find inputPlaces for transition t and save them to a list();
find outputPlaces for transition t and save them to a list();

if t is coEnabled then
    foreach outArc outa∈outArcs do
        foreach element e∈outa do
            if e is token then
                tranfer e from corresponding outputPlace to corressponding inputPlace ();
            else
                if e is newly created bond b then
                    breakTheBond b;
                    relocate elements of b in corresponding inputPlaces();
                    /*elements of b, can be either tokens or other bonds.*/
                end
                if e is bond, and bases from that bond are connected with other bases then
                    breakTheConnectionof(e); if there are connected components that they are
                    consists of a base of e then
                        relocate connected component in the corresponding inputPlaces(); re-
                        locate the base left from e in corresponding inputPlaces();
                    end
                end
            end
        end
    end
    t.historyValue=0;
end
```

**Figure 4. 12: Causal-order execution method written in pseudocode**

The figure above (Figure 4.12) represents the causal-order execution method written in pseudocode. It can be observed that the effects of the execution in causal-order are exactly the same as backtracking execution.

### 4.2.4 Out-of-causal algorithm

Out-of-causal reversibility can take place whenever a transition is executed before, i.e. it has a history value bigger than zero. When a transition is executed in reverse, using out-of-causal reversibility, it can be reversed before its effects are undone, and this makes possible the creation of new states, which they cannot even be reached with forward, backtracking, or causal-order execution paths. Thus, for out-of-causal-order algorithm

they have been implemented two methods, one for the o-enableness, and one for the out-of-causal execution.

### Out-of-causal-enabled method

**Algorithm 7** Causal-enabled method

**Input:** The method takes a Reversible Petri net as parameter
**Result:** Find all out-of-causal-enabled transitions for a given Petri net

```
foreach transition t∈PetriNet do
    if t.historyValue > 0 then
        t.oEnabled=true;
    end
end
```

**Figure 4. 13: Out-of-causal-enabled method written in pseudocode.**

The figure above (Figure 4.13) shows the o-enabled method written in pseudocode. Since, a transition is executed, is identified as o-enabled, so the method above just checks the history value of each transition, and if it greater than zero, it assigns to its boolean variable o_enabled the value true.

**Out-of-causal execution method**

**Algorithm 8** Reversal of transition t in out-of-causal order

```
1:  reverse transition t by setting H_n(t) = 0
2:  for all p ∈ P do
3:      M_n[p] = M_n[p] − effect(t)
4:      for all a ∈ B do
5:          C = con(a, M_n[p])
6:          t' = last transition in path(M_0, M_n)
7:          while ((C ∩ post(t') = ∅ or H_n(t') = 0) and t ≠ ε do
8:              t' = one transition back
9:          end while
10:         if t' = ε then
11:             for all p' ∈ P do
12:                 if C ⊆ M_0[p'] and p ≠ p' then
13:                     p = p − C
14:                     p' = p' + C
15:                 end if
16:             end for
17:         else
18:             for all p' ∈ P do
19:                 if C ∩ F(t', p') ≠ ∅ and p ≠ p' then
20:                     p = p − C
21:                     p' = p' + C
22:                 end if
23:             end for
24:         end if
25:     end for
26: end for
```

**Figure 4. 14: Out-of-causal execution method written in pseudocode[13]**

The figure above (Figure 4.14) [13] represents, in pseudocode the out-of-causal execution. Firstly, it resets the history of the transition to zero, and then it finds the effect (t), which is the bond that transition t, had created, and breaks it. Afterwards, for each place *p* of the Petri net, and for each bond a of place *p*, it saves bond *a* in the current marking, in a variable. Then, it also saves the last transition of the marking, before place *p*, in a variable named t'. Thereafter, while bond *a* does not belong to the output arc of transition t or history value of t' is equal to zero, it changes the value of t', to a transition back in the marking, and this action implements the idea of going as back as possible in the marking for that specific bond. After that, if t' is not null, i.e. there is a transition in the marking where bond *a*, has used before, for each place p' in places, if bond *a* in the current marking belongs to it and is not the same as place p, it relocates bond *a*, from place p, to place p'. On the other hand, if t' equals to null, it means that in

the current marking there is not a transition, which has used bond *a*, so it finds the place that the bond belonged in the initial marking and relocates it to that place.

## 4.3 Graphical User Interface

The simulator runs through a graphical user interface, in order to be easier for the user to use the simulator and execute the main commands. The graphical user interface was implemented simultaneously with the algorithms' implementation. The GUI initially gives to the user the opportunity to choose the method by which the input will be inserted, through the following figure (Figure 4.15). Depending on the user's choice a different set of screens will appear to the user, for the input of the initial marking.

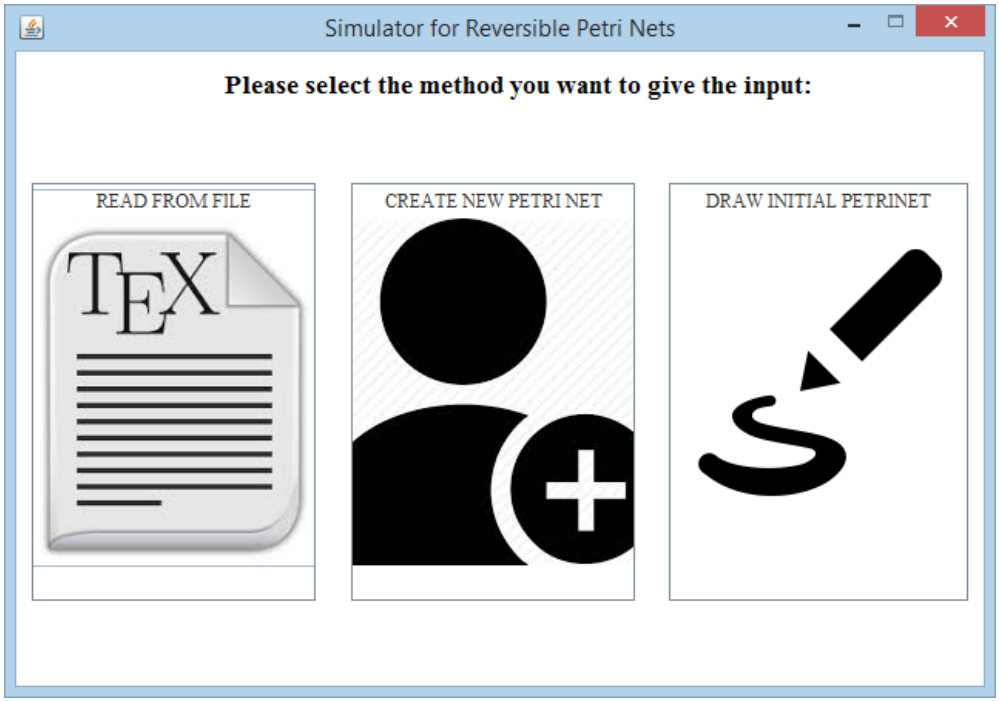



**Figure 4. 15: Screen with user's choices about the input method**

### 4.3.1 Read input from file

If the user selects the first choice, it means that the input will be imported to the simulator from a file. When the user press the button for "Read from file" the following window will appear:

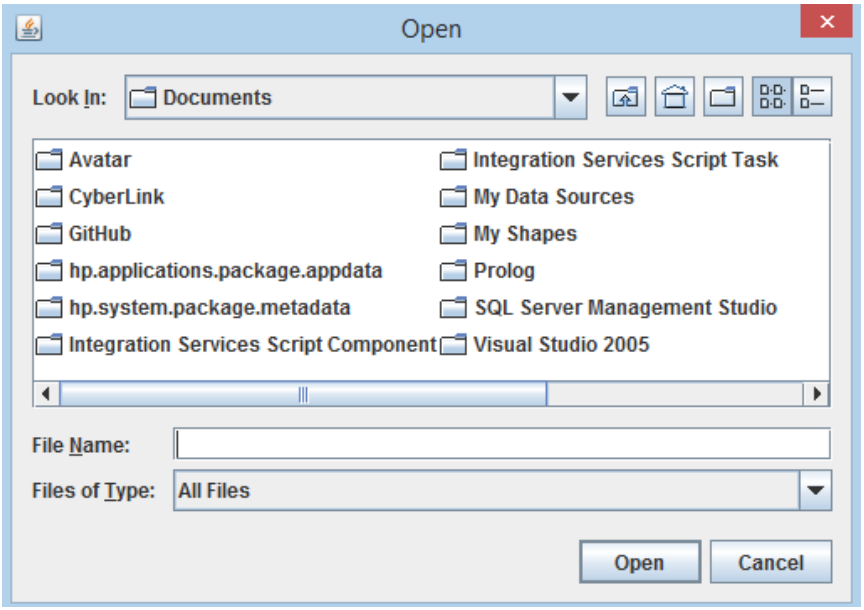



**Figure 4. 16: Window for importing the input file**

The figure above (Figure 4.16) shows the window that allows the user to choose a file, from its computer, for giving the initial marking to the simulator. The file has to be a .txt file, otherwise the simulator will show an error message, and then the user can try again to select a different file with the correct extension. In order for the simulator to read the file and create all the necessary objects and fields for the initial marking of the Reversible Petri net, the format of the file has to be as follows:

```
Tokens:
[a,b,c]
Places:
[p1,p2,p3,p4,p5]
Transitions:
[t1,t2]
Arcs:
[
(p1,t1)={a}
(p2,t1)={b}
(t1,p3)={a-b}
(p3,t2)={b}
(p4,t2)={c}
(t2,p5)={b-c}
]
Initial marking:
[
p1->a
p2->b
p3->0
p4->c
p5->0
]
```

**Figure 4. 17: File format and ordering of initial marking**

The figure above (Figure 4.17) shows the correct format of the file, in order for the simulator to create correctly the initial marking. As is shown, firstly the set of tokens is declared using the '[]' symbols to represent the beginning and the ending of the set. Then, places and transitions are defined similarly to tokens. The arcs are declared with the use of parentheses to indicate which place and transition are connected to that arc, and then after the equation mark inside the curly braces ('{}') the label of the arc is defined. Finally, the last part of the file must be the initial marking, which will indicate the distribution of tokens in the places. It is important that each file given to the simulator as input, has this format, with the same headings, symbols, and structure, in order for the simulator to work correctly.

When the user selects a file with the correct extension the simulator will read the file, create all objects for the object oriented approach of that initial marking, and continue to the next screen.

### 4.3.2 Create new Petri net from GUI

If the user selects the second choice, a different screen will appear, in which he/she will have to opportunity to create a new Petri net from the Graphical User Interface.

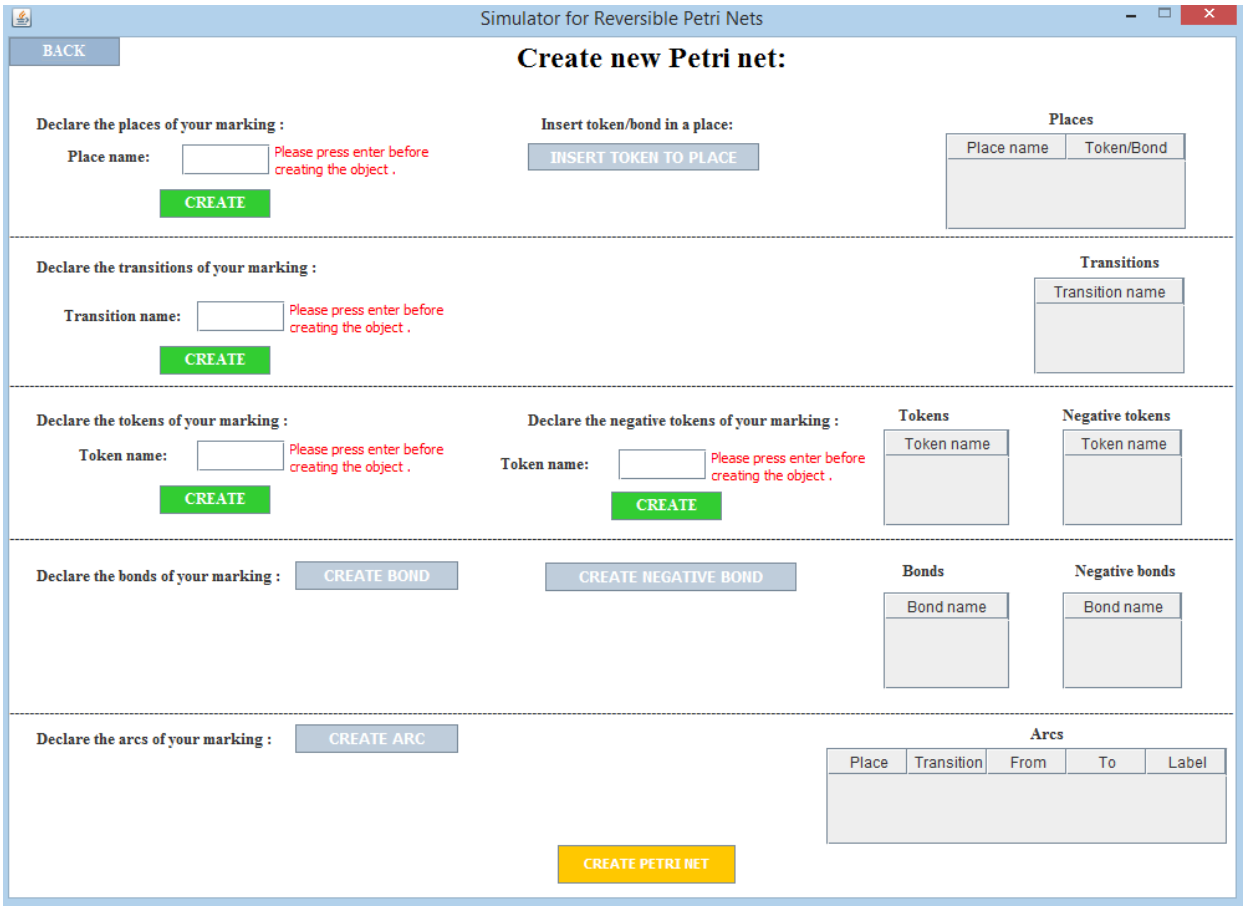



**Figure 4. 18: Screen for the creation of a new Petri net through the GUI**

The figure above (Figure 4.18) shows the screen that appears when the user chooses to create a new Petri net from the GUI. The user has to define all elements of the initial marking, and then click to “Create Petri Net” button to create the marking. The user has to define places, transitions, tokens and any negative tokens and/or bonds before defining arcs, because of the dependence between them. Furthermore, before the user creates a bond or a negative bond all tokens have to be defined, because bonds are consisting of exactly two tokens. All these checks are done from the simulator and the corresponding messages appear when it is necessary. All messages are in Appendix E – A Simulator manual for user. When a user creates an element successfully it will appear in the corresponding table at the right side of the screen.

### 4.3.3 Read input from graphical representation tool

In this case the user desires to give the initial marking of the Petri net through the graphical representation tool. Thus, the simulator is connected with the graphical representation tool, and this happens through an explanation screen, which explains to the user the steps that have to be taken, in order to create a reversing Petri net graphically. Unfortunately, the Obeo Designer software does not provide a way to make the project executable and can be accessible from Java code, so the simulator has a button that redirects the user to the environment of Obeo Designer, and then there are some further steps to create the reversing Petri net through the tool.

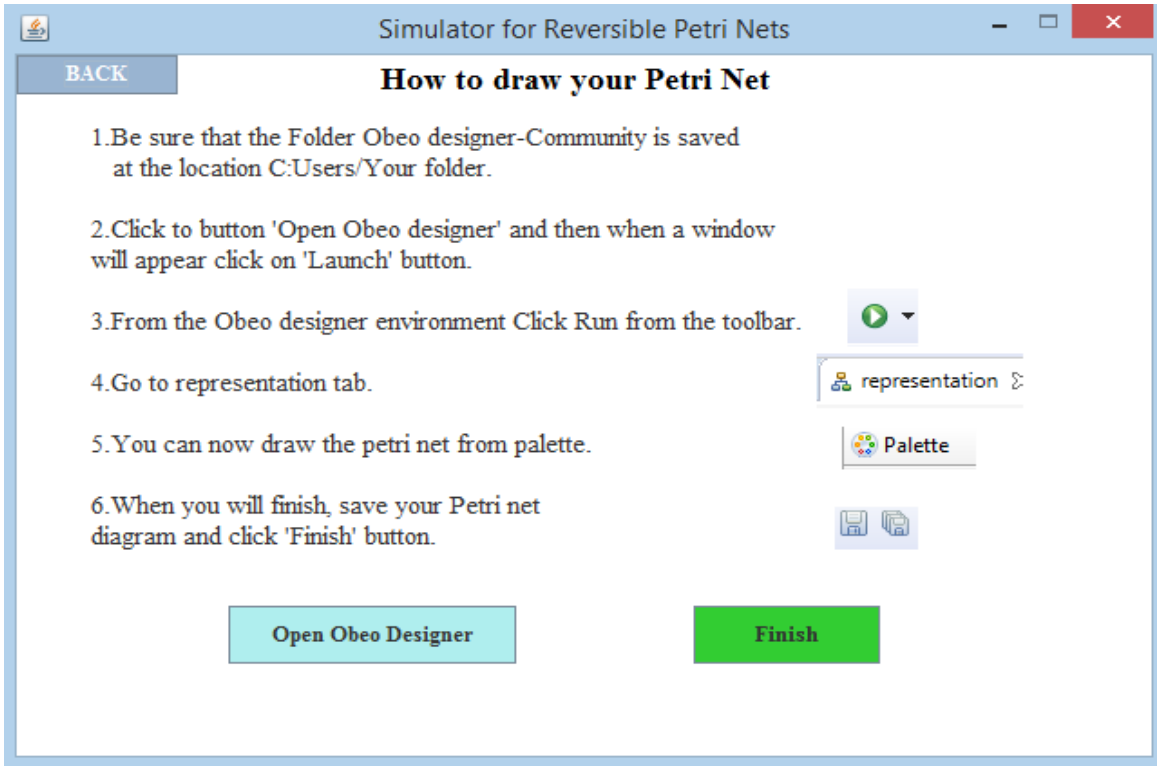



**Figure 4. 19: Explanation screen to connect with the Graphical Representation tool**

The figure above (Figure 4.19) represents the explanation screen that is mentioned before. The first step for the user is to ensure that the folder which contains all the necessary files and folders for Obeo Designer are stored at the location C: Users/User's_name on their personal computer. Then, the user has to click on the left button of the screen, named "Open Obeo Designer," and a window will appear where the user has to select "Launch" option and then Obeo Designer's environment will open. Therefore, from the Obeo Designer's environment the user has to click at the run button, in order to create a new runtime configuration for the reversing Petri nets domain model. When the new runtime configuration is created, a representation task is visible, and from there the user can create any valid Reversing Petri net marking.

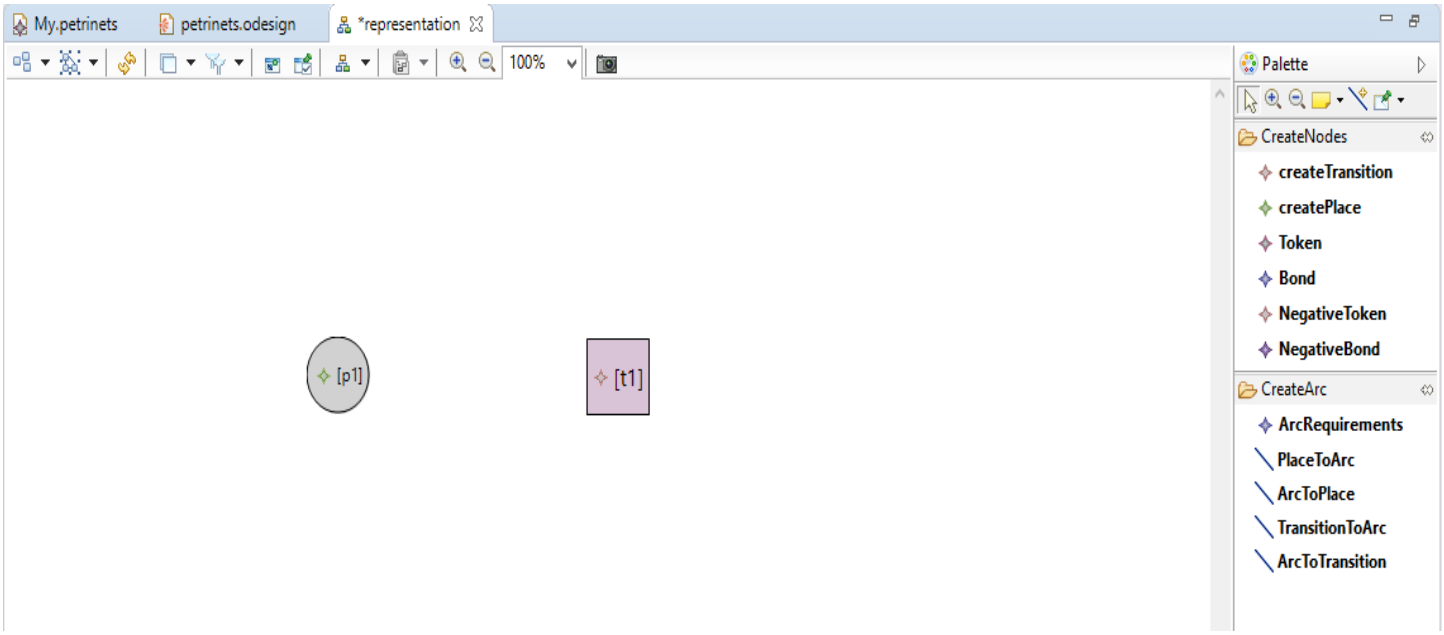



**Figure 4. 20: Graphical representation tool's environment**

The figure above (Figure 4.20) represents the Graphical representation tool's environment after the execution of the steps 1 until 4. There the user can create the desirable initial marking and when it is complete, the "Save" button has to be clicked, in order for the changes to be saved in the xml file. Since the initial marking has been created and be saved, the user has to click "Finish" button from Simulator's explanation window to continue.

When the user clicks the "Finish" button, the simulator calls the Parser's code, in order to read and process the xml file and create the object of the Petri net in an understandable way for the simulator.

### 4.4.4 Simulator execution choices screen

Independently of the import method of the initial marking, since the initial marking has been read, the simulator is ready to find forward and/or reverse enabled transitions and execute any of them. Thus, after the import of the initial marking the user will be redirected to a screen showing all options.

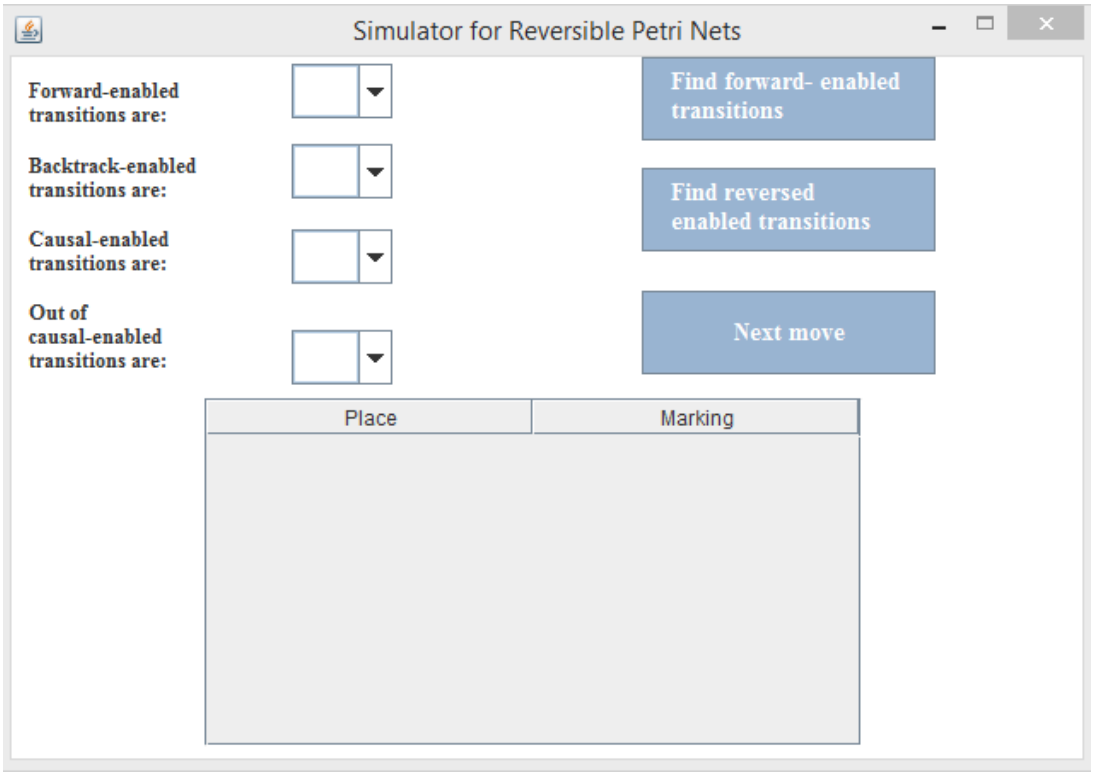



**Figure 4. 21: Screen representing all execution options to user**

The figure above (Figure 4.21) shows the simulator's screen which shows all the available choices for execution. Firstly the user has to select one of the buttons named "Find forward-enabled transitions" and/or "Find reversed enabled transitions." If the user selects the button "Find forward-enabled transitions," the simulator will call the forward_enabled function which is mentioned in chapter 4.2.1.1, and the corresponding combo box will be filled with all forward enabled transitions of the current marking. In addition when the user selects the button "Find reversed enabled transitions" the simulator will call bt_enabled, causal_enabled and out_of_causal enabled functions, which are mentioned in chapters 4.2.2.1, 4.2.3.1, and 4.2.4.1 respectively.

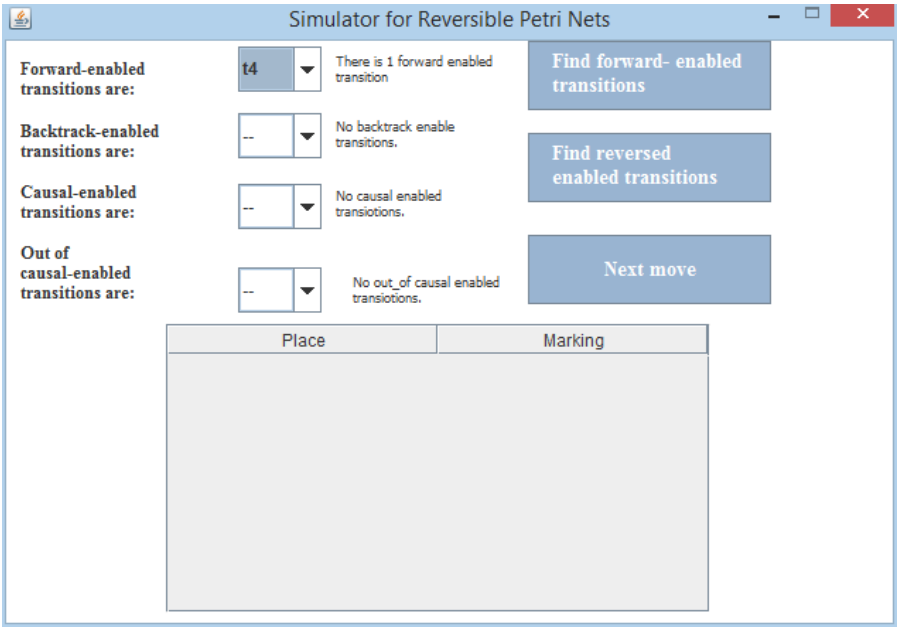



**Figure 4. 22: Representation of screen after selecting "Find forward-enabled transitions" and "Find reversed enabled transitions" buttons**

The figure above (Figure 4.22) shows how the screen changes when "Find forward-enabled transitions" and "Find reversed enabled transitions" buttons are clicked. Since the simulator identified the enabled transitions, the user can select any transition from this set, and by clicking the "Next move" button, the corresponding algorithm will be called from the simulator, based on the combo box that has a selected item. The alternations shown in the screen after the execution of a transition are shown in the following figure (Figure 4.23).

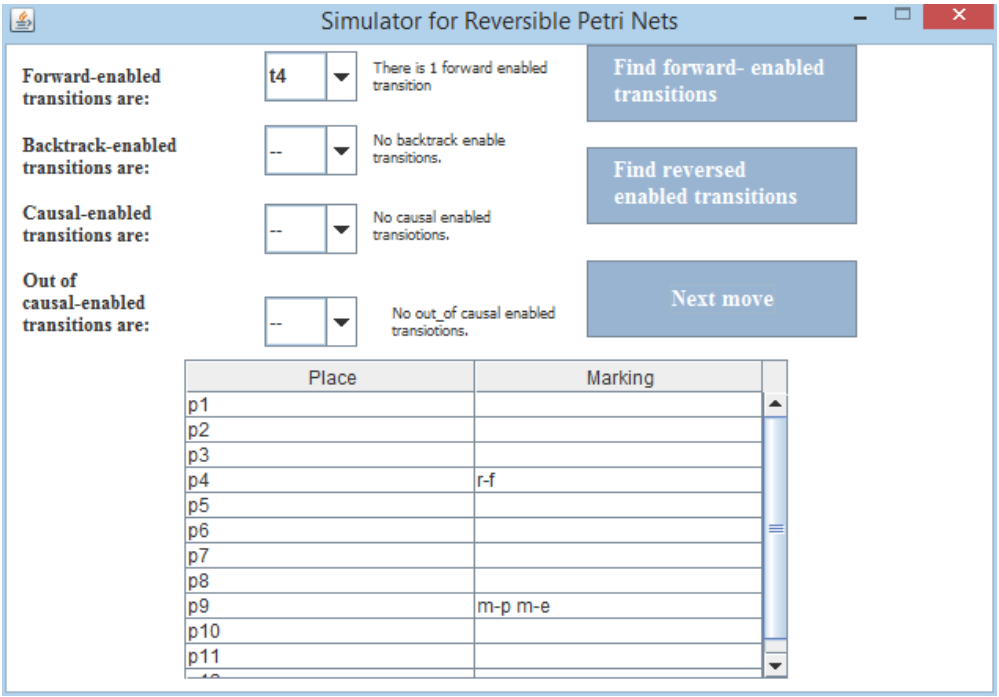



**Figure 4. 23: Alterations in screen after execution**

As shown above (Figure 4.23) after the execution of a transition the table at the down side of the window, is filled with the places of the marking and the distribution of tokens/bonds in it.

The simulator does some checks about the execution of the transitions as well. Firstly, it confirms that the user has selected only one transition for execution, i.e. a forward-enabled transition or a backtrack-enabled transition, and not both. If the user selects two enabled transitions for execution the simulator will show a warning message, as represented in the Figure 4.24.

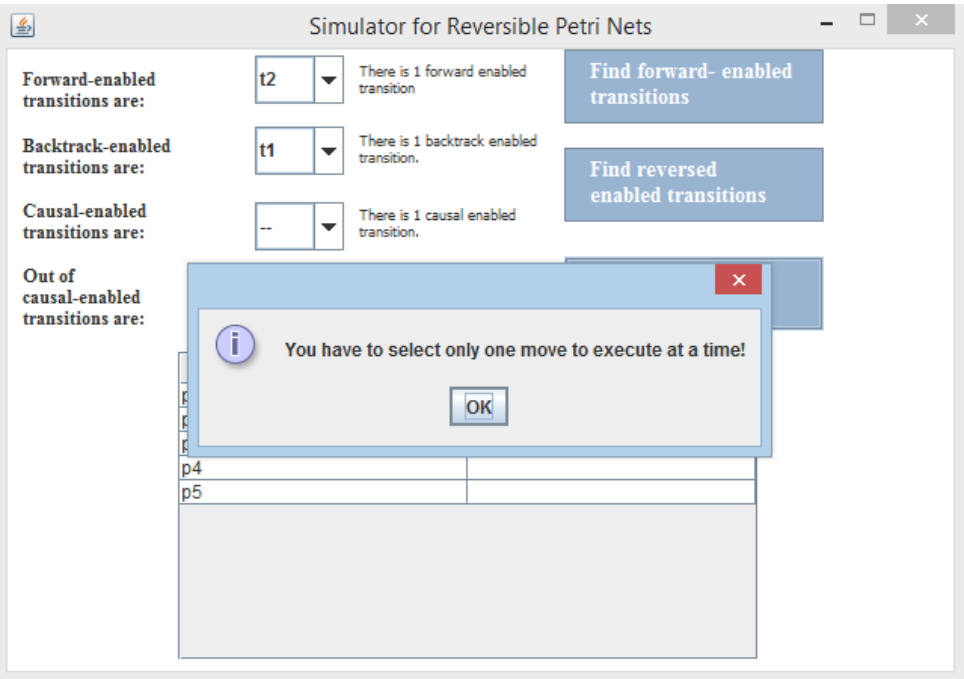



**Figure 4. 24: Warning message appeared after the selection of two transitions for execution**

### 4.3.5 Visual changes on Reversing Petri net diagram

In case that the user chooses to import the initial marking through the graphical representation tool, an extra feature is available for the user, which is to observe the changes on the graphical model as well. This feature is provided by the simulator because of the parser that converts the java representation to xml, and has been mentioned above, is responsible for modifying the xml file, when a transition is executed either in forward or reverse order. When the user executes a transition and the choice of the use of importing initial marking through the graphical representation tool have been selected, the xml file is changed, and the user can return to the tool to see the changes. Before the user can see the changes, the xml file has to be updated, and be read

again from the Obeo Designer software, so the last version is presented on the screen. In order for the xml to be updated, the user has to extend the package org.eclipse.sample.petrinets.sample and find the file My.petrinets. Then the user clicks on that file, the xml file will be updated, and the new marking will be available on the screen. The following figure (Figure 4.25) shows how the user can update the xml file.

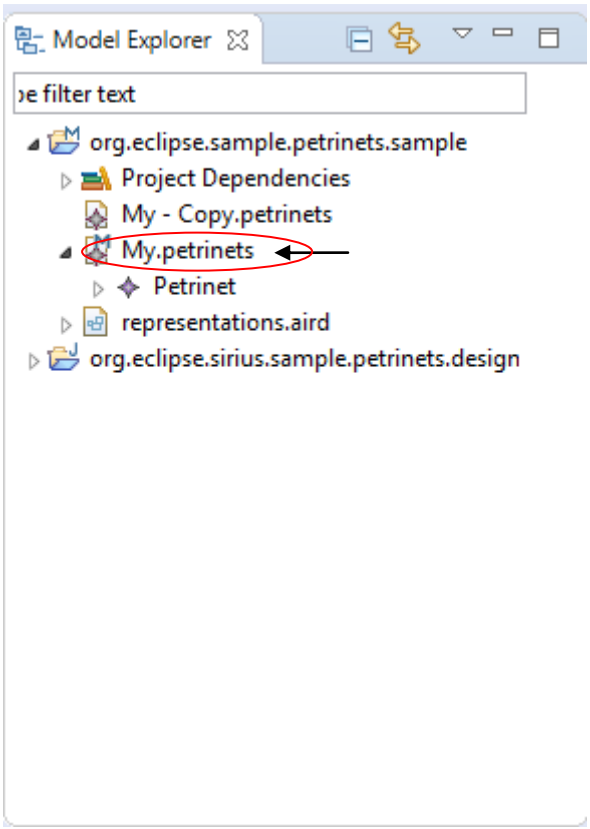



**Figure 4. 25: Screenshot from tool at the point of updating the xml file**

## 4.4 Graphical representation tool

The graphical representation tool has been implemented with Obeo Designer software, with the completion of a series of actions. Generally, the graphical representation tool is aimed to give the opportunity to user of drawing easily the initial marking of the Petri net, using shapes from an interactive palette, and define all the necessary information through the tool visually. The tool is connected with the main Graphical User Interface of the simulator. Moreover, when the user gives the initial marking to the simulator, and informs the simulator that the Petri net is ready to be processed, the user has the opportunity to find forward, backtrack, causal and out-of-causal-enabled transitions of the marking, and execute any enabled transition in forward or reverse order, from the main interface of the simulator. Thereupon, after each transition execution, the user has the chance to watch the changes on the marking directly from the graphical representation tool, besides the simulator's interface.

### 4.4.1 Domain model for the Reversing Petri nets

Foremost, as it mentioned above, a Class diagram was designed, in order to behave as the Domain model for the Reversing Petri nets, in Obeo Designer's project. The domain model of the project defines the main components of the model, for which an interactive workbench is desired. Since all components had been declared, including their attributes, the relationships between them had to be set. In case of Reversing Petri nets, the main components are place, transition, arc, token, bond, negative token, and negative bond. All these components are connected to a central component named Petri net. Petri net component is the element that uses all other elements of the model, in order to define a Reversing Petri net marking. The relationships between the items, determine which components can be connected with other specific components. In case of the Reversing Petri nets there are relationships between place and arc classes, and between transition and arc classes, to declare the relationship between place and transition. The presence of the Arc class, is to define the direction of the relationship (from place to transition or from transition to place), and to give the opportunity to user add the desired label on the arc. Thus, in order to add a label on arc, there is a relationship between arc and tokens, bonds, negative tokens and negative bonds. Moreover the user must have the chance to insert tokens and/or bonds in places, so a relationship exists between these elements and places. The last relationship that

completes the definition of the domain model is the relationship between tokens and bonds, which indicates that a bond consists of exactly two tokens from the marking. The domain model that was created for the Reversing Petri nets used in a subsequent stage of the graphical representation tool, in order to define which components of the palette can be connected with others, and which ones cannot be connected with which. Thus, it sets the rules for the creation of any Reversing Petri net, using the graphical representation tool.

### 4.4.2 Design specifications for the Reversing Petri nets workbench

Since the domain model for the Reversing Petri nets was constructed, and it could be used as the metamodel of the graphical representation tool, the next step was the creation of design specifications, in order to guide the user's choices. The creation of the design specifications was done through the Run –time configuration window of Obeo Designer software, which automatically generates the code that is needed, using the metamodel for the reversing Petri nets

Firstly, all main components of reversing Petri nets had to be defined at the design specifications and for each component a unique shape had to be assigned. Thus, for place a circle was chosen, for transition a square was chosen and for arcs a directed arrow was chosen, as they are in the definition. However, because the design specifications are based on the Class diagram and its relationships between its classes, in order to create the arcs in the right way, and the output would be efficient, the arc had to be defined separately if it is from place to transition, or if it is from transition to place. In addition because Arc is an independent class in the Class diagram, the only way for the software to identify the relationship between arc, place, and transition is the creation of an arc object before the creation of the arc. So, in the design specifications for the palette options, an arc object was created, which is identified by a small rectangle shape, and can be connected with an arc from place or from an arc from transition. Then, it can connect the first component with the corresponding component with similar arcs.

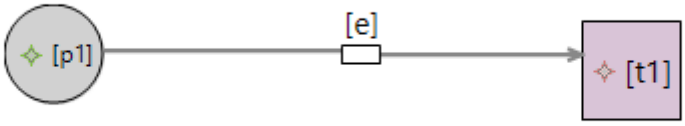



**Figure 4. 26: Representation of arc in the Graphical representation tool**

The figure above (Figure 4.26) shows how the arc is represented in the Graphical representation tool. It can be observed that the small rectangle, which indicates the presence of an arc, acts like the requirements field of arcs. The existence of this object is to allow the user to add tokens, bonds, negative tokens and/or negative bonds for an arc. Moreover, for tokens and bonds a shape had to be assigned too, so a small diamond was assigned for them, with a different color for each type. If the user creates a token the diamond color is light blue, while if a bond is created the diamond color is light green. However, if a negative token or a negative bond is created the diamond color is red, to indicate that the absence of this token/bond is necessary.


**Figure 4. 27: Representation of tokens, bonds, negative tokens/bonds in the tool**

The figure above (Figure 4.27) shows how tokens, bonds, negative tokens/bonds are defined to be represented in the Graphical representation tool.

So, after the definition of all components for their design representation, a definition about the relationships between them had to be declared. The next step was to define which components can contain other components, based on the relationships of the Class diagram. Hence, some rules were created, in order to allow the user adding tokens and/or bonds in places, and requirements in arcs.

| Property | Value |
|---|---|
| Place false | |
| Arc | Arc arc1 |
| Bond | |
| Empty place | false |
| Place id | 1 |
| Place name | p1 |
| Token | |

**Figure 4. 28: Place semantics in the Graphical representation tool**

| Property | Value |
|---|---|
| Arc arc1 | |
| Arc id | 1 |
| Arc name | arc1 |
| Bond | |
| From | 112 |
| Negative bond | |
| Negative token | |
| Place | |
| To | 116 |
| Token | Token r |
| Transition | Transition false |

**Figure 4. 29: Arc semantics in the Graphical representation tool**

The figures above (Figure 4.28 and Figure 4.29) show that the rules from the design specifications allow the users to add tokens and/or bonds into places, and to add tokens, bonds, negative tokens and/or negative bonds as arc requirements. Through these rules the user can declare the token/bond name as well. The formats of the token/bond's semantics are shown in the figures below (Figure 4.30 and Figure 4.31). Similarly, they are the semantics of negative tokens and negative bonds.

| Property | Value |
|---|---|
| Token r | |
| Token id | 1 |
| Token name | r |

**Figure 4. 30: Token semantics in the Graphical representation tool**

| Property | Value |
|---|---|
| Bond r-f | |
| Bond id | 201 |
| Bond name | r-f |

**Figure 4. 31: : Bond semantics in the Graphical representation tool**

The final step left for the completion of the design specifications was to create the palette's sections, and to divide them. Thus, two sections were created, one for the nodes of the model, and one for the edges of the model.

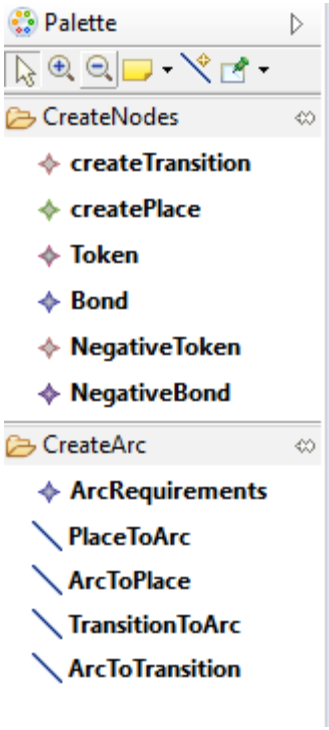



**Figure 4. 32: Palette sections of Graphical representation tool**

As it can be observed in Figure 4.32 the palette is divided in two sections, namely createNodes and createArc. The first part contains all choices for the user to create a node, such as place, transition, token, bond, negative token and/or negative bond, while the second section includes the creation of the arc object, which will contain the arc requirements (pre or post-conditions), and the edges that the user can create from and to the arc. Thus, with the completion of this step, the design specifications for the reversing Petri nets are done, and a workbench is ready for use.

### 4.4.3 Obeo Designer's output processing

After the completion of the design specifications, the graphical representation tool was ready for use, and the users could draw a Petri net. However, in order for the simulator to process the user's model as an input, some actions had to be done. Initially, the Obeo Designer software creates an xml file, while the user creates the Petri net model. The xml file contains all the necessary information about the initial marking, so this output

had to be processed from the simulator in order to create the corresponding object-oriented approach of that marking. Thus, a Parser for xml files was created in Java to parse the output from the Obeo Designer and create the object oriented approach for the simulator. In addition, in order to change the initial graphical representation of the Petri net through the simulator, while the user executes transitions, a Reverse Parser was created to change the xml file.

#### 4.4.3.1 Parser from xml to Java representation

The parser is a separate program written in Java that is connected to the simulator. Firstly, the parser finds the xml file created from Obeo Designer and opens it as a Document Builder object, which is the Class provided by Java for xml files. Then it has a function for each element of the Reverse Petri net, and each function it reads the corresponding element.

The figure below (Figure 4.33) shows the functions that are supported by the Parser.

Each function reads a specific set of elements for the Reversing Petri net initial marking. The order in which the elements are read is not random. Tokens and bonds are read first, in order to compare them with the negative tokens and negative bonds subsequently and confirm that negative tokens and negative bonds that are defined; they are in the tokens, or in the bonds set, respectively. Therefore, places and transitions are read, in order to be added in the arcs, which is the last set of elements to be read. Moreover, tokens, bonds, negative tokens and negative bonds are read before places and arcs, so they can be added to places or arcs subsequently. If a set of elements does not exist, it is not a problem because in each function the program firstly checks if that set is contained in the xml file.

```
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(file);
doc.getDocumentElement().normalize();
readTokens(doc);
readBonds(doc);
readNTokens(doc);
readNBonds(doc);
readPlaces(doc);
readTransitions(doc);
readArcs(doc);
```

**Figure 4. 33: Functions supported by the Parser**

In each function, while the elements are read the respective objects and their fields are created, and at the end a Petri net object is created, and is given to the simulator, as input.

#### 4.4.3.2 Parser form Java representation to xml

The reverse parser is also a separate program written in Java, and its aim is to alternate the xml file, when a transition's execution occurs. Thus, when a transition is executed, all tokens or bonds that have relocated to another place with these places are stored in an ArrayList. Before the end of the transition's execution the reverse parser function is called, which reads the xml file. Afterwards, it changes the format of the xml file according to the places and tokens and\or bonds stored in the ArrayList.

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(new File(Parser.file.getAbsolutePath()));
transformer.transform(source, result);
```

**Figure 4. 34: Transformation of the xml through Java**

In the figure above (Figure 4.34) is shown how the xml file change, when the reverse parser's function has made all the necessary changes. As is shown a Transformer object is created, which is a Class provided by Java, which enables editing in xml files. Therefore, a DOMSource object is created, which is also a Java Class that will store the initial xml file and then a StreamResult object will be created in order to hold the changes made in the xml file. Finally through the transformer object, all changes are made to the xml file, and can be visible on the diagram from the graphical representation tool.

# Chapter 5

## Case study

---



---

In this chapter we are going to examine the correctness and the effectiveness of the simulator, by executing two examples. The first example is an application, which represents the causal reversing behavior, while the second example is the ERK-pathway example from biochemistry in RPNs.

## 5.1 Causal order example

This case study checks the correctness of the simulator when the initial marking is the reversing Petri net in Figure 2. 12. This example represents the behavior of a reversing Petri net, when two transitions are independent, and they can use causal rules of reversibility to be reversed. Thus, with this case study our aim is to ensure that causal reversibility is executed correctly from the simulator.
The first step was the creation of an input for the simulator. Thus,we created the input in the form of a text file, in order to give it to the tool directly.
The input file is shown in the figure below:

```
Tokens:
[a,b]
Places:
[p1,p2,p3,p4,p5]
Transitions:
[t1,t2,t3]
Arcs:
[
(p1,t1)={a}
(p2,t2)={b}
(t1,p3)={a}
(t2,p4)={b}
(p3,t3)={a}
(p4,t3)={b}
(t3,p5)={a-b}
]
Initial marking:
[
p1->a
p2->b
p3->0
p4->0
p5->0
]
```

**Figure 5. 1: Definition of RPN**

Therefore, we launched the simulator in order to check its behavior and compare it with the expected one.

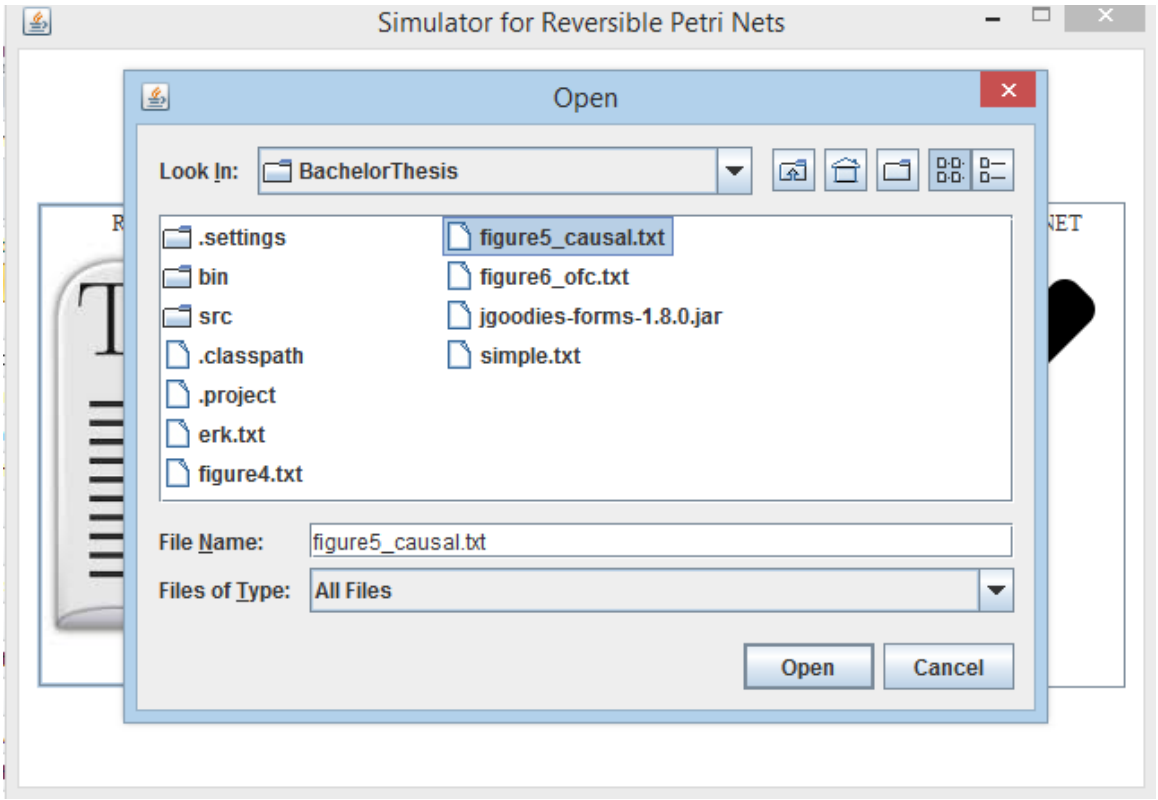



**Figure 5. 2: Choice of the file we have created**

The figure above (Figure 5.2) shows the first step of the process, which is to give the input file to the simulator. Then, the simulator redirects the user to the next screen, because the format of the file was correct, and the simulator was able to process it correctly. So, the next step is to check if the simulator can find correctly the forward-enabled and reversed-enabled transitions.

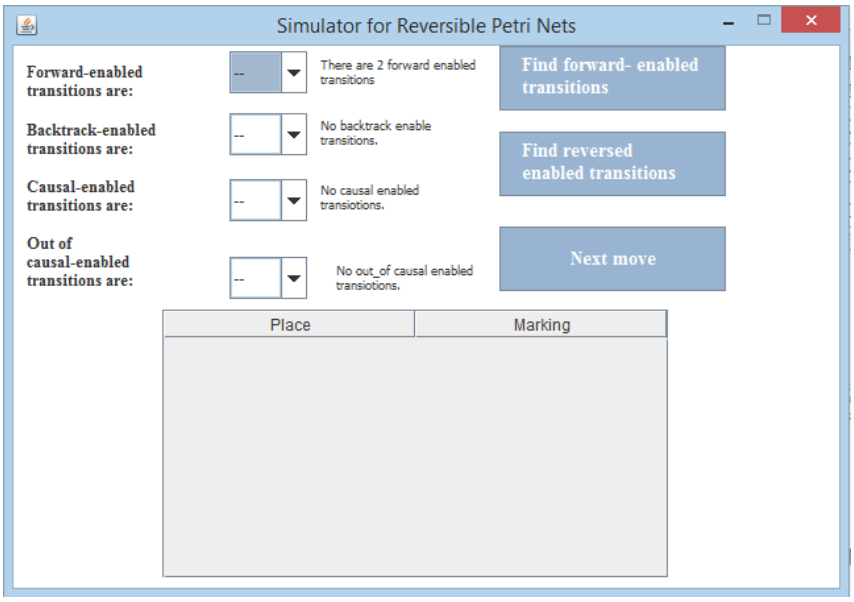



**Figure 5. 3: Simulator's results for enableness**

From the figure above (Figure 5.3) we can observe that the simulator has found two forward enabled transitions only, and no reverse-enabled transitions, correctly.
Then let us assume that we want to execute transition t1 first, and then t2 and t3.
The expected results are that after the execution of t3, the bond a-b will be created and located in place p5.

Let us now see the actual results of the simulator:

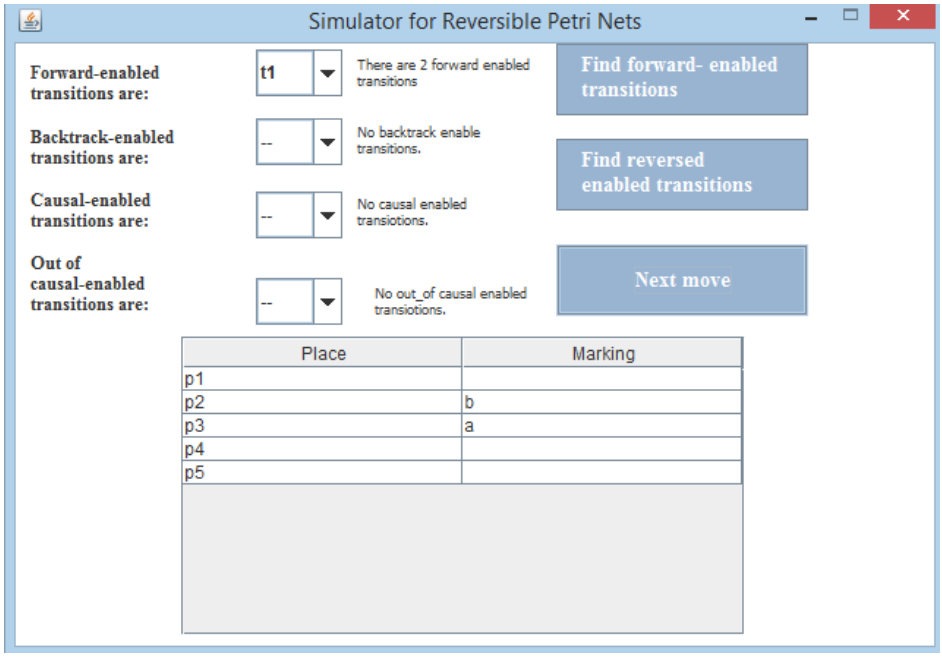



**Figure 5. 4: After the execution of transition t1**

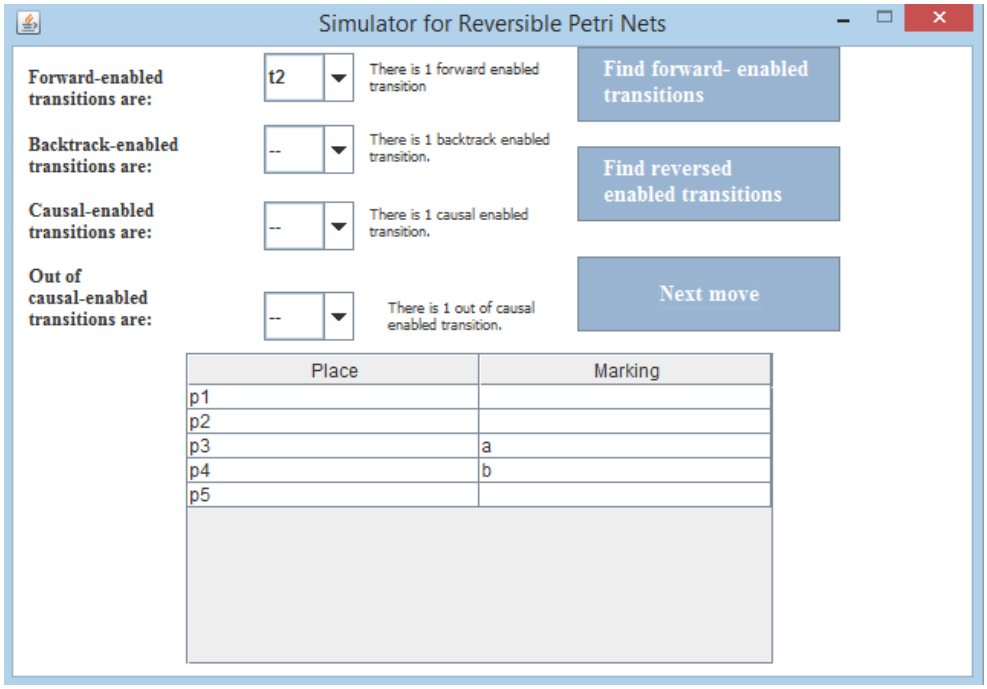



**Figure 5. 5: After the execution of transition t2**

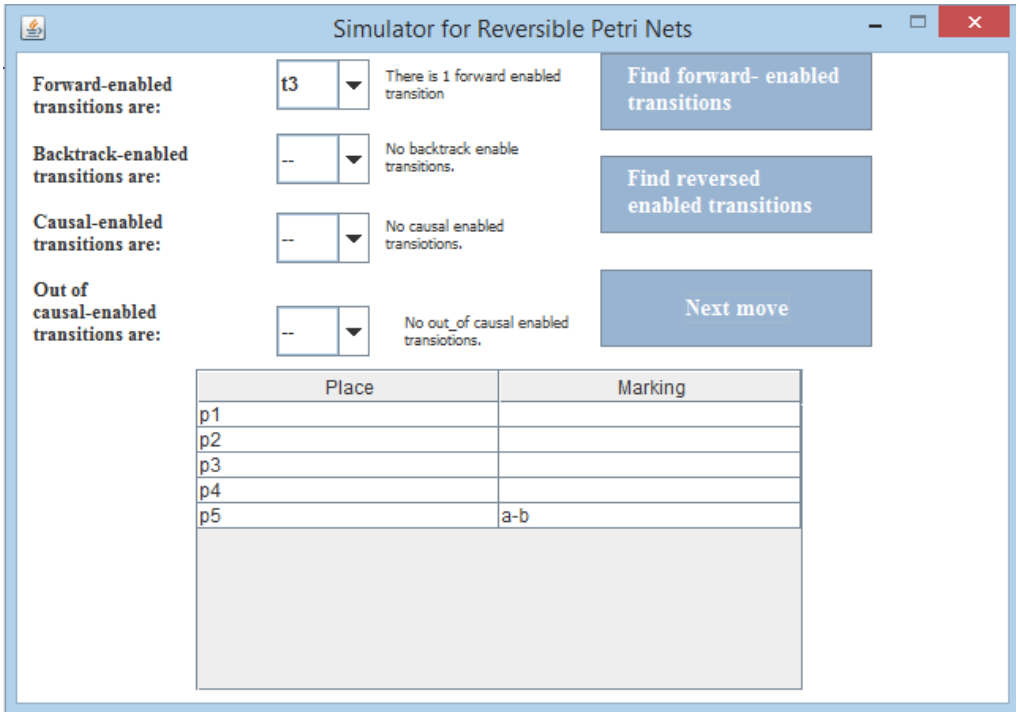



**Figure 5. 6: After the execution of transition t3**

Considering the figures above (Figure 5.4, 5.5, 5.6) we can observe that at each execution the simulator creates a new marking, where the distribution of tokens and bonds is the same as the expected one.
Lets us now check the reversible actions as well.

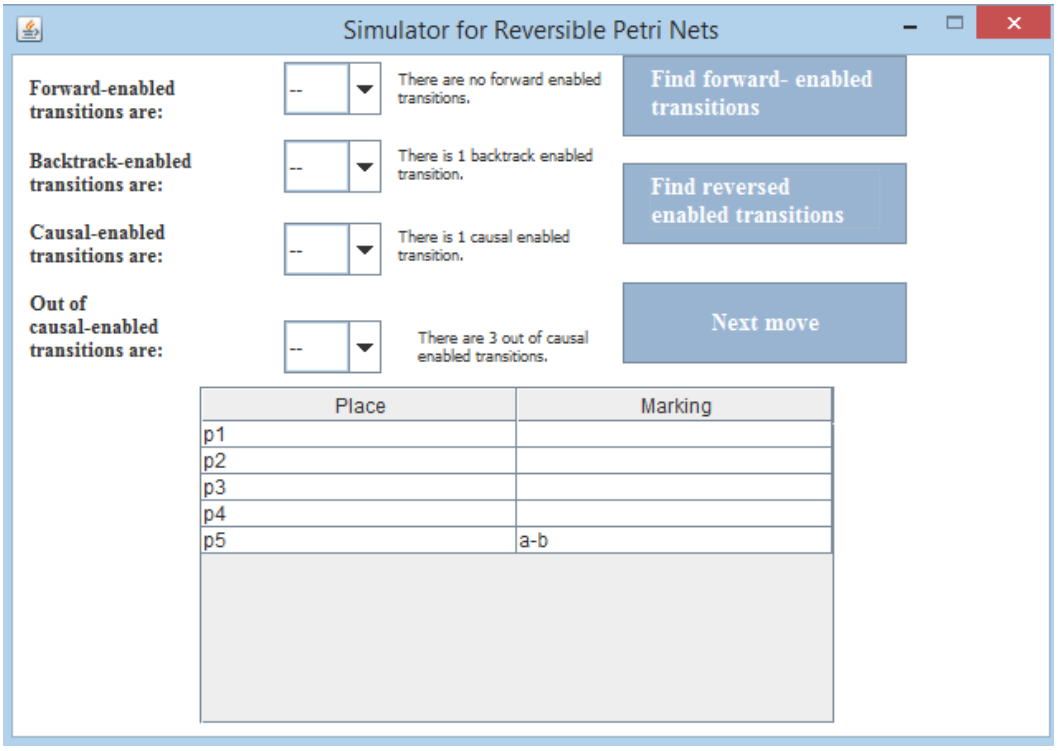



**Figure 5. 7: After the user has clicked to the buttons that find forward and reversed enabled transitions**

Firstly, from the figure above (Figure 5.7) we can observe that after the execution of transition t3, there are no more forward enabled transitions, but there are reversed enabled transitions. More specifically, there is one backtrack enabled and the one causal enabled transition, which is at both cases transition t3, and three out-of-causal enabled transitions. The results are correct, because only t3 is bt- and co-enabled at this stage of the execution, and all of them are o-enabled because they have been executed.
The next step is to execute t3, in reverse using causal reversibility, and expecting to break the connection between a-b, and relocate a to place p3 and b to place p4.

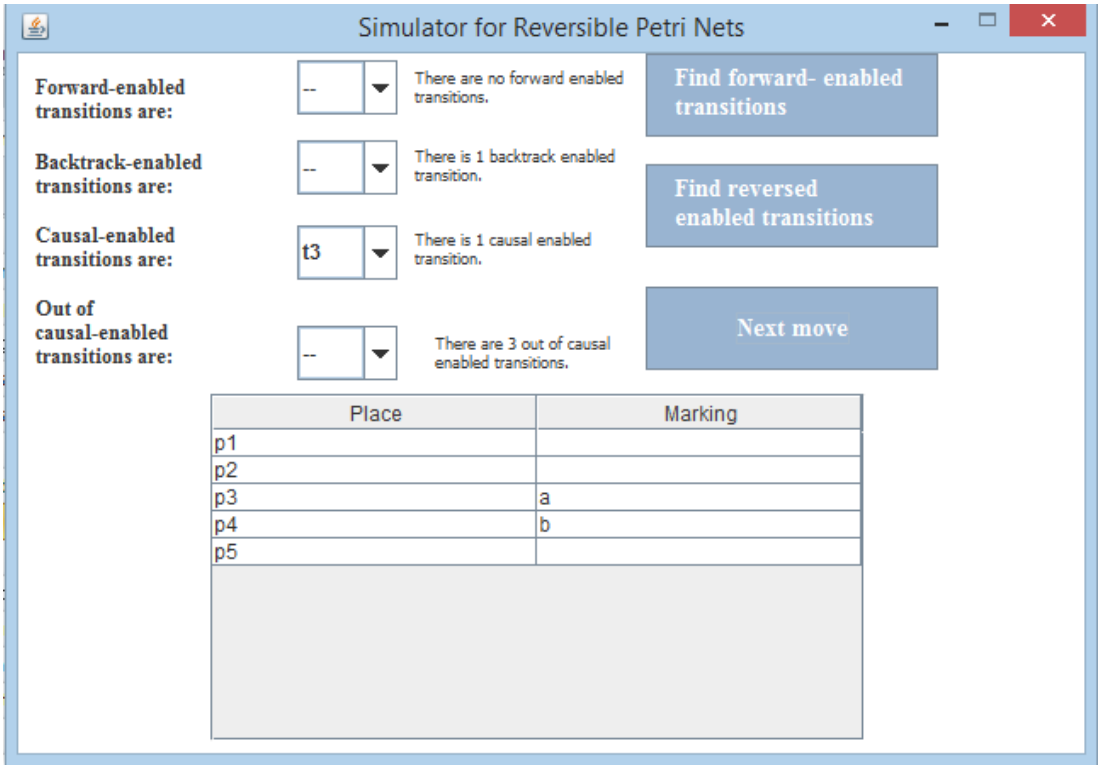



**Figure 5. 8: The new marking after the execution of the transition t3 reversiby**

The figure above (Figure 5.8) shows that the simulator has the expected results, after the execution of the transition t3, using causal reversibility.
Now, we are going to execute t2 and t1 using causal reversibility but not in the same order as they occur before. So, we want to check if the simulator finds two causal enabled transitions now.

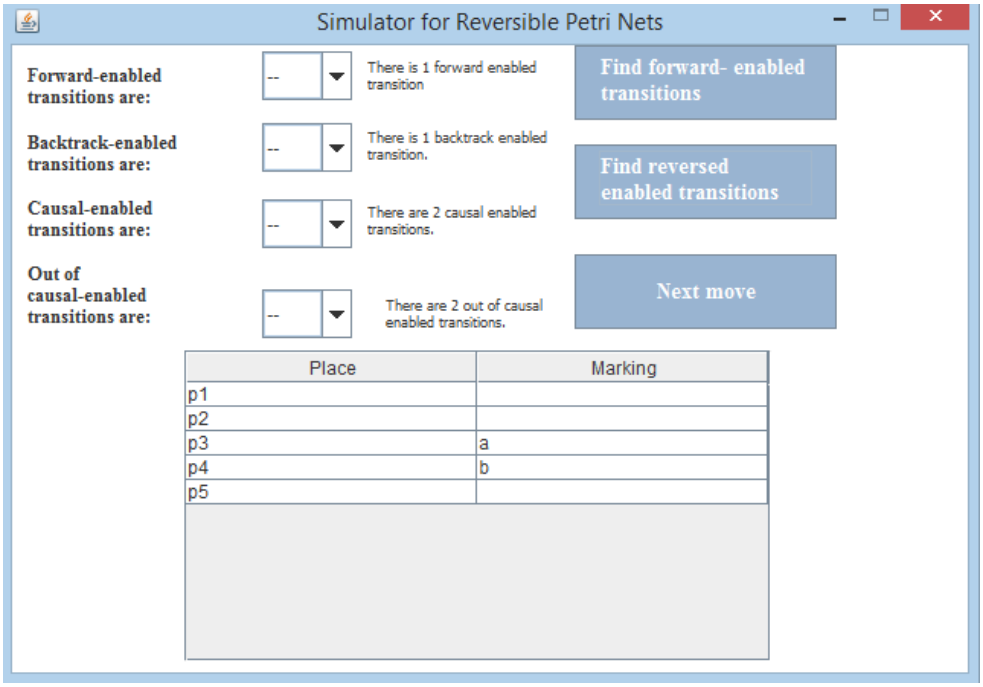



**Figure 5. 9: Results of simulator for enableness after the causal execution of t3**

The figure above (Figure 5.9) shows that the results are correct and we now can execute any of the causal enabled transitions.
Let us now assume that we want to execute the transition t1 first, and then the transition t2.

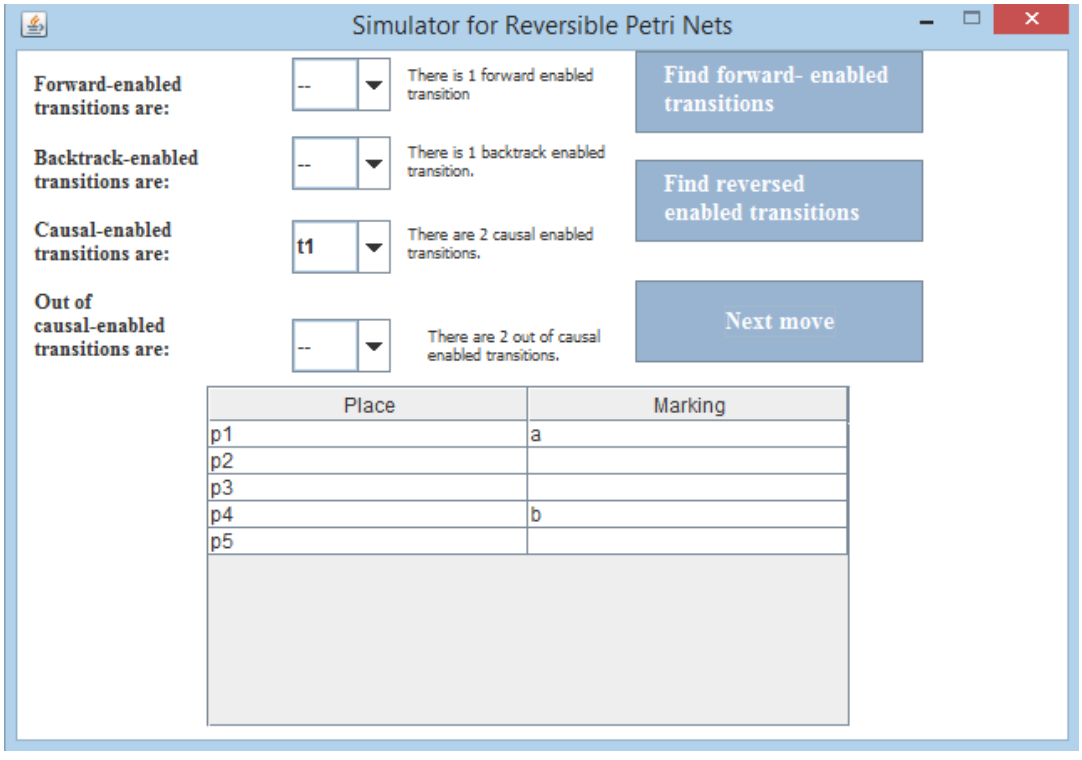



**Figure 5. 10: Results after the execution of the transition t1 with causal reversibility**

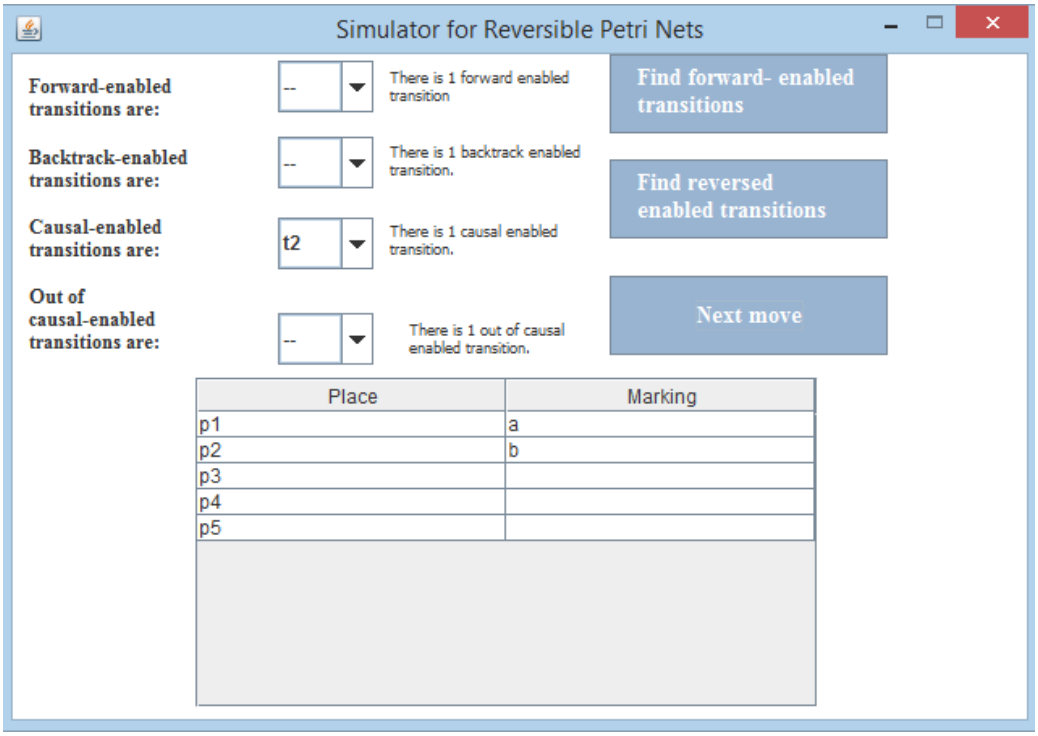



**Figure 5. 11: Results after the executin of the transition t2 with causal reversibility**

The figures above (Figure 5.10, 5.11) show the results of the simulator after execution of the transition t1 and then of the transition t2, using causl reversibility. We can observe that the markings that have been created are correct, and the tokens have been distributed correctly to the places. Moreover, the response time of the simulator is very good, as there are not idle time spaces.

This execution was the end of the case study, because we have checked the causal reversibility and the simulator has the expected results in an accepted time limit.

## 5.2 ERK-pathway example in RPNs

The second example that has been tested , is the ERK-pathway example from biochemistry.
The ERK-pathway example in RPNs is represented in the following Figure (5.12):

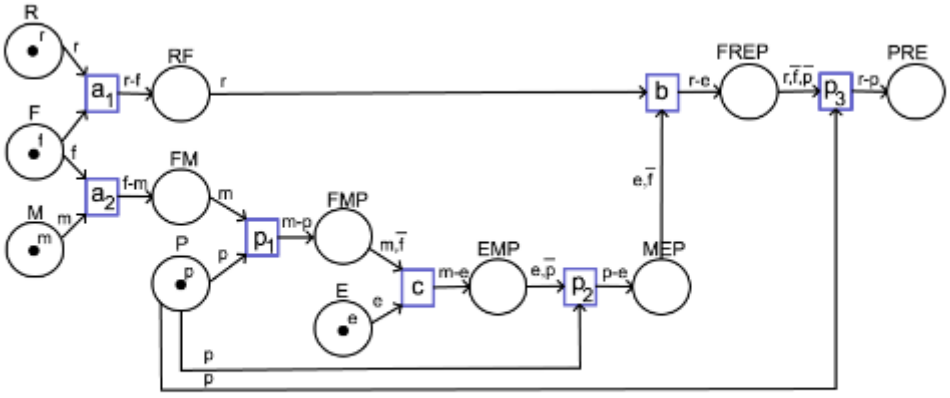



**Figure 5. 12: ERK-pathway example in RPNs [13]**

This example is ideal to check out-of-causal reversibility, because some states are not reachable with forward, backtrack and/or causal execution.
Thus, we can draw the ERK-pathway example in RPNs through the graphical representation tool, in order to check its correctness as well.
Firstly, we choose to give the initial marking through the graphical representation tool and after we have done all the necessary action we are able to draw it.
When we have drawn the initial marking it looked like that:

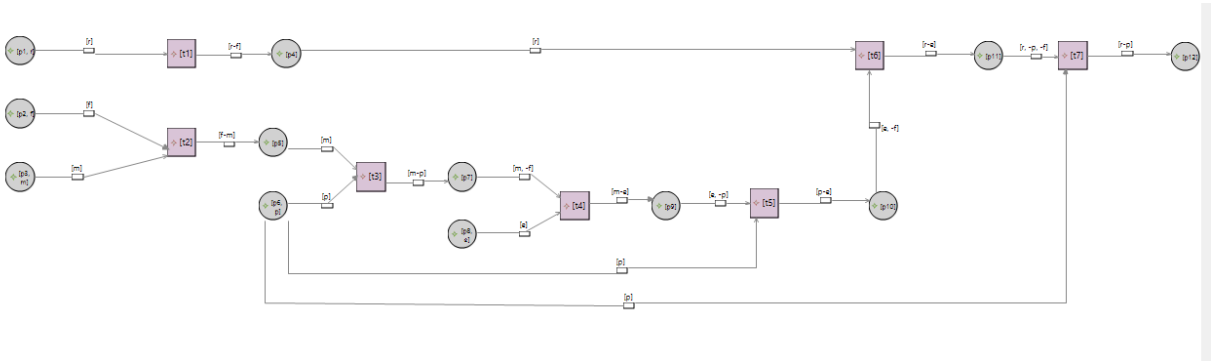


**Figure 5. 13: RPN diagram of ERK-pathway example that has been created from graphical representation tool**

Since, we have the diagram ready we can give it to the simulator, in order to process it. So, after we gave the input to the simulator, the user is redirected to the next screen, and this means that the diagram was defined correctly.

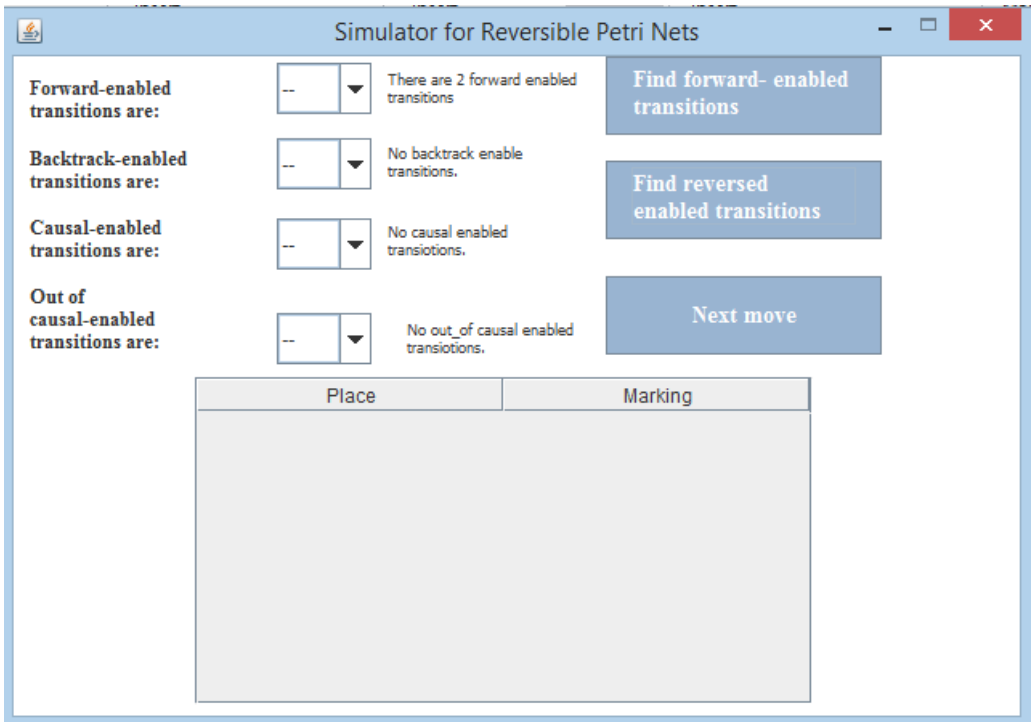



**Figure 5. 14: Results of the simulator for the enableness**

The figure above (Figure 5.14) shows that the simulator correctly found only two forward enabled transitions.

Let us now assume that we want to execute the transitions with the following order:
t2: in forward direction
t3: in forward direction
t2: in reverse direction, out-of-causal
t4: in forward direction
This sequence of executions, reaches some states that are not reachable through forward, backtrack and/or causal paths. For instance the transition t4 is not reachable, if we do not reverse t2 first, with out-of-causal reversibility.
Thus, we will execute the transitions, through the simulator to observe the results.

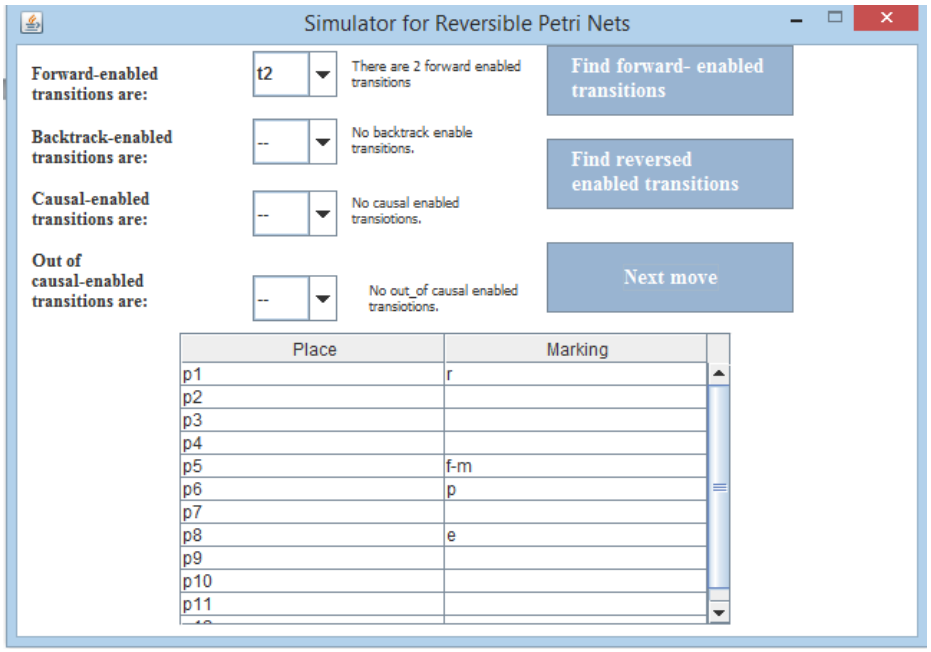



**Figure 5. 15: The new marking after the execution of the transition t2**

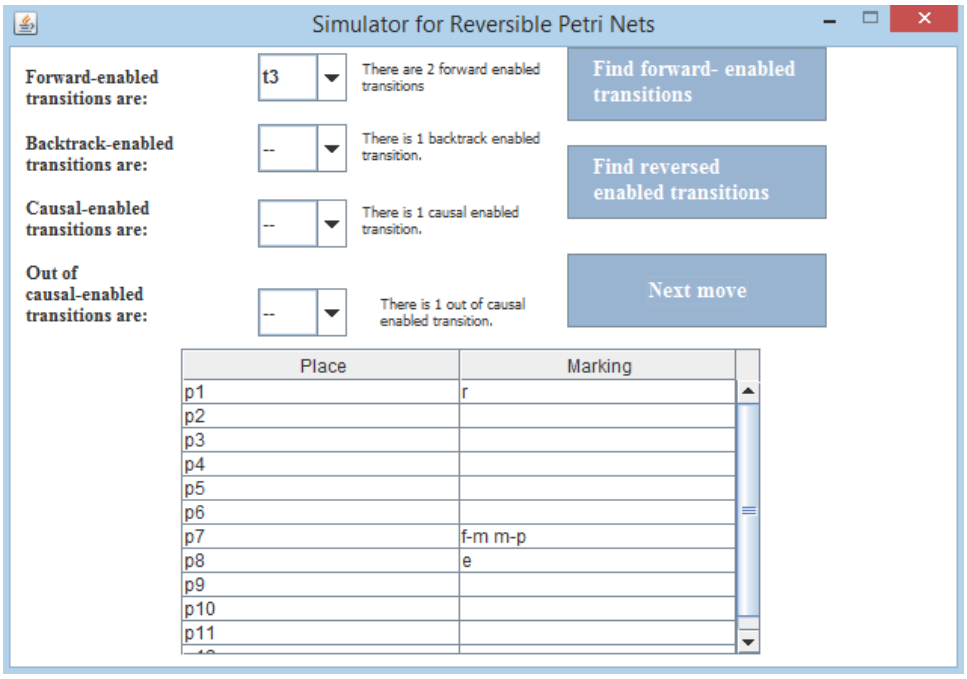



**Figure 5. 16: The new marking after the execution of the transition t3**

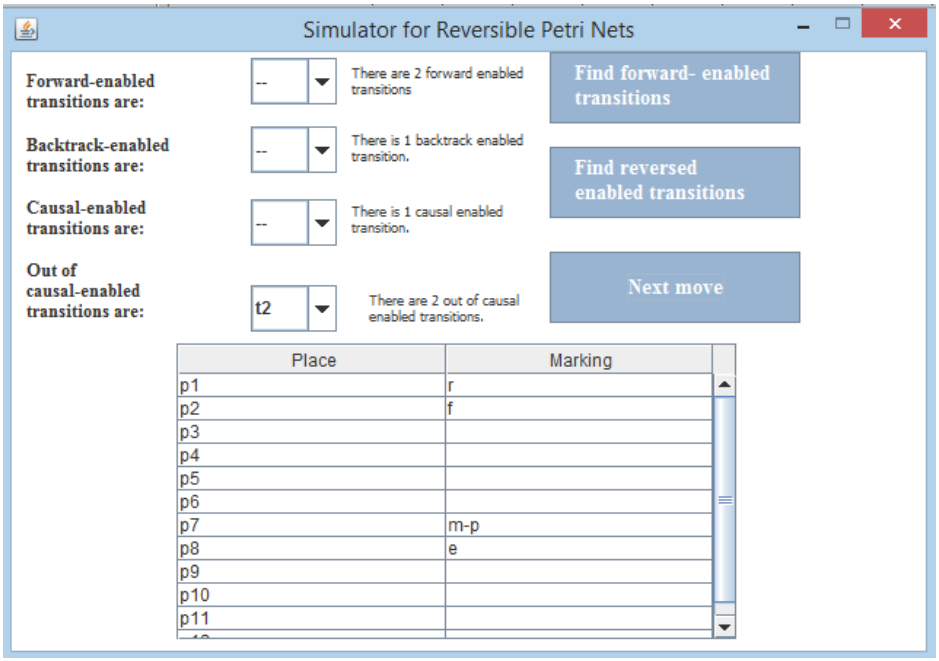



**Figure 5. 17: The new marking after the reverse of the transition t2**

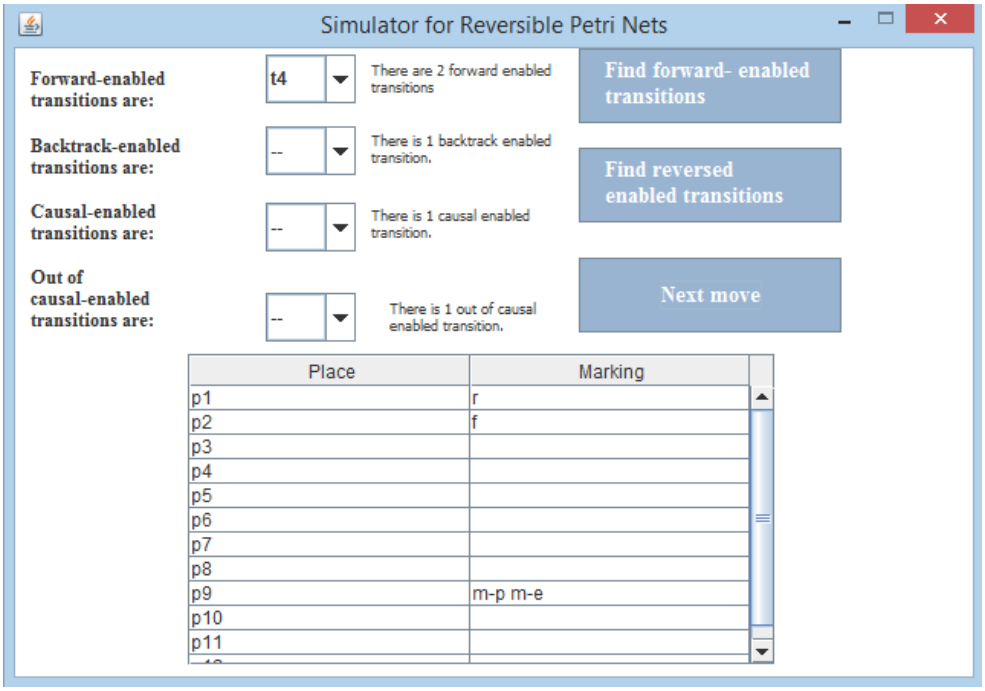



**Figure 5. 18: The new marking after the execution of the transition t4**

From the figures above (Figure 5.15, 5.16, 5.17, 5.18) we can observe that the simulator's output is correct, and after we had reversed the transition t2, we can also observe that the transition t4 is now forward-enabled and we can execute it.

We can also watch the changes that have occurred in the marking through the graphical representation tool. So, in order to check the correctness of the graphical representation tool, we return to its environment to find out if the changes have appeared to the diagram.

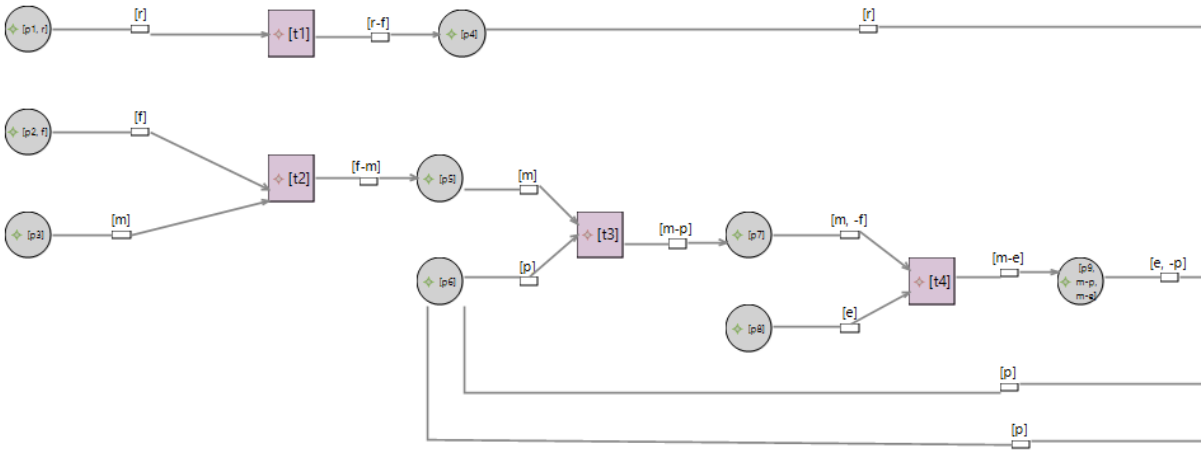



**Figure 5. 19: The changes that appeared on the diagram**

From the figure above (Figure 5.19) we can observe that the diagram is updated and the marking is the same as the one in the simulator.
Thus, with this case study we have checked the correctness of the out-of-causal algorithms and the correctness of the graphical representation tool. Moreover, the simulator's response time was the same as in the previous example. This example was much larger than the previous one, but there was not a change in the response time.
To conclude, the simulator can support large RPNs as effortlessly as small RPNs, without a change in its response time. However, the graphical representation tool is a little bit slow when the user launches it. So, there is some idle time when the user opens the graphical representation tool, but since it opens there are no more delays in its functions.

# Chapter 6

## Conclusions

---

6.1 Summary

6.2 Challenges

6.3 Future Work

---

## 6.1 Summary

In this bachelor thesis an integrated environment has been developed for simulating reversing Petri nets and visualizing their behavior. Firstly an approach was created and then used to implement the mechanisms, namely forward, backtracking, causal-order reversibility, and out-of-causal-order reversibility, in the Java programming language. In order to check the correctness of the algorithms' implementation, many examples have tested, before proceeding to the next steps. Subsequently, a Graphical User Interface was developed for the simulator, in order to give the user the opportunity to easily interact with it and have a better user experience. The simulator enables the user to give the information of a reversing Petri net, with three different ways. The user can choose to give the initial marking of the Petri net either from a file of a specific form, or to create a new Petri net through the simulator environment, or by connecting to the graphical representation tool through the simulator and create a Petri net diagram there. The graphical representation tool has been developed with the Obeo Designer software, which is an Eclipse plugin software. Since the input required has been imported, the user can execute any enabled transition indicated by the simulator step-by-step in both the forward and the reversed direction, and at each step the graphical user interface displays the new marking of the model in the form of a matrix. Furthermore, if the user selects to give the initial marking from the graphical representation tool, an additional feature is available. While the user executes transitions, the diagram definition in the graphical representation tool change too, thus the user can keep track of new created markings. Moreover, the user can start the simulation of an already existing Petri net,

which has been executed before, by importing it from a file or from the graphical representation tool.

## 6.2 Challenges

Through the whole procedure of this diploma thesis I faced some challenges, due to either problems that occurred, or decisions that had to be made, or just because it was my first major research and I was not aware of some basic concepts.

Initially, I had to devote time to fully understand the basic concepts of the subject and ensure that I am ready to continue with the implementation. In this introductory step a lot of articles have been studied related to the subject, which have been selected carefully, and they helped me to comprehend reversing Petri nets and their main features.

There were also difficulties in selecting the right structures to represent the Reversing Petri net's elements, and after consideration, it was decided to represent each element by a Java object, and their relationships between them are represented with lists of elements where needed. The lists are implemented with ArrayList structure, provided by Java, with the corresponding type of element each time. In addition when I had to transfer the algorithms from their mathematical definition to code in the Java programming language, some difficulties occurred that were overcome.

At a subsequent phase of the project, when the graphical representation tool had to be designed and implemented, two significant challenges have encountered. Firstly, after the creation of the Class diagram, in the design specifications of the model, due to the fact that the software used the relationships defined in the Class diagram, there was no way to represent arcs with a continuous directed arrow, and be identified from the software as an object. Thus, after a thorough research for a solution, the most suitable was to create the Arc object individually, to be able for the software to identify it as an element, and then from there create the relationships between places and transitions.

Another difficulty I faced with the development of the graphical representation tool was that after its creation, it was realized that there was not an option to make the project executable, and can be run from the simulator environment as a Runnable API from the simulator. The solution found was to add a button on the Graphical User Interface,

which launch the Reversing Petri net's project in Obeo Designer software, and with the execution of a few steps, the user can create the diagram of the initial marking.

Generally, the tool thas has been created meets the requirements that were specified at the first phase of the project. The simulator simulates the behavior of the reversing Petri nets, with an automated way and there is no need for the user to make any calculations. Moreover it is efficient, because even for large examples, its response time does not exceed 10-15 seconds. The graphical user interface is simple and friendly, so the user can remember its functions easily, and feel comfortable when she/he uses it. However, due to the difficulties that have been mentioned above, the simulator has some weakenesses as well. The first weakness is that there is not a direct connection with the graphical representation tool, and the user needs to do some actions before he/she will be able to draw the initial marking. The necessary actions may confuse the user and take him/her time to accomplish the desired action. Furthermore, the representation of the arcs in the graphical representation tool is different from the one which is know, so maybe the users do not understand how to create an arc through the tool.

## 6.3 Future work

Reversing Petri nets are an appealing and interesting concept, which is now developing and can be used in many applications because they are easily be mechanized. Although a simulator has been developed with a graphical representation tool, in this Bachelor thesis, some limitations have occurred. Due to these limitations, mainly with the representation of the arc in the graphical representation tool, in a subsequent research another software to be used may be found for the development of the tool, or a solution to the problem that did not came across in this project may mitigate this limitation.

Furthermore, the simulator can be expanded and be able to support some extended forms of Reversing Petri nets, such as Coloured Petri nets [3], or Cyclic Petri nets, which is a concept that will appear in [14].

# References


[1] (2019). *Obeo Designer : The Professional Solution to Deploy Sirius* . Available: https://www.obeodesigner.com/en/product/key-features.

[2] H. B. Axelsen and R. Glück, "What do reversible programs compute?" in *International Conference on Foundations of Software Science and Computational Structures,* pp.42-56*,* 2011.

[3] K. Barylska *et al*, "Reversing computations modelled by coloured petri nets." in *ATAED@ Petri Nets/ACSD,* pp.91-111, 2018 .

[4] J. Desel and W. Reisig, "Place/transition petri nets," in *Advanced Course on Petri Nets*, pp.122-173,1996.

[5] J. Krivine and J. Stefani, "Reversible computation," in Anonymous Conference, pp.113-115, 2015.

[6] *Java documentation*. Available: https://docs.oracle.com/en/java/,2018.

[7] K. S. Perumalla, "Introduction to reversible computing," in Anonymous Conference, pp.3-7, 2013.

[8] A. Philippou and K. Psara, "Reversible computation in Petri nets," in *International Conference on Reversible Computation,*pp.84-101 *,* 2018.

[9] K. Psara, "Reversible Computation in Formal Models of Concurrency,"unpublished.

[10] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies.* 2013.

[11] *All You Need to Know About UML Diagrams*. Available: https://tallyfy.com/uml-diagram/, 2019.

[12] C. Thachuk, "Logically and physically reversible natural computing: A tutorial," in *International Conference on Reversible Computation,*pp.247-262*,* 2013.

[13] K. Psara and A. Philippou, "Out-of-causal Order," unpublished.

[14] K. Psara and A. Philippou, "Reversible Computation in Cyclic Petri nets," unpublished.

# Appendix A

## Structures for main components in Java

### Place

```
import java.util.ArrayList;

public class Place {
	String place_name;
	int place_id;
	ArrayList<Token> tokens;
	ArrayList<Token>bonds;
	ArrayList<Integer> arc_id;
	boolean empty;

Place(){
	this.place_name="";
	this.place_id=0;
	this.tokens=new ArrayList<Token>();
	this.bonds=new ArrayList<Token>();
	this.empty=false;
	this.arc_id=new ArrayList<Integer>();
}

}
```

### Transition

```
import java.util.ArrayList;

public class Transition {
String transition_name;
int transition_id;
boolean enabled_for_execution;
boolean backtrack_enable;
boolean co_enabled;
boolean o_enabled;
int num_of_input;
int num_of_output;
ArrayList<Integer> arc_id;


Transition(){
	this.transition_name="";
	this.transition_id=0;
	this.enabled_for_execution=false;
	this.backtrack_enable=false;
	this.co_enabled=false;
	this.num_of_input=0;
	this.num_of_output=0;
	this.o_enabled=false;
	this.arc_id=new ArrayList<Integer>();

}
}
```

## Arc

```
import java.util.ArrayList;

public class Connection {
	int connection_id;
	Place place;
	Transition transition;
	ArrayList <Token>tokens;
	ArrayList <Token> bonds;
	ArrayList<Token> negative_tokens;
	ArrayList<Token> negative_bonds;
	char from;
	char to;

Connection(){
	this.connection_id=0;
	this.place=new Place();
	this.transition=new Transition();
	this.tokens=new ArrayList<Token>();
	this.from='\0';
	this.to='\0';
	this.bonds=new ArrayList<Token>();
	this.negative_bonds=new ArrayList<Token>();
	this.negative_tokens=new ArrayList<Token>();

}
}
```

## Token/Bond

```
public class Token {
	String name;
	int id=0;
	public static int e=0;

Token(){
this.name="";
e++;
id=e;

}
}
```

## Cell (History representation)

```
public class Cell {
	Transition tr;
	int history;

	Cell() {
		this.history = 0;
	}
}
```

## Petri net

```
import java.util.ArrayList;

public class PetriNet {
	ArrayList<Place> places;
	ArrayList<Transition> transitions;
	ArrayList<Connection> arcs;
	ArrayList<Token> tokens;
	ArrayList<Cell> history;
	Transition last_executed;

	PetriNet() {
		places = new ArrayList<Place>();
		transitions = new ArrayList<Transition>();
		arcs = new ArrayList<Connection>();
		tokens = new ArrayList<Token>();
		history = new ArrayList<Cell>();
		last_executed = new Transition();
	}

}
```

# Appendix B

## Algorithms' implementation in Java

## Forward algorithm functions

```
import java.io.IOException;
import java.util.ArrayList;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.xml.sax.SAXException;

public class Forward_algorithm {

	public static PetriNet initial_petrinet;
	public static ArrayList<PetriNet> forward_moves = new
ArrayList<PetriNet>();

/**
 * This function indicates by which of the three methods the
 * user has imported the input, and depends on the method, it
 * initializes the initial marking.
 * @throws IOException
 * @throws ParserConfigurationException
 * @throws SAXException
 */
public static void find_choice() throws IOException,
ParserConfigurationException, SAXException {

		if (MainWindow.choice == 1) {
			initial_petrinet = MainWindow.petri;
		} else if (MainWindow.choice == 2) {
			initial_petrinet = readUser2.petri;
		} else if (MainWindow.choice == 3) {
			Parser.mainF();
			initial_petrinet = Parser.initial_petrinet;
		}
		PetriNet newpetri = initial_petrinet;

		forward_moves.add(newpetri);
	}

/**
 * This method takes as parameter a Petri net structure,
 * and finds out which transitions of the given Petri net
 * are forward-enabled.
 * @param petri
 */
```

```
public static void forward_enabled(PetriNet petri) {

ArrayList<Connection> to_transition = new ArrayList<Connection>();
      for (int i = 0; i < petri.transitions.size(); i++) {
            petri.transitions.get(i).enabled_for_execution = false;
      }
      for (int i = 0; i < petri.transitions.size(); i++) {
            boolean checker = false;
            boolean f = false;
            boolean nc = false;

            for (int j = 0; j < petri.arcs.size(); j++) {
                  if(petri.arcs.get(j).transition.transition_name.
            compareTo(petri.transitions.get(i).transition_name)== 0 &&
            (petri.arcs.get(j).to == 't')) {
                  to_transition.add(petri.arcs.get(j));
                  }
            }

            for (int k = 0; k < to_transition.size(); k++) {
            if (to_transition.get(k).bonds.size() == 0) {
                  if (to_transition.size() == 1) {
                        f = false;
                  if (to_transition.get(k).place.empty == true) {
                        checker = false;
                  }

for (int h = 0; h < to_transition.get(k).place.tokens.size(); h++) {
      for (int s = 0; s < to_transition.get(k).tokens.size(); s++) {

      if (to_transition.get(k).place.tokens.get(h).name
            .compareTo(to_transition.get(k).tokens.get(s).name) == 0){
                        checker = true;
            }
      }
}

if (to_transition.get(k).negative_tokens.size() > 0) {
for (int h = 0; h < to_transition.get(k).place.tokens.size(); h++) {
for (int s = 0; s < to_transition.get(k).negative_tokens.size(); s++)
{
if (to_transition.get(k).place.tokens.get(h).name
.compareTo(to_transition.get(k).negative_tokens.get(s).name) == 0) {
      checker = false;
                  }
            }
      }
}

} else if (to_transition.size() > 1) {
            f = true;
            boolean check[] = new boolean[to_transition.size()];

      for (int g = 0; g < check.length; g++) {
            check[g] = false;
      }
      for (int h = 0; h < to_transition.size(); h++) {
            nc = true;
      if (to_transition.get(h).place.bonds.size() == 0) {
      for (int j = 0; j < to_transition.get(h).tokens.size(); j++) {
```

```
      for (int y = 0;y <to_transition.get(h).place.tokens.size();y++)
      {
            if (to_transition.get(h).tokens.size()!=
                  to_transition.get(h).place.tokens.size()) {
                  nc = false;
                  break;
} else if (to_transition.get(h).tokens.get(j).name
      .compareTo(to_transition.get(h).place.tokens.get(y).name) == 0)
{

      check[h] = true;

            }
      }
}

if (to_transition.get(h).negative_tokens.size() > 0) {
for (int g = 0; g < to_transition.get(h).place.tokens.size(); g++) {
for (int j = 0; j < to_transition.get(h).negative_tokens.size(); j++)
{
      if (to_transition.get(h).negative_tokens.get(j).name
      .compareTo(to_transition.get(h).place.tokens.get(g).name) == 0)
{

            nc = false;
            break;
                  }
            }
      }
}

if (nc != false && h == to_transition.size() - 1) {
      for (int r = 0; r < check.length; r++) {
            if (check[r] != true) {
                  nc = false;
            }
      }
}
} else if (to_transition.get(h).place.bonds.size() != 0) {
      for (int j = 0; j < to_transition.get(h).tokens.size(); j++) {
            if (to_transition.get(h).place.bonds.get(j).name
                  .contains(to_transition.get(h).tokens.get(j).name))
{

                  check[h] = true;
      }
}

if (to_transition.get(h).negative_bonds.size() > 0) {
for (int g = 0; g < to_transition.get(h).place.bonds.size(); g++) {
for (int j = 0; j < to_transition.get(h).negative_bonds.size(); j++) {
if (to_transition.get(j).place.bonds.get(h).name.
compareTo(to_transition.get(h).negative_bonds.get(g).name) == 0) {

      nc = false;
      break;
                  }
            }
      }
}
```

```
if (to_transition.get(h).negative_tokens.size() > 0) {
for (int g = 0; g < to_transition.get(h).place.tokens.size(); g++) {
for (int j = 0; j < to_transition.get(h).negative_tokens.size(); j++)
{
if (to_transition.get(j).place.tokens.get(h).name
.compareTo(to_transition.get(h).negative_tokens.get(g).name) == 0) {

      nc = false;
      break;
                        }
                  }
            }
}

if (nc != false && h < to_transition.size() - 1) {
      for (int r = 0; r < check.length; r++) {
            if (check[r] != true) {
                  nc = false;
                                    }
                              }
                        }

                  }

            }
      }
} else if (to_transition.get(k).bonds.size() != 0 &&
            to_transition.get(k).place.bonds.size() != 0) {

            nc = true;
for (int h = 0; h < to_transition.get(k).place.bonds.size(); h++) {
      for (int s = 0; s < to_transition.get(k).bonds.size(); s++) {
            if (to_transition.get(k).place.bonds.get(h).name
            .compareTo(to_transition.get(k).bonds.get(s).name) == 0) {

                        checker = true;
                        }
                  }

      }
if (to_transition.get(k).bonds.size() != 0 &&
      to_transition.get(k).place.bonds.size() == 0) {

            checker = false;
      }

if (to_transition.get(k).negative_bonds.size() > 0) {
for (int g = 0; g < to_transition.get(k).place.bonds.size(); g++) {
for (int j = 0; j < to_transition.get(k).negative_bonds.size(); j++) {
      if (to_transition.get(j).place.bonds.get(k).name
.compareTo(to_transition.get(k).negative_bonds.get(g).name) == 0) {

                  nc = false;
                  break;
                        }
                  }
            }
      }
```

```
if (to_transition.get(k).negative_tokens.size() > 0) {
for (int j = 0; j < to_transition.get(k).negative_tokens.size(); j++)
{
if (to_transition.get(j).place.tokens.get(k).name
.compareTo(to_transition.get(k).negative_tokens.get(j).name) == 0) {

            nc = false;
            break;
                        }
                  }
            }
      }

}

to_transition.clear();

if (checker == true && f == false) {
      petri.transitions.get(i).enabled_for_execution = true;

} else if (f == true) {
      if (nc == false) {

      petri.transitions.get(i).enabled_for_execution = false;

      } else {

      petri.transitions.get(i).enabled_for_execution = true;
                  }

            }

      }

}

/**
 * This function takes as parameter a String value, from the GUI,
which
 * indicates the transition name of the transition that the user wants
to
 * execute, and then executes this transition, by distribute the
tokens or
 * bonds in the output places of the transition.
 * @param sel
 * @return
 * @throws ParserConfigurationException
 * @throws TransformerException
 */
public static ArrayList<PetriNet> forward_execution(String sel)
            throws ParserConfigurationException, TransformerException
{

      ArrayList<Place> input_places = new ArrayList<Place>();
      ArrayList<Place> output_places = new ArrayList<Place>();
      ArrayList<Connection> out_arcs = new ArrayList<Connection>();
      ArrayList<Connection> in_arcs = new ArrayList<Connection>();

      ArrayList<String[]> add1 = new ArrayList<String[]>();
      ArrayList<String[]> remove1 = new ArrayList<String[]>();
```

```
for (int i = 0; i < forward_moves.size(); i++) {
for (int j = 0; j < forward_moves.get(i).transitions.size(); j++) {


if (forward_moves.get(i).transitions.get(j).enabled_for_execution ==
true && forward_moves.get(i).transitions.get(j).transition_name.
      compareTo(sel) == 0) {

      for (int k = 0; k < forward_moves.get(i).arcs.size(); k++) {
      if (forward_moves.get(i).arcs.get(k).to == 't' &&
            forward_moves.get(i).arcs.get(k).transition
                  .equals(forward_moves.get(i).transitions.get(j))) {

      input_places.add(forward_moves.get(i).arcs.get(k).place);

      in_arcs.add(forward_moves.get(i).arcs.get(k));

      } else if (forward_moves.get(i).arcs.get(k).to == 'p'
                  && forward_moves.get(i).arcs.get(k).transition
                  .equals(forward_moves.get(i).transitions.get(j))) {

      output_places.add(forward_moves.get(i).arcs.get(k).place);

      out_arcs.add(forward_moves.get(i).arcs.get(k));
            }
      }

      for (int w = 0; w < out_arcs.size(); w++) {

      if (out_arcs.get(w).tokens.size() != 0 &&
                  out_arcs.get(w).bonds.size() == 0) {

      for (int d = 0; d < out_arcs.get(w).tokens.size(); d++) {
      for (int q = 0; q < input_places.size(); q++) {
      for (int s = 0; s < input_places.get(q).tokens.size(); s++) {
            if (out_arcs.get(w).tokens.get(d).name
            .compareTo(input_places.get(q).tokens.get(s).name) == 0) {

            String s1[] = new String[2];

            s1[0] = out_arcs.get(w).place.place_name;

            s1[1] = input_places.get(q).tokens.get(s).name;

            add1.add(s1);


out_arcs.get(w).place.tokens.add(input_places.get(q).tokens.get(s));


      String s2[] = new String[2];

      s2[0] = input_places.get(q).place_name;

      s2[1] = input_places.get(q).tokens.get(s).name;

      remove1.add(s2);

input_places.get(q).tokens.remove(input_places.get(q).tokens.get(s));
```

```
                        }
                    }
                }

        }

} else if(out_arcs.get(w).bonds.size() != 0 && input_places.size()> 1)
{
        boolean check1 = false;
        boolean sw = true;
        for (int y = 0; y < out_arcs.get(w).bonds.size(); y++) {
            for (int d = 0; d < input_places.size(); d++) {

                for (int x = 0; x < input_places.size(); x++) {
                    if (input_places.get(x).bonds.size() != 0) {
                        sw = false;
                        }
                }

if (input_places.get(d).bonds.size() == 0 && sw == true) {

        for (int h = 0; h < input_places.get(d).tokens.size(); h++) {
            for (int s = 0; s < in_arcs.size(); s++) {

            for (int r = 0; r < in_arcs.get(s).tokens.size(); r++) {

            if (input_places.get(d).tokens.get(h).name
            .compareTo(in_arcs.get(s).tokens.get(r).name) == 0) {

                check1 = true;

            } else {

                check1 = false;

            }

        }

}
}
if (check1 == true) {

String s1[] = new String[2];

s1[0] = out_arcs.get(w).place.place_name;

s1[1] = out_arcs.get(w).bonds.get(y).name;

add1.add(s1);

out_arcs.get(w).place.bonds.add(out_arcs.get(w).bonds.get(y));

for (int r = 0; r < input_places.size(); r++) {
        for (int h = 0; h < input_places.get(r).tokens.size(); h++) {

            String s2[] = new String[2];

            s2[0] = input_places.get(r).place_name;
```

```
                s2[1] = input_places.get(r).tokens.get(h).name;

                remove1.add(s2);

input_places.get(r).tokens.remove(input_places.get(r).tokens.get(h));

                        }
                }
        }

} else if (input_places.get(d).bonds.size() != 0 && sw == false) {

        String name = "";

        ArrayList<Token> hold = new ArrayList<Token>();

        for (int s = 0; s < in_arcs.size(); s++) {
        if (in_arcs.get(s).bonds.size() == 0 &&
        in_arcs.get(s).tokens.size() != 0) {

        boolean check = false;

        String equal = "";

        String equal1 = "";

        for (int h = 0; h < input_places.get(d).bonds.size(); h++) {

                name = input_places.get(d).bonds.get(h).name;

                String splitted[] = name.split("-");

                String left = splitted[0];

                String right = splitted[1];

                equal = left;

                equal1 = right;

                for (int r = 0; r < in_arcs.get(s).tokens.size(); r++) {

        if (left.compareTo(in_arcs.get(s).tokens.get(r).name) == 0) {

                                check = true;


        } else if (right.compareTo(in_arcs.get(s).tokens.get(r).name)
                        == 0) {

                        check = true;


                        }

                }

        }
```

```
if (check == true) {

for (int u = 0; u < input_places.size(); u++) {
      for (int r = 0; r < input_places.get(u).bonds.size(); r++)
      {

if (input_places.get(u).bonds.get(r).name.contains(equal)

|| input_places.get(u).bonds.get(r).name.contains(equal1)) {

      hold.add(input_places.get(u).bonds.get(r));


            }

      }

}


for (int u = 0; u < hold.size(); u++) {

      String s1[] = new String[2];

      s1[0] = out_arcs.get(w).place.place_name;

      s1[1] = hold.get(u).name;

      add1.add(s1);

      out_arcs.get(w).place.bonds.add(hold.get(u));

}

hold.clear();


if (!out_arcs.get(w).place.bonds

      .contains(out_arcs.get(w).bonds.get(y))) {


      String s1[] = new String[2];

      s1[0] = out_arcs.get(w).place.place_name;

      s1[1] = out_arcs.get(w).bonds.get(y).name;

      add1.add(s1);


out_arcs.get(w).place.bonds.add(out_arcs.get(w).bonds.get(y));

            }

      }

} else if (in_arcs.get(s).bonds.size() != 0
```

```
            && in_arcs.get(s).tokens.size() == 0) {
                  boolean check2 = false;
      for (int h = 0; h < input_places.get(d).bonds.size(); h++) {
            for (int r = 0; r < in_arcs.get(s).bonds.size(); r++) {
                  if (input_places.get(d).bonds.get(d).name
                  .compareTo(in_arcs.get(s).bonds.get(r).name) == 0) {
                        check2 = true;
                  }
            }
      }
      if (check2 == true) {
            String s1[] = new String[2];
            s1[0] = out_arcs.get(w).place.place_name;
            s1[1] = out_arcs.get(w).bonds.get(y).name;
            add1.add(s1);

      out_arcs.get(w).place.bonds.add(out_arcs.get(w).bonds.get(y));
            for (int l = 0; l < input_places.size(); l++) {
            for (int h = 0; h < input_places.get(l).bonds.size(); l++)
            {
                        String s2[] = new String[2];
                        s2[0] = input_places.get(l).place_name;
                        s2[1] = input_places.get(l).bonds.get(h).name;
                        remove1.add(s2);
input_places.get(l).bonds.remove(input_places.get(l).bonds.get(h));
                                    }
                              }
                        }
                  }
            }
      }
```

```
}

for (int l = 0; l < input_places.size(); l++) {
      for (int g = 0; g < input_places.get(l).bonds.size(); g++) {
            String f[] = new String[2];
            f[0] = input_places.get(l).place_name;
            f[1] = input_places.get(l).bonds.get(g).name;

            remove1.add(f);
      }

      input_places.get(l).bonds.removeAll(input_places.get(l).bonds);

      for (int g = 0; g < input_places.get(l).tokens.size(); g++) {
            String f[] = new String[2];
            f[0] = input_places.get(l).place_name;
            f[1] = input_places.get(l).tokens.get(g).name;

            remove1.add(f);
      }

input_places.get(l).tokens.removeAll(input_places.get(l).tokens);

      }

}

// if out_arc contains bonds but it has only one input place

}else if (out_arcs.get(w).bonds.size() != 0&& input_places.size()== 1)
{
      for (int d = 0; d < out_arcs.get(w).bonds.size(); d++) {
            for (int s = 0; s < input_places.get(0).bonds.size(); s++)
            {
            if (input_places.get(0).bonds.get(0).name
                  .compareTo(out_arcs.get(w).bonds.get(d).name) == 0)
            {

                  String s2[] = new String[2];
                  s2[0] = out_arcs.get(w).place.place_name;
                  s2[1] = input_places.get(0).bonds.get(d).name;
                  add1.add(s2);

out_arcs.get(w).place.bonds.add(input_places.get(0).bonds.get(d));

String s3[] = new String[2];
s3[0] = input_places.get(0).place_name;
s3[1] = input_places.get(0).bonds.get(d).name;
remove1.add(s3);

input_places.get(0).bonds.remove(input_places.get(0).bonds.get(d));

                        }

                  }
            }
      }

}
// history update
```

```
forward_moves.get(i).last_executed=forward_moves.get(i)
      .transitions.get(j);

forward_moves.get(i).transitions.get(j).enabled_for_execution = false;
if (j > 0) {
forward_moves.get(i).history.get(j).history =
      forward_moves.get(i).history.get(j - 1).history
                                                + 1;
} else if (j == 0) {
      forward_moves.get(i).history.get(j).history = 1;

      }
}
input_places.clear();
output_places.clear();
out_arcs.clear();
in_arcs.clear();


if (MainWindow.choice == 3) {
      try {

      ReverseParser.update_xml(forward_moves.get(0), add1, remove1);
      } catch (SAXException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
                              }
                        }
                  }

            }

            return forward_moves;

      }

}
```

## Backtrack algorithm functions

```
import java.io.IOException;
import java.util.ArrayList;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.xml.sax.SAXException;

public class Backtracking_algorithm {

public static Transition bt_enabled = new Transition();

/**
* This function takes as parameter a Petri net and finds which
* transition is bt-enabled, if there is one. Then it returns the
* maximum history,which belongs to the bt-enabled transition. If there
* is not a bt-enabled transition the function returns -1.
 * @param petri
 * @return int max
 */
public static int backtracking_enabled(PetriNet petri) {
      int max = 0;
      Cell max_cell = new Cell();
      for(int i=0;i<petri.transitions.size();i++){
            petri.transitions.get(i).backtrack_enable=false;
      }

      for (int i = 0; i < petri.history.size(); i++) {
            if (petri.history.get(i).history > max) {
                  max = petri.history.get(i).history;
                  max_cell = petri.history.get(i);
            }
      }

      if (max_cell.tr != null) {
            for (int i = 0; i < petri.transitions.size(); i++) {
                  if (petri.transitions.get(i).transition_name.
                  compareTo(max_cell.tr.transition_name) == 0) {

                  petri.transitions.get(i).backtrack_enable = true;
                  break;
                  }
            }
      } else {
            max = 0;
      }

      return max;
      }

      /**
```

```
        * This function takes as parameter a String value, from the
        * GUI, which indicates the transition name of the transition
        * that the user wants to execute, and then executes this
        * transition in a backtracking fashion.
        * @param sel
        * @return Petri net
        */
public static ArrayList<PetriNet> backtrack_execution(String sel) {

      ArrayList<Connection> in_arcs = new ArrayList<Connection>();
      ArrayList<Connection> out_arcs = new ArrayList<Connection>();
      ArrayList<Place> input_places = new ArrayList<Place>();
      ArrayList<Place> out_places = new ArrayList<Place>();

      ArrayList<String[]>add1=new ArrayList<String[]>();
      ArrayList<String[]>remove1=new ArrayList<String[]>();


for (int i = 0; i < Forward_algorithm.forward_moves.size(); i++) {

for (int j = 0; j < Forward_algorithm.forward_moves.get(i)
                  .transitions.size(); j++) {
      if (Forward_algorithm.forward_moves.get(i).transitions
      .get(j).backtrack_enable == true
      &&(Forward_algorithm.forward_moves.get(i).transitions.get(j).
      transition_name.compareTo(sel)==0)) {
      bt_enabled = Forward_algorithm.forward_moves.get(i).
                  transitions.get(j);
                  break;
            }
      }

for (int j = 0; j <Forward_algorithm.forward_moves.get(i).arcs.size();
j++) {

if ((Forward_algorithm.forward_moves.get(i).arcs.get(j).
transition.transition_name.compareTo(bt_enabled.transition_name) == 0)
&& Forward_algorithm.forward_moves.get(i).arcs.get(j).to == 't') {

in_arcs.add(Forward_algorithm.forward_moves.get(i).arcs.get(j));

      input_places.add(Forward_algorithm.forward_moves.get(i).arcs.get
(j).place);
} else if ((Forward_algorithm.forward_moves.get(i).arcs.get(j).
transition.transition_name.compareTo(bt_enabled.transition_name) == 0)
&& Forward_algorithm.forward_moves.get(i).arcs.get(j).from == 't') {

out_arcs.add(Forward_algorithm.forward_moves.get(i).arcs.get(j));
out_places.add(Forward_algorithm.forward_moves.get(i).arcs.get(j).plac
e);
      }
}

for (int k = 0; k < out_arcs.size(); k++) {
if (out_arcs.get(k).bonds.size() != 0 && out_arcs.get(k).tokens.size()
== 0) {
      ArrayList<Token> bond = new ArrayList<Token>();
      boolean mode=false;
      for (int w = 0; w < out_arcs.get(k).bonds.size(); w++) {
      for (int j = 0; j < out_arcs.get(k).place.bonds.size(); j++) {
```

```
            if (out_arcs.get(k).place.bonds.get(j).name
            .compareTo(out_arcs.get(k).bonds.get(w).name) == 0) {

String breakb[] = out_arcs.get(k).place.bonds.get(j).name.split("-");
String left = breakb[0];
String right = breakb[1];

if (out_arcs.get(k).place.bonds.size() > 1) {
      mode=true;

for (int o = 0; o < out_arcs.get(k).place.bonds.size(); o++) {
if (out_arcs.get(k).place.bonds.get(o).name.contains(left) &&
out_arcs.get(k).place.bonds.get(o).name.compareTo(out_arcs.get(k).bond
s.get(w).name)!=0) {

      bond.add(out_arcs.get(k).place.bonds.get(o));

      String s[]=new String[2];

      s[0]=out_arcs.get(k).place.place_name;

      s[1]=out_arcs.get(k).place.bonds.get(o).name;

      remove1.add(s);



out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;
} else if (out_arcs.get(k).place.bonds.get(o).name.contains(right) &&
out_arcs.get(k).place.bonds.get(o).name.compareTo(out_arcs.get(k).bond
s.get(w).name)!=0) {

      bond.add(out_arcs.get(k).place.bonds.get(o));

      String s[]=new String[2];

      s[0]=out_arcs.get(k).place.place_name;

      s[1]=out_arcs.get(k).place.bonds.get(o).name;

      remove1.add(s);

out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;
      }
}
      String s1[]=new String[2];

      s1[0]=out_arcs.get(k).place.place_name;

      s1[1]=out_arcs.get(k).bonds.get(w).name;

      remove1.add(s1);

out_arcs.get(k).place.bonds.remove(out_arcs.get(k).bonds.get(w));

for (int g = 0; g < bond.size(); g++) {
      for (int o = 0; o < out_arcs.get(k).place.bonds.size(); o++) {
```

```
		String gg[] = bond.get(g).name.split("-");

		if (out_arcs.get(k).place.bonds.get(o).name.contains(gg[0])) {

			bond.add(out_arcs.get(k).place.bonds.get(o));

			String s2[]=new String[2];

			s2[0]=out_arcs.get(k).place.place_name;

			s2[1]=out_arcs.get(k).place.bonds.get(o).name;

			remove1.add(s2);

out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;

} else if (out_arcs.get(k).place.bonds.get(o).name.contains(gg[1])) {

			bond.add(out_arcs.get(k).place.bonds.get(o));

			String s3[]=new String[2];

			s3[0]=out_arcs.get(k).place.place_name;

			s3[1]=out_arcs.get(k).place.bonds.get(o).name;

			remove1.add(s3);

out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;

				}
			}
		}
}//end of >1 bonds in the place
else if(out_arcs.get(k).place.bonds.size() ==1){

		mode=false;

		bond.clear();
				}

			}
		}
}

if(mode==true){
		for (int z = 0; z < in_arcs.size(); z++) {
			boolean check = false;
			for (int g = 0; g < bond.size(); g++) {
				for (int f = 0; f < in_arcs.get(z).tokens.size();
				f++) {
					if (bond.get(g).name.contains(in_arcs.get(z).
					tokens.get(f).name)) {

						check = true;
						break;
					} else {
```

```
                                    check = false;
                              }
                        }
                  if(check==true){
                        break;
                  }
            }
      if (check == true) {
            for (int g = 0; g < bond.size(); g++) {
                  String s[]=new String[2];

                  s[0]=in_arcs.get(z).place.place_name;

                  s[1]=bond.get(g).name;
                  add1.add(s);

                  in_arcs.get(z).place.bonds.add(bond.get(g));
            }
      } else {
            for (int f = 0; f < in_arcs.get(z).tokens.size(); f++) {

            String s[]=new String[2];

            s[0]=in_arcs.get(z).place.place_name;

            s[1]=in_arcs.get(z).tokens.get(f).name;
            add1.add(s);

            in_arcs.get(z).place.tokens.add(in_arcs.get(z).tokens.get(f));
                              }
                        }
                  }
            }

            else{
                  for (int z = 0; z < in_arcs.size(); z++) {
                        for (int f = 0; f < in_arcs.get(z).tokens.size();
                        f++) {

                              String s[]=new String[2];

                              s[0]=in_arcs.get(z).place.place_name;

                              s[1]=in_arcs.get(z).tokens.get(f).name;

                              add1.add(s);

            in_arcs.get(z).place.tokens.add(in_arcs.get(z).tokens.get(f));
                              }
                  }

                  for(int d=0;d<out_arcs.get(k).place.bonds.size();d++){

                        String s[]=new String[2];

                        s[0]=out_arcs.get(k).place.place_name;

                        s[1]=out_arcs.get(k).place.bonds.get(d).name;
                        remove1.add(s);
                  }
```

```
out_arcs.get(k).place.bonds.removeAll(out_arcs.get(k).place.bonds);
      }

} else if (out_arcs.get(k).bonds.size() == 0 &&
            out_arcs.get(k).tokens.size() != 0) {

      for (int w = 0; w < out_arcs.get(k).tokens.size(); w++) {
      for (int j = 0; j < out_arcs.get(k).place.tokens.size(); j++) {

            String[]s6=new String[2];

            s6[0]=out_arcs.get(k).place.place_name;

            s6[1]=out_arcs.get(k).tokens.get(w).name;

            remove1.add(s6);


out_arcs.get(k).place.tokens.remove(out_arcs.get(k).tokens.get(w));
            }
      }
      for (int z = 0; z < in_arcs.size(); z++) {
            for (int r = 0; r < in_arcs.get(z).tokens.size(); r++) {

            String[]s8=new String[2];

            s8[0]=in_arcs.get(z).place.place_name;

            s8[1]=in_arcs.get(z).tokens.get(r).name;

            add1.add(s8);

      in_arcs.get(z).place.tokens.add(in_arcs.get(z).tokens.get(r));
                        }
                  }

            }
      }

      bt_enabled.backtrack_enable = false;
      out_arcs.clear();
      in_arcs.clear();
      input_places.clear();
      out_places.clear();

for (int j = 0; j < Forward_algorithm.forward_moves.
                                          get(i).history.size(); j++)
{

if (Forward_algorithm.forward_moves.get(i).
      history.get(j).tr.transition_name.
      compareTo(bt_enabled.transition_name) == 0) {

Forward_algorithm.forward_moves.get(i).history.get(j).history = 0;
break;
      }
}
```

```
        bt_enabled = null;
        if(MainWindow.choice==3){
        try {
        ReverseParser.update_xml(Forward_algorithm.forward_moves.get(0),
        add1, remove1);
        } catch (ParserConfigurationException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        } catch (SAXException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        } catch (TransformerException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
                                }
                        }


                }

              return Forward_algorithm.forward_moves;
        }

}
```

## Causal algorithm functions

```
import java.io.IOException;
import java.util.ArrayList;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.xml.sax.SAXException;

public class Causal_reversing {
/**
* This method takes as parameter a Petri net structure,
* and finds out which transitions of the given Petri net
* are co-enabled.
* @param petri
* @return
 */
public static int co_enabled(PetriNet petri) {

	ArrayList<Transition> executed = new ArrayList<Transition>();
	int counter = 0;

	// check which transitions have been executed (history!=0) and
	// store them in an ArrayList

		for (int i = 0; i < petri.transitions.size(); i++) {
			petri.transitions.get(i).co_enabled = false;
		}

		for (int i = 0; i < petri.history.size(); i++) {
			if (petri.history.get(i).history != 0) {
				executed.add(petri.history.get(i).tr);
			}
		}

	for (int i = 0; i < executed.size(); i++) {
	ArrayList<Connection> in_arcs = new ArrayList<Connection>();
	ArrayList<Connection> out_arcs = new ArrayList<Connection>();
	boolean can = false;
	for (int j = 0; j < petri.arcs.size(); j++) {
		if ((petri.arcs.get(j).transition.transition_name.
			compareTo(executed.get(i).transition_name) == 0)
			&& petri.arcs.get(j).to == 't') {

				in_arcs.add(petri.arcs.get(j));

		} else if ((petri.arcs.get(j).transition.transition_name
				.compareTo(executed.get(i).transition_name) ==
				0) && petri.arcs.get(j).to == 'p') {

					out_arcs.add(petri.arcs.get(j));
				}
			}
```

```
      for (int j = 0; j < out_arcs.size(); j++) {
            if (out_arcs.get(j).tokens.size() != 0 &&
                  out_arcs.get(j).bonds.size() == 0) {
      for (int k = 0; k < out_arcs.get(j).tokens.size(); k++) {
      for (int z = 0; z < out_arcs.get(j).place.tokens.size(); z++) {
            if (out_arcs.get(j).place.tokens.size() == 0) {

                  can = false;
                  break;
            } else if (out_arcs.get(j).tokens.get(k).name
            .compareTo(out_arcs.get(j).place.tokens.get(z).name) == 0)
            {
                  can = true;
                        }
                  }
      }


} else if (out_arcs.get(j).bonds.size() != 0 &&
                  out_arcs.get(j).tokens.size() == 0) {

      for (int k = 0; k < out_arcs.get(j).bonds.size(); k++) {
            for (int z = 0; z < out_arcs.get(j).place.bonds.size();
                  z++) {
                  if (out_arcs.get(j).place.bonds.size() == 0) {

                        can = false;
                        break;
                  } else if (out_arcs.get(j).place.bonds.get(z).name
                        .contains(out_arcs.get(j).bonds.get(k).name))
                  {
                        can = true;
                                    }
                              }
                        }
                  }
                  if (can == true) {
                        executed.get(i).co_enabled = true;
                        counter++;
                  } else {
                        executed.get(i).co_enabled = false;
                  }

            }

            return counter;

      }

/**
 * This function takes as parameter a String value, from the GUI,
 * which indicates the transition name of the transition that the user
 * wants to execute, and then executes this transition in a causal
 * order fashion.
 * @param sel
 * @return
 */
```

```
public static ArrayList<PetriNet> causal_execution(String sel) {

      ArrayList<Connection> in_arcs = new ArrayList<Connection>();
      ArrayList<Connection> out_arcs = new ArrayList<Connection>();
            ArrayList<Place> input_places = new ArrayList<Place>();
            ArrayList<Place> out_places = new ArrayList<Place>();
            ArrayList<String[]>add1=new ArrayList<String[]>();
            ArrayList<String[]>remove1=new ArrayList<String[]>();

for (int i = 0; i < Forward_algorithm.forward_moves.size(); i++) {

for (int g = 0; g < Forward_algorithm.forward_moves.get(i).
                  transitions.size(); g++) {

      if (Forward_algorithm.forward_moves.get(i).transitions.get(g).
      co_enabled == true && Forward_algorithm.forward_moves.get(i).
      transitions.get(g).transition_name.compareTo(sel) == 0) {

for (int j = 0; j < Forward_algorithm.forward_moves
                                    .get(i).arcs.size(); j++) {
      if ((Forward_algorithm.forward_moves.get(i)
            .arcs.get(j).transition.transition_name.compareTo(Forward_
            algorithm.forward_moves.get(i).transitions.get(g).
            transition_name) == 0) && Forward_algorithm.forward_moves.
            get(i).arcs.get(j).to == 't') {

      in_arcs.add(Forward_algorithm.forward_moves.get(i).arcs.get(j));

      input_places.add(Forward_algorithm.forward_moves.get(i).arcs.get
      (j).place);

      } else if ((Forward_algorithm.forward_moves.get(i).arcs.get(j)
            .transition.transition_name.compareTo(Forward_algorithm.fo
            rward_moves.get(i).transitions.get(g).transition_name) ==
            0)&& Forward_algorithm.forward_moves.get(i).arcs
                                                .get(j).from == 't') {

      out_arcs.add(Forward_algorithm.forward_moves.get(i).arcs.get(j))
;

      out_places.add(Forward_algorithm.forward_moves.get(i).arcs.get(j
).place);
            }
}


for (int k = 0; k < out_arcs.size(); k++) {
if (out_arcs.get(k).bonds.size() != 0 && out_arcs.get(k).tokens.size()
== 0) {
      ArrayList<Token> bond = new ArrayList<Token>();
      boolean mode = false;
      for (int w = 0; w < out_arcs.get(k).bonds.size(); w++) {
      for (int j = 0; j < out_arcs.get(k).place.bonds.size(); j++) {
      if (out_arcs.get(k).place.bonds.get(j).name
      .compareTo(out_arcs.get(k).bonds.get(w).name) == 0) {

String breakb[] = out_arcs.get(k).place.bonds.get(j).name.split("-");
String left = breakb[0];
String right = breakb[1];
```

```
if (out_arcs.get(k).place.bonds.size() > 1) {
      mode = true;
      for (int o = 0; o < out_arcs.get(k).place.bonds.size(); o++) {

      if (out_arcs.get(k).place.bonds.get(o).name.contains(left)
                  && out_arcs.get(k).place.bonds.get(o).name
                  .compareTo(out_arcs.get(k).bonds.get(w).name) != 0)
      {

            bond.add(out_arcs.get(k).place.bonds.get(o));

            String s[]=new String[2];

            s[0]=out_arcs.get(k).place.place_name;

            s[1]=out_arcs.get(k).place.bonds.get(o).name;

            remove1.add(s);



out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;

      } else if (out_arcs.get(k).place.bonds.get(o).
            name.contains(right)&&
            out_arcs.get(k).place.bonds.get(o).name.compareTo(out_arcs
            .get(k).bonds.get(w).name) != 0)
      {

            bond.add(out_arcs.get(k).place.bonds.get(o));

            String s[]=new String[2];

            s[0]=out_arcs.get(k).place.place_name;

            s[1]=out_arcs.get(k).place.bonds.get(o).name;

            remove1.add(s);



out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;

      }
}

out_arcs.get(k).place.bonds.remove(out_arcs.get(k).bonds.get(w));

for (int t = 0; t < bond.size(); t++) {

      for (int o = 0; o < out_arcs.get(k).place.bonds.size(); o++) {

            String gg[] = bond.get(t).name.split("-");

      if (out_arcs.get(k).place.bonds.get(o).name.contains(gg[0])) {

                  bond.add(out_arcs.get(k).place.bonds.get(o));
```

```
                    String s[]=new String[2];

                    s[0]=out_arcs.get(k).place.place_name;

                    s[1]=out_arcs.get(k).place.bonds.get(o).name;

                    remove1.add(s);


out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;

} else if (out_arcs.get(k).place.bonds.get(o).name.contains(gg[1])) {

      bond.add(out_arcs.get(k).place.bonds.get(o));

      String s[]=new String[2];

      s[0]=out_arcs.get(k).place.place_name;

      s[1]=out_arcs.get(k).place.bonds.get(o).name;

      remove1.add(s);



out_arcs.get(k).place.bonds.remove(out_arcs.get(k).place.bonds.get(o))
;

                    }
              }
      }
} // end of >1 bonds in the place

else if (out_arcs.get(k).place.bonds.size() == 1) {
      mode = false;

      bond.clear();
                              }

                        }
                    }
              }

if (mode == true) {
for (int z = 0; z < in_arcs.size(); z++) {
boolean check = false;

for (int t = 0; t < bond.size(); t++) {
for (int f = 0; f < in_arcs.get(z).tokens.size(); f++) {
if (bond.get(t).name.contains(in_arcs.get(z).tokens.get(f).name)) {

      check = true;

      break;

} else {
```

```
      check = false;
      }
}

if (check == true) {

      break;
      }
}
if (check == true) {
      for (int t = 0; t < bond.size(); t++) {

            String s[]=new String[2];

            s[0]=in_arcs.get(z).place.place_name;

            s[1]=bond.get(t).name;

            add1.add(s);

            in_arcs.get(z).place.bonds.add(bond.get(t));
                                                                  }
} else {
      for (int f = 0; f < in_arcs.get(z).tokens.size(); f++) {

            String s[]=new String[2];

            s[0]=in_arcs.get(z).place.place_name;

            s[1]=in_arcs.get(z).tokens.get(f).name;

            add1.add(s);

            in_arcs.get(z).place.tokens.add(in_arcs.get(z).tokens.get(
            f));
                  }
            }
      }
} else {

      for (int z = 0; z < in_arcs.size(); z++) {
            for (int f = 0; f < in_arcs.get(z).tokens.size(); f++) {

            String s[]=new String[2];

            s[0]=in_arcs.get(z).place.place_name;

            s[1]=in_arcs.get(z).tokens.get(f).name;

            add1.add(s);

      in_arcs.get(z).place.tokens.add(in_arcs.get(z).tokens.get(f));
      }
}

for(int d=0;d<out_arcs.get(k).place.bonds.size();d++){
      String s[]=new String[2];

      s[0]=out_arcs.get(k).place.place_name;
```

```
      s[1]=out_arcs.get(k).place.bonds.get(d).name;

      remove1.add(s);
}

out_arcs.get(k).place.bonds.removeAll(out_arcs.get(k).place.bonds);
      }

} else if (out_arcs.get(k).bonds.size() == 0 &&
                              out_arcs.get(k).tokens.size() != 0) {


      for (int w = 0; w < out_arcs.get(k).tokens.size(); w++) {
      for (int j = 0; j < out_arcs.get(k).place.tokens.size(); j++) {

            String[]s6=new String[2];

            s6[0]=out_arcs.get(k).place.place_name;

            s6[1]=out_arcs.get(k).tokens.get(w).name;

            remove1.add(s6);


out_arcs.get(k).place.tokens.remove(out_arcs.get(k).tokens.get(w));
            }
      }

for (int z = 0; z < in_arcs.size(); z++) {
      for (int r = 0; r < in_arcs.get(z).tokens.size(); r++) {

            String[]s8=new String[2];

            s8[0]=in_arcs.get(z).place.place_name;

            s8[1]=in_arcs.get(z).tokens.get(r).name;

            add1.add(s8);


      in_arcs.get(z).place.tokens.add(in_arcs.get(z).tokens.get(r));
                        }
                  }


            }
      }

      out_arcs.clear();
      in_arcs.clear();
      input_places.clear();
      out_places.clear();

// history update and make transition not co_enabled

for (int y = 0; y < Forward_algorithm.forward_moves.get(i).
                  history.size(); y++) {

if (Forward_algorithm.forward_moves.get(i).history.
```

```
      get(y).tr.transition_name.compareTo(Forward_algorithm.forward_mo
      ves.get(i).transitions.get(g).transition_name) == 0) {

Forward_algorithm.forward_moves.get(i).history.get(y).history = 0;

Forward_algorithm.forward_moves.get(i).transitions.get(g)
                                              .co_enabled = false;
break;
                                    }
                              }

                        }

            }

      if(MainWindow.choice==3){

      try {
      ReverseParser.update_xml(Forward_algorithm.forward_moves.get(0),
      add1, remove1);
      } catch (ParserConfigurationException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      } catch (SAXException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      } catch (TransformerException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
                                    }
                              }

                        }

                        return Forward_algorithm.forward_moves;
            }
      }
```

## Out of causal algorithm functions

```
import java.io.IOException;
import java.util.ArrayList;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;

import org.xml.sax.SAXException;

public class Out_of_causal {
	public static PetriNet initial_marking = new PetriNet();
	/**
	 * This function indicates by which of the three methods the
	 * user has imported the input, and depends on the method, it
	 * initializes the initial marking.
	 */
	public static void find_initial() {

		if (MainWindow.choice == 1) {
			initial_marking = MainWindow.initial_marking;
		} else if (MainWindow.choice == 2) {
			initial_marking = readUser2.initial_marking;
		}
		else if(MainWindow.choice==3){
			initial_marking=Parser.initial_marking;
		}
	}

	/**
	 * /**
	 * This method takes as parameter a Petri net structure,
	 * and finds out which transitions of the given Petri net
	 * are o-enabled.
	 * @param petri
	 */
	public static void o_enabled(PetriNet petri) {
	// any executed transition can be reversed at any time
	// check which transitions have history!=0, then they are
	// o_enabled

	for (int i = 0; i < petri.transitions.size(); i++) {
		for (int j = 0; j < petri.history.size(); j++) {
			if (petri.history.get(j).tr.transition_name.
			compareTo(petri.transitions.get(i).transition_name)
			 == 0) {

		if (petri.history.get(j).history != 0) {
			petri.transitions.get(i).o_enabled = true;
		} else {
			petri.transitions.get(i).o_enabled = false;
				}
			}
```

```
                  }
            }

      } // end of function o_enabled

      /**
       * This function takes as parameter a String value, from the
       * GUI, which
       * indicates the transition name of the transition that the
       * user wants to execute, and then executes
       * this transition, in an out-of-causal-order fashion.
       * @param sel
       * @return Petri net
       * @throws ParserConfigurationException
       * @throws TransformerException
       */
      public static void out_of_causal_execution(String t1) {
            find_initial();
            Transition t = null;

      for (int i = 0; i < Forward_algorithm.forward_moves.get(0).
            transitions.size(); i++) {
            if (Forward_algorithm.forward_moves.get(0).transitions.
                  get(i).transition_name.compareTo(t1) == 0) {


      t = Forward_algorithm.forward_moves.get(0).transitions.get(i);
      break;
            }
      }

      // H(t)=0
      for (int i = 0; i < Forward_algorithm.forward_moves.get(0)
                        .history.size(); i++) {
            if (Forward_algorithm.forward_moves.get(0)
                  .history.get(i).tr.transition_name.compareTo(t.trans
                  ition_name) == 0) {

Forward_algorithm.forward_moves.get(0).history.get(i).history = 0;
            }
      }


Connection goes = null;
ArrayList<Connection> comes = new ArrayList<Connection>();
for (int i = 0; i < Forward_algorithm.forward_moves.
                  get(0).arcs.size(); i++) {

if ((Forward_algorithm.forward_moves.get(0).arcs.get(i).transition.
            transition_name.compareTo(t.transition_name) == 0)
      && (Forward_algorithm.forward_moves.get(0).arcs.get(i).from ==
            't')) {

      goes = Forward_algorithm.forward_moves.get(0).arcs.get(i);

} else if ((Forward_algorithm.forward_moves.get(0).arcs.get(i).
            transition.transition_name.compareTo(t.transition_name) ==
            0)&&(Forward_algorithm.forward_moves.get(0).arcs.get(i).fr
            om == 'p')) {
```

```
      comes.add(Forward_algorithm.forward_moves.get(0).arcs.get(i));
      }
}
ArrayList<String[]> remove1 = new ArrayList<String[]>();
ArrayList<String[]> add1 = new ArrayList<String[]>();

// the effect of the transition is a token

if (goes.tokens.size() != 0 && goes.bonds.size() == 0) {

      for (int j = 0; j < goes.tokens.size(); j++) {
      for (int i = 0; i < goes.place.tokens.size(); i++) {
      if (goes.place.tokens.get(i).name.
      compareTo(goes.tokens.get(j).name) == 0) {

      for (int k = 0; k < comes.size(); k++) {
      for (int g = 0; g < comes.get(k).tokens.size(); g++) {

      if (comes.get(k).tokens.get(g).name.
      compareTo(goes.tokens.get(j).name) == 0) {


      comes.get(k).place.tokens.add(goes.place.tokens.get(i));
      String s[] = new String[3];
      s[0] = comes.get(k).place.place_name;
      s[1] = goes.place.tokens.get(i).name;
      add1.add(s);
                  }
            }
      }

      goes.place.tokens.remove(goes.place.tokens.get(i));
      String s8[] = new String[3];
      s8[0] = goes.place.place_name;
      s8[1] = goes.place.tokens.get(i).name;
      remove1.add(s8);
                        }
                  }

            }
      }

      // the effect of the transition is a bond
      else if (goes.bonds.size() != 0 && goes.tokens.size() == 0) {


      ArrayList<Place> clear = new ArrayList<Place>();
      for (int i = 0; i < Forward_algorithm.forward_moves.
            get(0).places.size(); i++) {
      for (int j = 0; j < Forward_algorithm.forward_moves
            .get(0).places.get(i).bonds.size(); j++) {
            for (int k = 0; k < goes.bonds.size(); k++) {
            if (Forward_algorithm.forward_moves.get(0).places.get
                  bonds.get(j).name.compareTo(goes.bonds.get(k).name)
                  == 0) {

            Token break1 = new Token();
            Token break2 = new Token();
            String breaked[] = Forward_algorithm.forward_moves.get(0).
```

```
            places.get(i).bonds.get(j).name.split("-");
            break1.name = breaked[0];
            break2.name = breaked[1];
            boolean contain1 = false;
            boolean contain2 = false;

String[]s=new String[3];

s[0]=Forward_algorithm.forward_moves.get(0).places.get(i).place_name;

s[1]=Forward_algorithm.forward_moves.get(0).places.get(i).bonds.get(j)
.name;
remove1.add(s);

Forward_algorithm.forward_moves.get(0).places.get(i).bonds
      .remove(Forward_algorithm.forward_moves.get(0)
      .places.get(i).bonds.get(j));


      for (int h = 0; h < Forward_algorithm.forward_moves.get(0).

      places.get(i).bonds.size(); h++) {

      if (Forward_algorithm.forward_moves.get(0).places.get(i)
            .bonds.get(h).name.contains(break1.name)) {

                  contain1 = true;
      } else if (Forward_algorithm.forward_moves.get(0).places.

      get(i).bonds.get(h).name.contains(break2.name)) {
                  contain2 = true;
                  }
      }

      if (contain1 == false && contain2 == false) {

            String[] s1 = new String[2];
            s1[0] = Forward_algorithm.forward_moves.get(0).
                  places.get(i).place_name;
            s1[1] = break1.name;

            String s9[]=new String[2];
            s9[0] = Forward_algorithm.forward_moves.get(0).
                  places.get(i).place_name;
            s9[1] = break2.name;



      Forward_algorithm.forward_moves.get(0).places.get(i).tokens.add(
      break1);

      Forward_algorithm.forward_moves.get(0).places.get(i).tokens.add(
      break2);


      } else if (contain1 == false && contain2 == true) {

            String s11[] = new String[2];
            s11[0] = Forward_algorithm.forward_moves.get(0)
```

```
                        .places.get(i).place_name;
                s11[1] = break1.name;

                Forward_algorithm.forward_moves.get(0).
                places.get(i).tokens.add(break1);


        } else if (contain1 == true && contain2 == false) {

                String s2[] = new String[2];
                s2[0] = Forward_algorithm.forward_moves.get(0)
                        .places.get(i).place_name;
                s2[1] = break2.name;

                Forward_algorithm.forward_moves.get(0).
                places.get(i).tokens.add(break2);


        }

ArrayList<Connection> input_arcs = new ArrayList<Connection>();

for (int g = 0; g < Forward_algorithm.forward_moves.get(0)
                        .arcs.size(); g++) {
        if ((Forward_algorithm.forward_moves.get(0).arcs.get(g).place
        .place_name.compareTo(Forward_algorithm.forward_moves.get(0).
        places.get(i).place_name) ==0)&&Forward_algorithm.forward_moves
        get(0).arcs.get(g).to == 'p') {

        input_arcs.add(Forward_algorithm.forward_moves.get(0)
        .arcs.get(g));
        }
}

boolean exists_b = false;
for (int h = 0; h < Forward_algorithm.forward_moves.get(0).places
                        .get(i).bonds.size(); h++) {

        exists_b = false;
        Connection save = null;
        for (int y = 0; y < input_arcs.size(); y++) {
                for (int u = 0; u < input_arcs.get(y).bonds.size(); u++) {
                        if (Forward_algorithm.forward_moves.get(0)
                        .places.get(i).bonds.get(h).name.compareTo(inp
                        ut_arcs.get(y).bonds.get(u).name) == 0) {

                        exists_b = true;

                        }
                }
        }

if (exists_b == false) {
        for (int r = Forward_algorithm.forward_moves.get(0)
                .arcs.size() - 1; r >= 0; r--) {

                int history = 0;
                boolean same = false;
                for (int p = 0; p < Forward_algorithm.forward_moves.get(0)
                        .arcs.get(r).bonds.size(); p++) {
```

```
            if (Forward_algorithm.forward_moves.get(0).arcs.get(r)
            .bonds.get(p).name.compareTo(Forward_algorithm.
            forward_moves.get(0).places.get(i).bonds.get(h).name) ==
            0) {

      if (Forward_algorithm.forward_moves.get(0).arcs
            .get(r).transition.transition_name
            .compareTo(t.transition_name) == 0) {

            same = true;

      }

for (int w = 0; w < Forward_algorithm.forward_moves.get(0).history
      .size(); w++) {

      if (Forward_algorithm.forward_moves.get(0).history
            .get(w).tr.transition_name.compareTo(

            Forward_algorithm.forward_moves.get(0).arcs
            .get(r).transition.transition_name) == 0) {

            history = Forward_algorithm.forward_moves.get(0).history
            .get(w).history;

                  }

            }
      }
}
if (history != 0 && same == false) {
      save = Forward_algorithm.forward_moves.get(0).arcs.get(r);

      }
}

if (save != null) {
      boolean ff = true;
for (int rr = 0; rr < Forward_algorithm.forward_moves.get(0).places
      .get(i).bonds.size(); rr++) {

      String f[] = Forward_algorithm.forward_moves.get(0)
                  .places.get(i).bonds.get(rr).name.split("-");

      if ((Forward_algorithm.forward_moves.get(0).places.get(i)
            .bonds.get(h).name.contains(f[0])
            || Forward_algorithm.forward_moves.get(0).places.get(i)
            .bonds.get(h).name.contains(f[1]))
            && (Forward_algorithm.forward_moves.get(0).places.get(i)
            .bonds.get(rr).name.compareTo(Forward_algorithm
            .forward_moves.get(0).places.get(i).bonds.get(h).name) !=
            0)) {


                  ff = false;
            }
}

if (ff == true) {
```

```
        String[] s3 = new String[2];

        s3[0] = save.place.place_name;

        s3[1] = Forward_algorithm.forward_moves.get(0).places.
              get(i).bonds.get(h).name;

        add1.add(s3);

        save.place.bonds.add(

        Forward_algorithm.forward_moves.get(0).places.get(i)
        .bonds.get(h));


        String s4[]=new String[3];

        s4[0] = Forward_algorithm.forward_moves.get(0)
              .places.get(i).place_name;

        s4[1] = Forward_algorithm.forward_moves.get(0).places.
              get(i).bonds.get(h).name;

        s4[2]=Integer.toString(Forward_algorithm.forward_moves.
              get(0).places.get(i).bonds.get(h).id);

        remove1.add(s4);

        Forward_algorithm.forward_moves.get(0).places.get(i).bonds.
        remove(Forward_algorithm.forward_moves.get(0).places.get(i)

        .bonds.get(h));

                              }
                        }
              }
        }

        boolean exists_t = false;
        for (int h = 0; h < Forward_algorithm.forward_moves.get(0).
              places.get(i).tokens.size(); h++) {

              exists_t = false;
              Connection save = null;
              for (int y = 0; y < input_arcs.size(); y++) {
                    for (int u = 0; u < input_arcs.get(y).tokens.size();
                    u++) {
                    if (Forward_algorithm.forward_moves.get(0).
                    places.get(i).tokens.get(h).name.compareTo(input_arc
                    s.get(y).tokens.get(u).name) == 0) {

                          exists_t = true;
                    }
              }
        }


  if (exists_t == false) {
        for (int r = Forward_algorithm.forward_moves.get(0).arcs.size()
```

```
        - 1; r >= 0; r--) {
        for (int p = 0; p < Forward_algorithm.forward_moves.get(0)
              .arcs.get(r).tokens.size(); p++) {
        if (Forward_algorithm.forward_moves.get(0).arcs.get(r)
        .tokens.get(p).name.compareTo(Forward_algorithm.forward_mo
        ves.get(0).places.get(i).tokens.get(h).name) == 0) {

              int history = 0;

              boolean same = false;

        for (int w = 0; w < Forward_algorithm.forward_moves.
              get(0).history.size(); w++) {

        if (Forward_algorithm.forward_moves.get(0)
              .arcs.get(r).transition.transition_name.compareTo(t.
              transition_name) == 0) {

              same = true;

        }

        if (Forward_algorithm.forward_moves.get(0).history.get(w).
              tr.transition_name.compareTo(Forward_algorithm.forwa
              rd_moves.

        get(0).arcs.get(r).transition.transition_name) == 0) {

        history = Forward_algorithm.forward_moves
              .get(0).history.get(w).history;

        }

}


if (history != 0 && same == false) {

        save = Forward_algorithm.forward_moves.get(0).arcs.get(r);



} else if (same == true && history == 0) {

        Place init = null;

        for (int o = 0; o < initial_marking.places.size(); o++) {

        for (int f = 0; f < initial_marking.places.get(o).tokens
              .size(); f++) {

              if (initial_marking.places.get(o)
                    .tokens.get(f).name.compareTo(Forward_algorith
                    m.forward_moves.get(0).places.get(i).tokens.
                    get(h).name) == 0) {

                    init = initial_marking.places.get(o);
```

```
                        }
                    }

                }

for (int l = 0; l < Forward_algorithm.forward_moves.get(0)
      .places.size(); l++) {

      if (init.place_name.compareTo(Forward_algorithm.
            forward_moves.get(0).places.get(l).place_name) == 0) {



            String[] s5 = new String[2];

            s5[0] = Forward_algorithm.forward_moves.get(0).places
                        .get(l).place_name;

            s5[1] = Forward_algorithm.forward_moves.get(0).places
                  .get(i).tokens.get(h).name;

            add1.add(s5);



      Forward_algorithm.forward_moves.get(0).places.get(l).
      tokens.add(Forward_algorithm.forward_moves.get(0).
      places.get(i).tokens.get(h));



                  }

            }

      }



      clear.add(Forward_algorithm.forward_moves.get(0).places.get(i));

                  }

            }

      }

      if (save != null) {
            String[] s6 = new String[2];
            s6[0] = save.place.place_name;
            s6[1] = Forward_algorithm.forward_moves.get(0)
                  .places.get(i).tokens.get(h).name;

            add1.add(s6);
            boolean omg=false;

      for(int d=0;d<save.place.tokens.size();d++){

      if(save.place.tokens.get(d).name
```

```
            .compareTo(Forward_algorithm.forward_moves.get(0)
            .places.get(i).tokens.get(h).name)==0){

      omg=true;
            }
      }

      if(omg==false){

      save.place.tokens.add(Forward_algorithm.forward_moves.get(0)
      .places.get(i).tokens.get(h));
      }

      String[] s7 = new String[3];
      s7[0] = Forward_algorithm.forward_moves.get(0).
            places.get(i).place_name;
      s7[1] = Forward_algorithm.forward_moves.get(0).places.get(i).
            tokens.get(h).name;

      s7[2]=Integer.toString( Forward_algorithm.forward_moves.get(0)
            .places.get(i).tokens.get(h).id);

      remove1.add(s7);


                                          }

                                    }

                              }
                        }
                  }

            }

      }

for (int j = 0; j < Forward_algorithm.forward_moves.get(0).
      places.size(); j++) {

      for (int i = 0; i < clear.size(); i++) {

            if (Forward_algorithm.forward_moves.get(0).places.get(j)
            .place_name.compareTo(clear.get(i).place_name) == 0) {

                  for (int k = 0; k < Forward_algorithm.forward_moves.
                        get(0).places.get(j).tokens.size(); k++) {

                        String[] s66 = new String[3];
                        s66[0] = Forward_algorithm.forward_moves.
                              get(0).places.get(j).place_name;
                        s66[1] = Forward_algorithm.forward_moves.
                              get(0).places.get(j).tokens.get(k).name;

      s66[2]=Integer.toString(Forward_algorithm.forward_moves.get(0)
            .places.get(j).tokens.get(k).id);

      remove1.add(s66);

      Forward_algorithm.forward_moves.get(0).places.get(j).tokens
```

```
        .remove(Forward_algorithm.forward_moves.get(0).places.get(j)
        .tokens.get(k));


                    }
                }

        }
}


if (MainWindow.choice == 3) {
try {
ReverseParser.update_xml(Forward_algorithm.forward_moves.get(0), add1,
remove1);
} catch (ParserConfigurationException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (SAXException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (TransformerException e) {
// TODO Auto-generated catch block
e.printStackTrace();

                        }
                    }

                }

        }
}
```

# Appendix C

## Simulator interface functions

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import java.awt.Color;
import java.awt.Desktop;
import java.awt.Panel;
import java.awt.BorderLayout;
import javax.swing.JTextField;
import java.awt.Font;
import java.awt.Image;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import java.awt.SystemColor;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.SwingConstants;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;
import java.util.regex.Pattern;
import java.awt.event.ActionEvent;
import javax.swing.JPanel;
import java.awt.Graphics;

public class MainWindow {

	private JFrame frame;
	public static PetriNet petri = new PetriNet();
	private Screen2 secondS;
	public static int choice;
	public static PetriNet initial_marking = new PetriNet();
	public static JPanel panel = new JPanel();
	public JLabel lblNewLabel;

	/**
 * Launch the application.
 */
public static void main(String[] args) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				MainWindow window = new MainWindow();
				window.frame.setVisible(true);
```

```
                    } catch (Exception e) {
                          e.printStackTrace();
                    }
              }
        });
}

/**
 * Create the application.
 */
public MainWindow() {
      initialize();
}

/**
 * Initialize the contents of the frame.
 *
 * @return
 */

private void read_file(Scanner scan) {
      int count = 0;
      while (scan.hasNext()) {
            String line = scan.nextLine();
            if (line.compareTo("Tokens:") == 0) {
String f = scan.nextLine();
String[] t = f.split("\\[");
String[] comma = t[1].split(",");
String[] fs = comma[comma.length - 1].split("\\]");
      for (int i = 0; i < comma.length - 1; i++) {
            Token tok = new Token();
            tok.name = comma[i];
            petri.tokens.add(tok);
            Token newt = new Token();
            newt.name = comma[i];
            initial_marking.tokens.add(newt);

      }
      Token tok = new Token();
      tok.name = fs[0];
      petri.tokens.add(tok);
      Token newt = new Token();
      newt.name = fs[0];
      initial_marking.tokens.add(newt);



}
if (line.compareTo("Places:") == 0) {
count = 1;
String f = scan.nextLine();
String[] t = f.split("\\[");
String[] comma = t[1].split(",");
String[] fs = comma[comma.length - 1].split("\\]");
      for (int i = 0; i < comma.length - 1; i++) {
            Place place = new Place();
            place.place_name = comma[i];
            place.place_id = count;
            petri.places.add(place);
```

```
            Place newp = new Place();
            newp.place_name = comma[i];
            newp.place_id = count;
            initial_marking.places.add(newp);
            count++;

      }
      Place tok = new Place();
      tok.place_name = fs[0];
      tok.place_id = count;
      petri.places.add(tok);
      Place newp = new Place();
      newp.place_name = fs[0];
      newp.place_id = count;
      initial_marking.places.add(newp);
      count++;


}
if (line.compareTo("Transitions:") == 0) {
count = 1;
String f = scan.nextLine();
String[] t = f.split("\\[");
String[] comma = t[1].split(",");
String[] fs = comma[comma.length - 1].split("\\]");
      for (int i = 0; i < comma.length - 1; i++) {
            Transition tr = new Transition();
            tr.transition_name = comma[i];
            tr.transition_id = count;

            petri.transitions.add(tr);
            Transition newt = new Transition();
            newt.transition_name = comma[i];
            newt.transition_id = count;
            initial_marking.transitions.add(newt);

            Cell cell = new Cell();
            cell.tr = tr;
            petri.history.add(cell);

            Cell newc = new Cell();

            newc.tr = newt;
            initial_marking.history.add(newc);
            count++;
      }
      Transition tr = new Transition();
      tr.transition_name = fs[0];
      tr.transition_id = count;
      Cell cell = new Cell();
      cell.tr = tr;
      petri.transitions.add(tr);
      petri.history.add(cell);

      Transition newt = new Transition();
      newt.transition_name = fs[0];
      newt.transition_id = count;
      initial_marking.transitions.add(newt);
      Cell newc = new Cell();
```

```
      newc.tr = newt;
      initial_marking.history.add(newc);
      count++;



}

if (line.compareTo("Arcs:") == 0) {
count = 1;
String f = scan.nextLine();
while ((f.compareTo("]") != 0)) {
Connection con = new Connection();
Connection newc = new Connection();
f = scan.nextLine();
if (f.compareTo("]") == 0) {
      break;
}
String[] t = f.split("=");
String left = t[0];
String right = t[1];
String[] l = left.split(",");
String[] tl = l[0].split("\\(");
String l1[] = l[1].split("\\)");

String[] r1 = right.split("\\{");
String[] rn = r1[1].split("\\}");
String[] r = null;
if (rn[0].contains(",")) {
r = rn[0].split(",");
for (int i = 0; i < r.length; i++) {

if (r[i].contains("-") && r[i].contains("not")) {
Token neg_bond = new Token();
Token negb = new Token();
String bb[] = r[i].split("not");
String v[] = bb[1].split(" ");
      neg_bond.name = v[1];
      negb.name = v[1];
      con.negative_bonds.add(neg_bond);
      newc.negative_bonds.add(negb);
      break;

} else if (!r[i].contains("not") && r[i].contains("-")) {
      Token bond = new Token();
      Token nbond = new Token();
      bond.name = r[i];
      nbond.name = r[i];
      con.bonds.add(bond);
      newc.bonds.add(nbond);
}

else if (!r[i].contains("-")) {
if (!r[i].contains("not")) {

      for (int j = 0; j < petri.tokens.size(); j++) {
            if (r[i].compareTo(petri.tokens.get(j).name) == 0) {
                  con.tokens.add(petri.tokens.get(j));
                  Token nt = new Token();
                  nt.name = petri.tokens.get(j).name;
```

```
                        nt.id = petri.tokens.get(j).id;
                        newc.tokens.add(nt);
                        break;
                  }
            }
      } else if (r[i].contains("not")) {
      String bb[] = r[i].split("not");
      String v[] = bb[1].split(" ");
      for (int j = 0; j < petri.tokens.size(); j++) {
      if (v[1].compareTo(petri.tokens.get(j).name) == 0) {
      con.negative_tokens.add(petri.tokens.get(j));
      Token nt = new Token();
      nt.name = petri.tokens.get(j).name;
      nt.id = petri.tokens.get(j).id;
      newc.negative_tokens.add(nt);
      break;
                              }
                        }

                  }
            }

      }
      } else {
            if (rn[0].compareTo(" ") == 0) {
      //    con.empty_token = true;
      } else {
            if (rn[0].length() == 1 || rn[0].length() == 3) {
                  for (int j = 0; j < petri.tokens.size(); j++) {
                        if (rn[0].compareTo(petri.tokens.get(j).name) == 0)
      {
                              con.tokens.add(petri.tokens.get(j));
                              Token nt = new Token();
                              nt.name = petri.tokens.get(j).name;
                              nt.id = petri.tokens.get(j).id;
                              newc.tokens.add(nt);
                              break;
                        } else if (rn[0].contains("-")) {
                                    Token bond = new Token();
                                    Token nb = new Token();
                                    bond.name = rn[0];
                                    nb.name = rn[0];
                                    con.bonds.add(bond);
                                    newc.bonds.add(nb);
                                    break;
                              }

                        }
                  }
            }
      }
      if (tl[1].startsWith("p") == true) {
      for (int i = 0; i < petri.places.size(); i++) {
      if (petri.places.get(i).place_name.compareTo(tl[1]) == 0) {
      con.place = petri.places.get(i);
      con.from = 'p';
      con.to = 't';
      newc.from = 'p';
      newc.to = 't';
      break;
```

```
      }
}
for (int i = 0; i < petri.transitions.size(); i++) {
      if (petri.transitions.get(i).transition_name.compareTo(l1[0]) ==
0) {
            con.transition = petri.transitions.get(i);
            con.transition.num_of_input = con.transition.num_of_input
+ 1;
            break;
      }
}

} else if (tl[1].startsWith("t") == true) {
for (int i = 0; i < petri.transitions.size(); i++) {
if (petri.transitions.get(i).transition_name.compareTo(tl[1]) == 0) {
con.transition = petri.transitions.get(i);
con.from = 't';
con.to = 'p';
newc.from = 't';
newc.to = 'p';
                  }
            }
for (int i = 0; i < petri.places.size(); i++) {
if (petri.places.get(i).place_name.compareTo(l1[0]) == 0) {
      con.place = petri.places.get(i);
      con.transition.num_of_output = con.transition.num_of_output + 1;
      break;
                  }
            }
      }

con.connection_id = count;
newc.connection_id = count;
count++;
petri.arcs.add(con);
initial_marking.arcs.add(newc);

      }
}



if (line.compareTo("Initial marking:") == 0) {
String f = scan.nextLine();
while ((f.compareTo("]") != 0)) {
f = scan.nextLine();
String[] p = f.split("->");
if (f.compareTo("]") == 0) {
      break;
}

for (int i = 0; i < petri.places.size(); i++) {
      if (petri.places.get(i).place_name.compareTo(p[0]) == 0) {
            if (p[1].compareTo("0") == 0) {
      petri.places.get(i).empty = true;

} else {
      if (p[1].contains("-")) {
                  Token bond = new Token();
                  Token nb = new Token();
```

```
                    bond.name = p[1];
                    nb.name = p[1];
                    petri.places.get(i).bonds.add(bond);
                    initial_marking.places.get(i).bonds.add(nb);

                } else {
                for (int j = 0; j < petri.tokens.size(); j++) {
                if (petri.tokens.get(j).name.compareTo(p[1]) == 0) {
                    Token nb = new Token();
                    nb.name = p[1];
                    petri.places.get(i).tokens.add(petri.tokens.get(j));
                    initial_marking.places.get(i).tokens.add(nb);
                    break;
                                                        }
                                                    }

                                                }
                                            }
                                        }

                                    }
                                }
                            }

        }

}

private void initialize() {

frame = new JFrame();
frame.setResizable(false);
frame.setTitle("Simulator for Reversible Petri Nets");
frame.setBounds(100, 100, 633, 440);
panel.setBackground(Color.WHITE);
panel.setLayout(null);
panel.setBounds(100, 100, 633, 440);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
lblNewLabel = new JLabel("Please select the method you want to give
the input:");
lblNewLabel.setBounds(86, 11, 478, 20);
lblNewLabel.setHorizontalAlignment(SwingConstants.CENTER);
lblNewLabel.setForeground(Color.BLACK);
lblNewLabel.setFont(new Font("Times New Roman", Font.BOLD, 17));
lblNewLabel.setVisible(true);
panel.add(lblNewLabel);
JButton btnuser = new JButton("CREATE NEW PETRI NET",
new ImageIcon(MainWindow.class.getResource("/images/user.png")));
btnuser.setBounds(217, 85, 184, 271);
panel.add(btnuser);
btnuser.setHorizontalTextPosition(SwingConstants.CENTER);
btnuser.setOpaque(false);
btnuser.setContentAreaFilled(false);
btnuser.setFont(new Font("Times New Roman", Font.PLAIN, 12));
btnuser.setVerticalAlignment(SwingConstants.TOP);
btnuser.setVerticalTextPosition(SwingConstants.TOP);

JButton btnread = new JButton("READ FROM FILE",
new ImageIcon(MainWindow.class.getResource("/images/file.png")));
btnread.setBounds(10, 85, 184, 271);
```

```
panel.add(btnread);
btnread.setVerticalTextPosition(SwingConstants.TOP);
btnread.setHorizontalTextPosition(SwingConstants.CENTER);
btnread.setOpaque(false);
btnread.setContentAreaFilled(false);
btnread.setFont(new Font("Times New Roman", Font.PLAIN, 12));
btnread.setVerticalAlignment(SwingConstants.TOP);
btnread.setText("READ FROM FILE\r\n");
btnread.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		choice = 1;
		JFileChooser fc = new JFileChooser();
		File file;
		String filename = "";
int returnval = fc.showOpenDialog(fc);
if (returnval == JFileChooser.APPROVE_OPTION) {
fc.setCurrentDirectory(new File(System.getProperty("user.home")));
file = fc.getSelectedFile();
filename = file.getAbsolutePath();
Scanner scan=null;

try {
	scan = new Scanner(file);
} catch (FileNotFoundException e1) {
	// TODO Auto-generated catch block
	e1.printStackTrace();
}
String path = file.getAbsolutePath();
if (!path.endsWith(".txt")) {
JOptionPane.showMessageDialog
(panel, "The file type is incorrect. It has to be a .txt file.",
null, JOptionPane.INFORMATION_MESSAGE);
} else {
	read_file(scan);
	panel.setVisible(false);
	try {
		secondS = new Screen2();
	} catch (IOException e) {
		// TODO Auto-generated catch block
	e.printStackTrace();
} catch (ParserConfigurationException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
} catch (SAXException e) {
	// TODO Auto-generated catch block
e.printStackTrace();
}
secondS.setVisible(true);
frame.getContentPane().add(secondS);
		}
	} else if (returnval == JFileChooser.ERROR_OPTION) {
		frame.setBounds(100, 100, 633, 440);
		MainWindow.panel.setVisible(true);
	}

	}
});

btnuser.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
```

```
            choice = 2;
            frame.setBounds(100, 100, 761, 680);
            readUser2 rf=new readUser2(frame);
            panel.setVisible(false);
            rf.setVisible(true);
      }
});

JButton btnDrawTheInitial = new JButton("DRAW INITIAL PETRINET");
btnDrawTheInitial.setVerticalAlignment(SwingConstants.TOP);
btnDrawTheInitial.setBackground(Color.WHITE);
btnDrawTheInitial.setToolTipText("");
btnDrawTheInitial.setIcon(new
ImageIcon(MainWindow.class.getResource("/images/images.png")));
btnDrawTheInitial.setFont(new Font("Times New Roman", Font.PLAIN,
12));
btnDrawTheInitial.setBounds(423, 85, 194, 271);
btnDrawTheInitial.setVerticalTextPosition(SwingConstants.TOP);
btnDrawTheInitial.setHorizontalTextPosition(SwingConstants.CENTER);
btnDrawTheInitial.addActionListener(new ActionListener() {

      public void actionPerformed(ActionEvent arg0) {
            choice = 3;
            panel.setVisible(false);
            Explanation exp = new Explanation(frame);
            frame.getContentPane().add(exp);
            exp.setVisible(true);
            }
      });
      panel.add(btnDrawTheInitial);

      frame.getContentPane().add(panel);

      }
}
```

```
import java.awt.Color;
import java.awt.Font;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

import javax.swing.SwingConstants;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyAdapter;
```

```
import java.awt.event.KeyEvent;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.JComboBox;
import javax.swing.UIManager;
import java.awt.SystemColor;
import javax.swing.DefaultComboBoxModel;
import javax.swing.DefaultListModel;
import java.awt.Panel;

public class readUser2 extends JPanel {

private static final long serialVersionUID = 1L;
public static Object[] columns = { "Place name", "Token/Bond" };
public static Object[] columns1 = { "Transition name" };
public static Object[] columns2 = { "Token name"};
public static Object[] columns3 = { "Bond name"};
public static Object[] columns4 = { "Place", "Transition", "From",
"To" ,"Label"};

public static Object[] columns5 = { "Bond name"};
public static Object[] columns6 = { "Token name"};
private JTextField textField;
private JTextField textField_1;
private JTextField textField_2;
public static PetriNet initial_marking = new PetriNet();
public static PetriNet petri = new PetriNet();
private JTextField textField_3;
public JTextField textField_4;
public ArrayList<String[]>arc_label=new ArrayList<String[]>();
private Screen2 secondS;


public readUser2(JFrame frame) {
      JPanel panel = new JPanel();
      setBackground(Color.WHITE);
      setLayout(null);
      frame.setBounds(130, 130, 992, 720);
      panel.setBounds(20, 20, 990, 717);
      panel.setBackground(Color.WHITE);
      panel.setLayout(null);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JLabel lblReadInputFrom = new JLabel("Create new Petri net:");
lblReadInputFrom.setForeground(Color.BLACK);
lblReadInputFrom.setHorizontalAlignment(SwingConstants.CENTER);
lblReadInputFrom.setBounds(400, 0, 233, 32);
lblReadInputFrom.setFont(new Font("Times New Roman", Font.BOLD, 23));
panel.add(lblReadInputFrom);

DefaultTableModel model = new DefaultTableModel();
JTable table = new JTable();
JScrollPane scroll = new JScrollPane();
panel.add(scroll);
scroll.setViewportView(table);
model.setColumnIdentifiers(columns);
table.setModel(model);
table.setVisible(true);
scroll.setBounds(753, 77, 194, 78);
```

```
JLabel lblDeclareThePlaces = new JLabel("Declare the places of your
marking :");
lblDeclareThePlaces.setFont(new Font("Times New Roman", Font.BOLD,
13));
lblDeclareThePlaces.setBounds(22, 54, 261, 32);
panel.add(lblDeclareThePlaces);

JLabel lblPlaces = new JLabel("Places");
lblPlaces.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblPlaces.setHorizontalAlignment(SwingConstants.CENTER);
lblPlaces.setBounds(822, 54, 65, 25);
panel.add(lblPlaces);

DefaultTableModel model_transitions = new DefaultTableModel();
JTable table1 = new JTable();
JScrollPane scroll1 = new JScrollPane();
panel.add(scroll1);
scroll1.setViewportView(table1);
model_transitions.setColumnIdentifiers(columns1);
table1.setModel(model_transitions);
table1.setVisible(true);
scroll1.setBounds(824, 193, 122, 78);

DefaultTableModel model_negt = new DefaultTableModel();
JTable tablenegt = new JTable();
JScrollPane scrollnt = new JScrollPane();
panel.add(scrollnt);
scrollnt.setViewportView(tablenegt);
model_negt.setColumnIdentifiers(columns6);
tablenegt.setModel(model_negt);
tablenegt.setVisible(true);
scrollnt.setBounds(847, 315, 97, 78);

DefaultTableModel model_negb = new DefaultTableModel();
JTable tablenegb = new JTable();
JScrollPane scrollnb = new JScrollPane();
panel.add(scrollnb);
scrollnb.setViewportView(tablenegb);
model_negb.setColumnIdentifiers(columns5);
tablenegb.setModel(model_negb);
tablenegb.setVisible(true);
scrollnb.setBounds(847, 446, 97, 78);

JLabel lblTransitions = new JLabel("Transitions");
lblTransitions.setHorizontalAlignment(SwingConstants.CENTER);
lblTransitions.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblTransitions.setBounds(861, 169, 65, 25);
panel.add(lblTransitions);

DefaultTableModel model_tokens = new DefaultTableModel();
JTable table2 = new JTable();
JScrollPane scroll2 = new JScrollPane();
panel.add(scroll2);
scroll2.setViewportView(table2);
model_tokens.setColumnIdentifiers(columns2);
table2.setModel(model_tokens);
table2.setVisible(true);
scroll2.setBounds(703, 315, 102, 78);

JLabel lblTokens = new JLabel("Tokens");
```

```
lblTokens.setHorizontalAlignment(SwingConstants.CENTER);
lblTokens.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblTokens.setBounds(703, 292, 65, 25);
panel.add(lblTokens);

JLabel lblDeclareTheTransitions = new JLabel("Declare the transitions
of your marking :");
lblDeclareTheTransitions.setFont(new Font("Times New Roman",
Font.BOLD, 13));
lblDeclareTheTransitions.setBounds(22, 169, 261, 32);
panel.add(lblDeclareTheTransitions);

JLabel lblDeclareTheTokens = new JLabel("Declare the tokens of your
marking :");
lblDeclareTheTokens.setFont(new Font("Times New Roman", Font.BOLD,
13));
lblDeclareTheTokens.setBounds(22, 292, 261, 32);
panel.add(lblDeclareTheTokens);

JLabel lblPlaceName = new JLabel("Place name:");
lblPlaceName.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblPlaceName.setHorizontalAlignment(SwingConstants.CENTER);
lblPlaceName.setBounds(45, 89, 71, 14);
panel.add(lblPlaceName);

JLabel lblPleasePressEnter = new JLabel
("<html> Please press enter before </b>creating the object .</html>");
lblPleasePressEnter.setFont(new Font("Tahoma", Font.PLAIN, 11));
lblPleasePressEnter.setVerticalAlignment(SwingConstants.TOP);
lblPleasePressEnter.setForeground(Color.RED);
lblPleasePressEnter.setBounds(214, 85, 147, 37);
panel.add(lblPleasePressEnter);

JButton btnCreate = new JButton("CREATE");
btnCreate.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent arg0) {
boolean flag=false;
for(int i=0;i<model.getRowCount();i++){
      if(model.getValueAt(i,
0).toString().compareTo(textField.getText())==0){
            flag=true;
      }
}
if(!flag){
Object[] name = new Object[1];
name[0] = textField.getText();
model.addRow(name);
textField.setText("");
lblPleasePressEnter.setForeground(Color.red);
} else {
JOptionPane.showMessageDialog(panel,
"The places are unique, you have to create a place with a different
name.", null,
JOptionPane.INFORMATION_MESSAGE);
textField.setText("");
lblPleasePressEnter.setForeground(Color.RED);

            }

            }
```

```
});
btnCreate.setBackground(new Color(50, 205, 50));
btnCreate.setFont(new Font("Times New Roman", Font.BOLD, 13));
btnCreate.setForeground(Color.WHITE);
btnCreate.setBounds(121, 122, 89, 23);
panel.add(btnCreate);

textField = new JTextField();

textField.addKeyListener(new KeyAdapter() {
	@Override
	public void keyPressed(java.awt.event.KeyEvent evt) {
		if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
			lblPleasePressEnter.setForeground(Color.GREEN);
		}
	}

});

textField.setBounds(139, 86, 71, 25);
panel.add(textField);
textField.setColumns(10);

frame.getContentPane().add(panel);

JLabel lblTransitionName = new JLabel("Transition name:");
lblTransitionName.setHorizontalAlignment(SwingConstants.CENTER);
lblTransitionName.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblTransitionName.setBounds(44, 217, 94, 14);
panel.add(lblTransitionName);
JLabel label_1 = new JLabel
("<html> Please press enter before </b>creating the object .</html>");
label_1.setVerticalAlignment(SwingConstants.TOP);
label_1.setForeground(Color.RED);
label_1.setFont(new Font("Tahoma", Font.PLAIN, 11));
label_1.setBounds(226, 212, 147, 37);
panel.add(label_1);

textField_1 = new JTextField();
textField_1.addKeyListener(new KeyAdapter() {
	@Override
	public void keyPressed(java.awt.event.KeyEvent evt) {
		if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
			label_1.setForeground(Color.GREEN);
		}
	}

});
textField_1.setColumns(10);
textField_1.setBounds(151, 212, 71, 25);
panel.add(textField_1);

JButton button = new JButton("CREATE");
button.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
boolean flag=false;
for(int i=0;i<model_transitions.getRowCount();i++){
if(model_transitions.getValueAt(i,
0).toString().compareTo(textField_1.getText())==0){
```

```
flag=true;
		}
}
if(!flag){
Object[] name = new Object[1];
name[0] = textField_1.getText();
model_transitions.addRow(name);
textField_1.setText("");
label_1.setForeground(Color.red);

}
else{
		JOptionPane.showMessageDialog(panel,
		"The transitions are unique, you have to create a transition
with a different name."
		, null,JOptionPane.INFORMATION_MESSAGE);
		textField_1.setText("");
		label_1.setForeground(Color.red);
			}
		}
});
button.setForeground(Color.WHITE);
button.setFont(new Font("Times New Roman", Font.BOLD, 13));
button.setBackground(new Color(50, 205, 50));
button.setBounds(121, 248, 89, 23);
panel.add(button);

JLabel lblTokenName = new JLabel("Token name:");
lblTokenName.setHorizontalAlignment(SwingConstants.CENTER);
lblTokenName.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblTokenName.setBounds(44, 329, 94, 14);
panel.add(lblTokenName);
JLabel label_2 = new JLabel
("<html> Please press enter before </b>creating the object .</html>");
label_2.setVerticalAlignment(SwingConstants.TOP);
label_2.setForeground(Color.RED);
label_2.setFont(new Font("Tahoma", Font.PLAIN, 11));
label_2.setBounds(226, 324, 147, 37);
panel.add(label_2);

textField_2 = new JTextField();
textField_2.addKeyListener(new KeyAdapter() {
		@Override
		public void keyPressed(java.awt.event.KeyEvent evt) {
			if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
				label_2.setForeground(Color.GREEN);
			}
		}

});
textField_2.setColumns(10);
textField_2.setBounds(151, 324, 71, 25);
panel.add(textField_2);

JButton button_1 = new JButton("CREATE");
button_1.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
		boolean flag=false;
		for(int i=0;i<model_tokens.getRowCount();i++){
```

```
            if(model_tokens.getValueAt(i,
0).toString().compareTo(textField_2.getText())==0){
                  flag=true;
            }
      }
      if(!flag){
      Object[] name = new Object[1];
      name[0] = textField_2.getText();
      model_tokens.addRow(name);
      textField_2.setText("");
label_2.setForeground(Color.red);
}
else{
JOptionPane.showMessageDialog(panel,
"The tokens are unique, you have to create a token with a different
name."
, null,JOptionPane.INFORMATION_MESSAGE);
textField_2.setText("");
      label_2.setForeground(Color.red);
            }
      }
});
button_1.setForeground(Color.WHITE);
button_1.setFont(new Font("Times New Roman", Font.BOLD, 13));
button_1.setBackground(new Color(50, 205, 50));
button_1.setBounds(121, 360, 89, 23);
panel.add(button_1);

JLabel lblDeclareTheBonds = new JLabel("Declare the bonds of your
marking :");
lblDeclareTheBonds.setFont(new Font("Times New Roman", Font.BOLD,
13));
lblDeclareTheBonds.setBounds(22, 417, 261, 32);
panel.add(lblDeclareTheBonds);

JComboBox<String> comboBox = new JComboBox<String>();

comboBox.setBounds(221, 482, 54, 20);

comboBox.setVisible(false);
panel.add(comboBox);

JLabel lblNewLabel = new JLabel("Token 1:");
lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 11));
lblNewLabel.setBounds(218, 460, 60, 14);
panel.add(lblNewLabel);
lblNewLabel.setVisible(false);

JLabel lblToken = new JLabel("Token 2:");
lblToken.setFont(new Font("Tahoma", Font.BOLD, 11));
      lblToken.setBounds(313, 460, 60, 14);
      panel.add(lblToken);
      lblToken.setVisible(false);

      JComboBox<String> comboBox_1 = new JComboBox<String>();
      comboBox_1.setBounds(310, 482, 59, 20);
      panel.add(comboBox_1);
      comboBox_1.setVisible(false);

      DefaultTableModel model_bonds = new DefaultTableModel();
```

```
		JTable table3 = new JTable();
		JScrollPane scroll3 = new JScrollPane();
		panel.add(scroll3);
		scroll3.setViewportView(table3);
		model_bonds.setColumnIdentifiers(columns3);
		table3.setModel(model_bonds);
		table3.setVisible(true);
		scroll3.setBounds(703, 446, 102, 78);

		DefaultTableModel model_arcs = new DefaultTableModel();
		JTable table4 = new JTable();
		JScrollPane scroll4 = new JScrollPane();
		panel.add(scroll4);
		scroll4.setViewportView(table4);
		model_arcs.setColumnIdentifiers(columns4);
		table4.setModel(model_arcs);
		table4.setVisible(true);
		scroll4.setBounds(657, 571, 323, 78);


		JButton button_2 = new JButton("CREATE");
	button_2.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent arg0) {
			String c1 = comboBox.getSelectedItem().toString();
			String c2 = comboBox_1.getSelectedItem().toString();
			if(c1.compareTo(c2)==0){
				JOptionPane.showMessageDialog(panel,
				"The bond has to be composed from two different
	tokens.Please try again"
				, null,JOptionPane.INFORMATION_MESSAGE);
		comboBox.setSelectedIndex(-1);
		comboBox_1.setSelectedIndex(-1);
	}
	else{
	Object[] c = new Object[1];
	c[0] = c1 + "-" + c2;
			model_bonds.addRow(c);
			comboBox.setSelectedIndex(-1);
			comboBox_1.setSelectedIndex(-1);
			}
		}
	});
	button_2.setForeground(Color.WHITE);
	button_2.setFont(new Font("Times New Roman", Font.BOLD, 13));
	button_2.setBackground(new Color(50, 205, 50));
	button_2.setBounds(250, 513, 89, 23);
	panel.add(button_2);
	button_2.setVisible(false);
	JButton btnCreateBond = new JButton("CREATE BOND");
	btnCreateBond.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent arg0) {
			if (model_tokens.getRowCount() > 1) {
				lblToken.setVisible(true);
				lblNewLabel.setVisible(true);
				comboBox.setVisible(true);
				comboBox_1.setVisible(true);
				button_2.setVisible(true);
				Object f = null;

				for (int i = 0; i < table2.getRowCount(); i++) {
```

```
                        f = model_tokens.getValueAt(i, 0);
                        comboBox.insertItemAt(f.toString(), i);

                    }
                    for (int i = 0; i < table2.getRowCount(); i++) {
                        f = model_tokens.getValueAt(i, 0);
                        comboBox_1.insertItemAt(f.toString(), i);

                    }

                    comboBox.setSelectedIndex(-1);
                    comboBox_1.setSelectedIndex(-1);

                } else {
                    JOptionPane.showMessageDialog(panel,
                    "You have to define the tokens of the marking before
creating the bonds."
                    , null,JOptionPane.INFORMATION_MESSAGE);
                }
            }
});
btnCreateBond.setBackground(SystemColor.inactiveCaption);
btnCreateBond.setForeground(Color.WHITE);
btnCreateBond.setFont(new Font("Tahoma", Font.BOLD, 12));
btnCreateBond.setBounds(230, 421, 131, 23);
panel.add(btnCreateBond);

JLabel lblBonds = new JLabel("Bonds");
lblBonds.setVerticalAlignment(SwingConstants.TOP);
lblBonds.setHorizontalAlignment(SwingConstants.CENTER);
lblBonds.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblBonds.setBounds(703, 421, 65, 25);
panel.add(lblBonds);

JLabel lblDeclareTheArcs = new JLabel("Declare the arcs of your
marking :");
lblDeclareTheArcs.setHorizontalAlignment(SwingConstants.LEFT);
lblDeclareTheArcs.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblDeclareTheArcs.setBounds(22, 548, 208, 32);
panel.add(lblDeclareTheArcs);

JComboBox<String> comboBox_2 = new JComboBox<String>();
comboBox_2.setBounds(22, 608, 54, 20);
panel.add(comboBox_2);
comboBox_2.setVisible(false);

JComboBox<String> comboBox_3 = new JComboBox<String>();
comboBox_3.setBounds(104, 608, 54, 20);
panel.add(comboBox_3);
comboBox_3.setVisible(false);

JLabel lblNewLabel_1 = new JLabel("Place:");
lblNewLabel_1.setFont(new Font("Tahoma", Font.BOLD, 11));
lblNewLabel_1.setBounds(22, 586, 46, 14);
panel.add(lblNewLabel_1);
lblNewLabel_1.setVisible(false);

JLabel lblNewLabel_2 = new JLabel("Transition:");
lblNewLabel_2.setFont(new Font("Tahoma", Font.BOLD, 11));
lblNewLabel_2.setBounds(104, 586, 71, 14);
```

```
panel.add(lblNewLabel_2);
lblNewLabel_2.setVisible(false);

JComboBox<String> comboBox_4 = new JComboBox<String>();
comboBox_4.setModel(new DefaultComboBoxModel(new String[] { "Place",
"Transition" }));
comboBox_4.setBounds(22, 654, 87, 20);
panel.add(comboBox_4);
comboBox_4.setVisible(false);


JButton button_3 = new JButton("CREATE");
button_3.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
            String c1 = comboBox_2.getSelectedItem().toString();
            String c2 = comboBox_3.getSelectedItem().toString();
            String c3 = comboBox_4.getSelectedItem().toString();
            String c4=textField_4.getText();
            Object[] c = new Object[5];
            c[0] = c1;
            c[1] = c2;
            String from = c3.substring(0, 1);
            from = from.toLowerCase();
            c[2] = from;
            if (from.compareTo("p") == 0) {
c[3] = "t";
} else {
      c[3] = "p";
}
if(c4.contains(",")){
String split[]=c4.split(",");
String s[]=new String[split.length];
for(int i=0;i<split.length;i++){
      s[i]=split[i];
}
arc_label.add(s);
boolean check[]=new boolean[split.length];
for(int i=0;i<split.length;i++){
      for(int j=0;j<table2.getRowCount();j++){
            if(split[i].compareTo(table2.getValueAt(j,
0).toString())==0){
                  check[i]=true;
            }
      }
      for(int j=0;j<table3.getRowCount();j++){
            if(split[i].compareTo(table3.getValueAt(j,
0).toString())==0){
                  check[i]=true;
            }
      }
      for(int j=0;j<tablenegt.getRowCount();j++){

      if(split[i].compareTo(tablenegt.getValueAt(j,0).toString())==0){
                  check[i]=true;
            }
      }
      for(int j=0;j<tablenegb.getRowCount();j++){
            if(split[i].compareTo(tablenegb.getValueAt(j,
0).toString())==0){
                  check[i]=true;
```

```
                }
        }
}
boolean cf=true;
for(int i=0;i<check.length;i++){
        if(check[i]!=true){
                cf=false;
        }
}
if(cf==true){
        c[4]=c4;
        model_arcs.addRow(c);
}
else{
        JOptionPane.showMessageDialog(panel,
        "You have to define the tokens/bonds before assign them to an
arc as label.",
        null, JOptionPane.INFORMATION_MESSAGE);
}
}
else{
        String s[]=new String[1];
        s[0]=c4;
        arc_label.add(s);
        boolean check=false;
        for(int i=0;i<table2.getRowCount();i++){
                if(c4.compareTo(table2.getValueAt(i, 0).toString())==0){
                        check=true;
                }
        }
        for(int i=0;i<table3.getRowCount();i++){
                if(c4.compareTo(table3.getValueAt(i, 0).toString())==0){
                        check=true;
                }
        }
        for(int i=0;i<tablenegt.getRowCount();i++){
                if(c4.compareTo(tablenegt.getValueAt(i,
0).toString())==0){
                        check=true;
                }
        }
        for(int i=0;i<tablenegb.getRowCount();i++){
                if(c4.compareTo(tablenegb.getValueAt(i,
0).toString())==0){
                        check=true;
                }
        }
        if(check==true){
        c[4]=c4;
        model_arcs.addRow(c);
        }
}

comboBox_2.setSelectedIndex(-1);
comboBox_3.setSelectedIndex(-1);
comboBox_4.setSelectedIndex(-1);
textField_4.setText("");

        }
});
```

```
button_3.setForeground(Color.WHITE);
button_3.setFont(new Font("Times New Roman", Font.BOLD, 13));
button_3.setBackground(new Color(50, 205, 50));
button_3.setBounds(155, 657, 89, 23);
panel.add(button_3);
button_3.setVisible(false);

JLabel lblFrom = new JLabel("From:");
lblFrom.setFont(new Font("Tahoma", Font.BOLD, 11));
lblFrom.setBounds(22, 635, 46, 14);
panel.add(lblFrom);
lblFrom.setVisible(false);
JLabel lblArcLabel = new JLabel("Arc label:");
lblArcLabel.setFont(new Font("Tahoma", Font.BOLD, 11));
lblArcLabel.setBounds(176, 586, 54, 14);
panel.add(lblArcLabel);
lblArcLabel.setVisible(false);

textField_4 = new JTextField();
textField_4.setBounds(168, 608, 157, 20);
panel.add(textField_4);
textField_4.setColumns(10);
textField_4.setVisible(false);

JLabel lbluseCommas = new JLabel
("<html>Use commas (\",\") for each different</b> token/bond</b> and
\"not\" "
+ "for each negative token</b> and/or bond.</html>");
lbluseCommas.setVerticalAlignment(SwingConstants.TOP);
lbluseCommas.setFont(new Font("Tahoma", Font.PLAIN, 10));
lbluseCommas.setBounds(168, 626, 250, 37);
panel.add(lbluseCommas);
lbluseCommas.setVisible(false);

JButton btnCreateArc = new JButton("CREATE ARC");
btnCreateArc.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
            if (model.getRowCount() >= 1 &&
model_transitions.getRowCount() >= 1
                  && model_tokens.getRowCount()>=1) {
                  lblNewLabel_2.setVisible(true);
                  lblNewLabel_1.setVisible(true);
                  comboBox_3.setVisible(true);
                  comboBox_2.setVisible(true);
                  button_3.setVisible(true);
                  comboBox_4.setVisible(true);
                  lblArcLabel.setVisible(true);
                  textField_4.setVisible(true);
                  lbluseCommas.setVisible(true);
                  lblFrom.setVisible(true);
                  Object f = null;
                  for (int i = 0; i < table.getRowCount(); i++) {
                        f = model.getValueAt(i, 0);
                        comboBox_2.insertItemAt(f.toString(), i);
                  }
                  for (int i = 0; i < table1.getRowCount(); i++) {
                        f = model_transitions.getValueAt(i, 0);
                        comboBox_3.insertItemAt(f.toString(), i);
                  }
```

```
                    comboBox_2.setSelectedIndex(0);
                    comboBox_3.setSelectedIndex(0);

              } else {
                    JOptionPane.showMessageDialog(panel,
                    "You have to define the places ,tokens,and
transtions of the"
                    + " marking before creating the arcs.",
                    null, JOptionPane.INFORMATION_MESSAGE);
              }
        }
});
btnCreateArc.setForeground(Color.WHITE);
btnCreateArc.setFont(new Font("Tahoma", Font.BOLD, 12));
btnCreateArc.setBackground(SystemColor.inactiveCaption);
btnCreateArc.setBounds(230, 552, 131, 23);
panel.add(btnCreateArc);

JLabel lblArcs = new JLabel("Arcs");
lblArcs.setVerticalAlignment(SwingConstants.TOP);
lblArcs.setHorizontalAlignment(SwingConstants.CENTER);
lblArcs.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblArcs.setBounds(802, 552, 65, 25);
panel.add(lblArcs);

JLabel lblInsertTokenbondIn = new JLabel("Insert token/bond in a
place:");
lblInsertTokenbondIn.setHorizontalAlignment(SwingConstants.LEFT);
lblInsertTokenbondIn.setFont(new Font("Times New Roman", Font.BOLD,
13));
lblInsertTokenbondIn.setBounds(428, 54, 163, 32);
panel.add(lblInsertTokenbondIn);
JLabel lblPlace = new JLabel("Place:");
lblPlace.setFont(new Font("Tahoma", Font.BOLD, 11));
lblPlace.setBounds(428, 110, 46, 14);
panel.add(lblPlace);
lblPlace.setVisible(false);

JComboBox<String> comboBox_6 = new JComboBox<String>();
comboBox_6.setBounds(428, 132, 54, 25);
panel.add(comboBox_6);
comboBox_6.setVisible(false);

JLabel lblTokenbond = new JLabel("Token/Bond:");
lblTokenbond.setFont(new Font("Tahoma", Font.BOLD, 11));
lblTokenbond.setBounds(516, 110, 79, 14);
panel.add(lblTokenbond);
lblTokenbond.setVisible(false);

JComboBox <String>comboBox_7 = new JComboBox<String>();
comboBox_7.setBounds(526, 132, 54, 24);
panel.add(comboBox_7);
comboBox_7.setVisible(false);
JButton button_4 = new JButton("CREATE");
button_4.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
            String c1 = comboBox_6.getSelectedItem().toString();
            String c2 = comboBox_7.getSelectedItem().toString();
            Object[] c = new Object[1];
```

```
            int d=Integer.parseInt(c1.substring(1, 2));
            c[0] = c2;
            String before="";
if(model.getValueAt(d-1, 1)!=null){
 before=model.getValueAt(d-1, 1).toString()+","+c2;
             model.setValueAt(before,d-1, 1);
            }

            else{
                  model.setValueAt(c2,d-1, 1);

            }
            comboBox_6.setSelectedIndex(-1);
            comboBox_7.setSelectedIndex(-1);

      }
});
button_4.setForeground(Color.WHITE);
button_4.setFont(new Font("Times New Roman", Font.BOLD, 13));
button_4.setBackground(new Color(50, 205, 50));
button_4.setBounds(605, 118, 89, 23);
panel.add(button_4);
button_4.setVisible(false);

JButton btnInsertTokenbondTo = new JButton("INSERT TOKEN TO PLACE");
btnInsertTokenbondTo.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
            if(model.getRowCount()>=1 &&
model_tokens.getRowCount()>=1){
                  comboBox_7.setVisible(true);
                  comboBox_6.setVisible(true);
                  lblTokenbond.setVisible(true);
                  lblPlace.setVisible(true);
                  button_4.setVisible(true);
                  Object f = null;
                  for(int i=0;i<model.getRowCount();i++){
                              f = model.getValueAt(i, 0);
                              comboBox_6.insertItemAt(f.toString(),
i);
                        }
                  for (int i = 0; i < table2.getRowCount(); i++) {
                        f = model_tokens.getValueAt(i, 0);
                        comboBox_7.insertItemAt(f.toString(), i);
                  }
                  for (int i = 0; i < table3.getRowCount(); i++) {
                        f = model_bonds.getValueAt(i, 0);
                        comboBox_7.insertItemAt(f.toString(), i);
                  }
                  comboBox_6.setSelectedIndex(0);
                  comboBox_7.setSelectedIndex(0);
            }

else {
      JOptionPane.showMessageDialog(panel,
      "You have to define the places and tokens of the marking before
define tokens to a place.",
      null, JOptionPane.INFORMATION_MESSAGE);
}
}
});
```

```
btnInsertTokenbondTo.setForeground(Color.WHITE);
btnInsertTokenbondTo.setFont(new Font("Tahoma", Font.BOLD, 12));
btnInsertTokenbondTo.setBackground(SystemColor.inactiveCaption);
btnInsertTokenbondTo.setBounds(417, 85, 186, 22);
panel.add(btnInsertTokenbondTo);



JButton btnCreatePetriNet = new JButton("CREATE PETRI NET");
btnCreatePetriNet.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		ArrayList<String[]>places=new ArrayList<String[]>();
		ArrayList<String>transitions=new ArrayList<String>();
		ArrayList<String>tokens=new ArrayList<String>();
		ArrayList<String>bonds=new ArrayList<String>();
		ArrayList<String[]>arcs=new ArrayList<String[]>();
		ArrayList<String>neg_bonds=new ArrayList<String>();
		ArrayList<String>neg_tokens=new ArrayList<String>();


		for(int i=0;i<model.getRowCount();i++){
			String contents[]=new String[2];
			contents[0]=model.getValueAt(i, 0).toString();
			if(model.getValueAt(i, 1)!=null){
			contents[1]=model.getValueAt(i, 1).toString();
			}
			else{
				contents[1]="";
	}
	places.add(contents);
}
for(int i=0;i<model_transitions.getRowCount();i++){
	transitions.add(model_transitions.getValueAt(i, 0).toString());
}
for(int i=0;i<model_tokens.getRowCount();i++){
	tokens.add(model_tokens.getValueAt(i, 0).toString());
}
for(int i=0;i<model_bonds.getRowCount();i++){
	bonds.add(model_bonds.getValueAt(i, 0).toString());
}
for(int i=0;i<model_negb.getRowCount();i++){
	neg_bonds.add(model_negb.getValueAt(i, 0).toString());
}
for(int i=0;i<model_negt.getRowCount();i++){
	neg_tokens.add(model_negt.getValueAt(i, 0).toString());
}
for(int i=0;i<model_arcs.getRowCount();i++){
	String contents[]=new String[5];
	contents[0]=model_arcs.getValueAt(i, 0).toString();
	contents[1]=model_arcs.getValueAt(i, 1).toString();
	contents[2]=model_arcs.getValueAt(i, 2).toString();
	contents[3]=model_arcs.getValueAt(i, 3).toString();
	contents[4]=model_arcs.getValueAt(i, 3).toString();
	arcs.add(contents);
}
for(int i=0;i<places.size();i++){
	Place p=new Place();
	Place initp=new Place();
	p.place_name=places.get(i)[0];
	p.place_id=i;
```

```
		initp.place_id=i;
		initp.place_name=places.get(i)[0];
		petri.places.add(p);
		initial_marking.places.add(initp);
	}
	for(int i=0;i<transitions.size();i++){
		Transition t=new Transition();
		Transition initt=new Transition();
		Cell cell=new Cell();
		Cell newc=new Cell();
		t.transition_id=i;
		t.transition_name=transitions.get(i);
		cell.tr=t;
		initt.transition_id=i;
		initt.transition_name=transitions.get(i);
		newc.tr=initt;
		petri.transitions.add(t);
		petri.history.add(cell);
		initial_marking.transitions.add(initt);
		initial_marking.history.add(newc);
	}
	for(int i=0;i<tokens.size();i++){
		Token t=new Token();
		Token initt=new Token();
		t.id=i;
		t.name=tokens.get(i);
		initt.id=i;
		initt.name=tokens.get(i);
		petri.tokens.add(t);
		initial_marking.tokens.add(initt);
	}
	int count=0;
	for(int i=0;i<arcs.size();i++){
		Connection arc=new Connection();
		Connection arc1=new Connection();
		Transition t1=new Transition();
		Place p1=new Place();
		String from="";
	String to="";
	arc.connection_id=count;
	arc1.connection_id=count;
	count++;
	for(int j=0;j<petri.places.size();j++){
		if(petri.places.get(j).place_name.compareTo(arcs.get(i)[0])==0){
			arc.place=petri.places.get(j);
			p1.place_id=petri.places.get(j).place_id;
			p1.place_name=petri.places.get(j).place_name;
			arc1.place=p1;
		}
	}
	for(int j=0;j<petri.transitions.size();j++){
		if(petri.transitions.get(j).transition_name.compareTo(arcs.get(i
	)[1])==0){
			arc.transition=petri.transitions.get(j);
			t1.transition_id=petri.transitions.get(j).transition_id;

		t1.transition_name=petri.transitions.get(j).transition_name;
			arc1.transition=t1;
		}
	}
```

```
from=arcs.get(i)[2];
to=arcs.get(i)[3];
arc.from=from.charAt(0);
arc1.from=from.charAt(0);
arc.to=to.charAt(0);
arc1.to=to.charAt(0);

for(int c=0;c<arc_label.get(i).length;c++){
if(arc_label.get(i)[c].contains("not")){
if(arc_label.get(i)[c].contains("-")){
Token t=new Token();
Token tinit=new Token();
String d[]=arc_label.get(i)[c].split("not");
      t.name=d[1];
      tinit.name=d[1];
      arc.negative_bonds.add(t);
      arc1.negative_bonds.add(tinit);
}
else{
      Token t=new Token();
      Token tinit=new Token();
      String d[]=arc_label.get(i)[c].split("not");
            t.name=d[1];
            tinit.name=d[1];
            arc.negative_tokens.add(t);
            arc1.negative_tokens.add(tinit);

      }
}
else{
      if(arc_label.get(i)[c].contains("-")){
                        Token t=new Token();
                        Token tinit=new Token();
                        t.name=arc_label.get(i)[c];
                        tinit.name=arc_label.get(i)[c];
                        arc1.bonds.add(tinit);
                        arc.bonds.add(t);
                  }
                  else{
                        for(int q=0;q<petri.tokens.size();q++){

      if(petri.tokens.get(q).name.compareTo(arc_label.get(i)[c])==0){
                        Token tinit=new Token();
                        tinit.name=arc_label.get(i)[c];
                        arc1.tokens.add(tinit);
                        arc.tokens.add(petri.tokens.get(q));
                  }
                        }
                  }
            }
      }
            petri.arcs.add(arc);
            initial_marking.arcs.add(arc1);
}



for(int i=0;i<places.size();i++){
      for(int j=0;j<petri.places.size();j++){
```

```
	if(petri.places.get(j).place_name.compareTo(places.get(i)[0])==0
		&& !places.get(i)[1].isEmpty()){
		if(places.get(i)[1].contains(",")){
String[] split=places.get(i)[1].split(",");
for(int y=0;y<split.length;y++){
	if(split[y].contains("-")){
	Token bond=new Token();
	Token b1=new Token();
	bond.name=split[y];
	b1.name=split[y];
	petri.places.get(i).bonds.add(bond);
	initial_marking.places.get(i).bonds.add(b1);
	}
	else{
		for(int h=0;h<petri.tokens.size();h++){
			if(petri.tokens.get(h).name.compareTo(split[y])==0){

	petri.places.get(i).tokens.add(petri.tokens.get(h));
				Token t=new Token();
				t.name=petri.tokens.get(h).name;
				initial_marking.places.get(i).tokens.add(t);
			}
		}
	}
}
}
else{
	if(places.get(i)[1].contains("-")){
				Token bond=new Token();
				Token b1=new Token();
				bond.name=places.get(i)[1];
				b1.name=places.get(i)[1];
				petri.places.get(i).bonds.add(bond);
				initial_marking.places.get(i).bonds.add(b1);
			}
			else{
				for(int h=0;h<petri.tokens.size();h++){

	if(petri.tokens.get(h).name.compareTo(places.get(i)[1])==0){

	petri.places.get(i).tokens.add(petri.tokens.get(h));
					Token t=new Token();
					t.name=petri.tokens.get(h).name;

	initial_marking.places.get(i).tokens.add(t);
				}
			}
		}
		}
	}
}

panel.setVisible(false);
try {
	secondS = new Screen2();
} catch (IOException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
```

```
} catch (ParserConfigurationException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
} catch (SAXException e) {
	// TODO Auto-generated catch block
			e.printStackTrace();
		}
		frame.setBounds(100, 100, 600, 447);
		secondS.setVisible(true);
		frame.getContentPane().add(secondS);

	}
});
btnCreatePetriNet.setBackground(Color.ORANGE);
btnCreatePetriNet.setForeground(Color.WHITE);
btnCreatePetriNet.setFont(new Font("Tahoma", Font.BOLD, 11));
btnCreatePetriNet.setBounds(441, 649, 143, 31);
panel.add(btnCreatePetriNet);

JLabel lblDeclareTheNegative = new JLabel("Declare the negative tokens
of your marking :");
lblDeclareTheNegative.setFont(new Font("Times New Roman", Font.BOLD,
13));
lblDeclareTheNegative.setBounds(417, 292, 261, 32);
panel.add(lblDeclareTheNegative);

JLabel label_3 = new JLabel("Token name:");
label_3.setHorizontalAlignment(SwingConstants.CENTER);
label_3.setFont(new Font("Times New Roman", Font.BOLD, 13));
label_3.setBounds(383, 336, 94, 14);
panel.add(label_3);

JLabel label_4 = new JLabel("<html> Please press enter before
</b>creating the object .</html>");
label_4.setVerticalAlignment(SwingConstants.TOP);
label_4.setForeground(Color.RED);
label_4.setFont(new Font("Tahoma", Font.PLAIN, 11));
label_4.setBounds(565, 331, 147, 37);
panel.add(label_4);
textField_3 = new JTextField();
textField_3.addKeyListener(new KeyAdapter() {
	@Override
	public void keyPressed(java.awt.event.KeyEvent evt) {
		if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
			label_4.setForeground(Color.GREEN);
		}
	}

});
textField_3.setColumns(10);
textField_3.setBounds(490, 331, 71, 25);
panel.add(textField_3);


JButton button_5 = new JButton("CREATE");
button_5.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		boolean flag=false;
		boolean flag1=false;
		for(int i=0;i<model_negt.getRowCount();i++){
```

```
                  if(model_negt.getValueAt(i,
0).toString().compareTo(textField_3.getText())==0){
                        flag=true;
                  }
            }
            for(int i=0;i<model_tokens.getRowCount();i++){
                  if(model_tokens.getValueAt(i,
0).toString().compareTo(textField_3.getText())==0){
                        flag1=true;
                  }
            }
            if(!flag && flag1){
            Object[] name = new Object[1];
            name[0] = textField_3.getText();
            model_negt.addRow(name);
            textField_3.setText("");
      label_4.setForeground(Color.red);
}
      else if(flag1  && flag){
            JOptionPane.showMessageDialog(panel,
            "The tokens are unique, you have to create a token with a
different name."
            , null,JOptionPane.INFORMATION_MESSAGE);
textField_3.setText("");
}
else if(!flag1 && !flag){
      JOptionPane.showMessageDialog(panel,
      "The tokens that it will be declared as negative they have to be
declared to the tokens set."
      , null,JOptionPane.INFORMATION_MESSAGE);
textField_3.setText("");
            }
      }


});
button_5.setForeground(Color.WHITE);
button_5.setFont(new Font("Times New Roman", Font.BOLD, 13));
button_5.setBackground(new Color(50, 205, 50));
button_5.setBounds(484, 365, 89, 23);
panel.add(button_5);

JLabel lblToken_1 = new JLabel("Token 1:");
lblToken_1.setFont(new Font("Tahoma", Font.BOLD, 11));
lblToken_1.setBounds(453, 460, 65, 14);
panel.add(lblToken_1);
lblToken_1.setVisible(false);

JLabel lblToken_2 = new JLabel("Token 2:");
lblToken_2.setFont(new Font("Tahoma", Font.BOLD, 11));
lblToken_2.setBounds(560, 460, 60, 14);
panel.add(lblToken_2);
lblToken_2.setVisible(false);

JComboBox <String>comboBox_8 = new JComboBox<String>();
comboBox_8.setBounds(453, 482, 46, 20);
panel.add(comboBox_8);
comboBox_8.setVisible(false);
JComboBox <String>comboBox_9 = new JComboBox<String>();
comboBox_9.setBounds(560, 482, 46, 20);
```

```
panel.add(comboBox_9);
comboBox_9.setVisible(false);

JButton button_6 = new JButton("CREATE");
button_6.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
            String c1 = comboBox_8.getSelectedItem().toString();
            String c2 = comboBox_9.getSelectedItem().toString();

            if(c1.compareTo(c2)==0){
                  JOptionPane.showMessageDialog(panel,
                  "The bond has to be composed from two different
tokens.Please try again"
                  , null,JOptionPane.INFORMATION_MESSAGE);
      comboBox_8.setSelectedIndex(-1);
      comboBox_9.setSelectedIndex(-1);
}
else{
Object[] c = new Object[1];
c[0] = c1 + "-" + c2;
            model_negb.addRow(c);
            comboBox_8.setSelectedIndex(-1);
            comboBox_9.setSelectedIndex(-1);
      }
      }
});
button_6.setForeground(Color.WHITE);
button_6.setFont(new Font("Times New Roman", Font.BOLD, 13));
button_6.setBackground(new Color(50, 205, 50));
button_6.setBounds(478, 514, 89, 23);
panel.add(button_6);
button_6.setVisible(false);

JButton btnCreateNegativeBond = new JButton("CREATE NEGATIVE BOND");
btnCreateNegativeBond.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
            if (model_tokens.getRowCount() > 1) {
                  lblToken_2.setVisible(true);
                  lblToken_1.setVisible(true);
                  comboBox_8.setVisible(true);
                  comboBox_9.setVisible(true);
                  button_6.setVisible(true);

                  Object f = null;
                  for (int i = 0; i < model_tokens.getRowCount(); i++)
{
                        f = model_tokens.getValueAt(i, 0);
                        comboBox_8.insertItemAt(f.toString(), i);

                  }
                  for (int i = 0; i < model_tokens.getRowCount(); i++)
{
                        f = model_tokens.getValueAt(i, 0);
                        comboBox_9.insertItemAt(f.toString(), i);

                  }


                  comboBox_8.setSelectedIndex(-1);
```

```
                    comboBox_9.setSelectedIndex(-1);

                } else {
                    JOptionPane.showMessageDialog(panel,
                    "You have to define the tokens of the marking before
creating the bonds."
                    , null,JOptionPane.INFORMATION_MESSAGE);
                }
            }
});
btnCreateNegativeBond.setForeground(Color.WHITE);
btnCreateNegativeBond.setFont(new Font("Tahoma", Font.BOLD, 12));
btnCreateNegativeBond.setBackground(SystemColor.inactiveCaption);
btnCreateNegativeBond.setBounds(431, 422, 202, 23);
panel.add(btnCreateNegativeBond);


JLabel lblNegativeTokens = new JLabel("Negative tokens");
lblNegativeTokens.setHorizontalAlignment(SwingConstants.CENTER);
lblNegativeTokens.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblNegativeTokens.setBounds(837, 292, 107, 25);
panel.add(lblNegativeTokens);

JLabel lblNegativeBonds = new JLabel("Negative bonds");
lblNegativeBonds.setVerticalAlignment(SwingConstants.TOP);
lblNegativeBonds.setHorizontalAlignment(SwingConstants.CENTER);
lblNegativeBonds.setFont(new Font("Times New Roman", Font.BOLD, 13));
lblNegativeBonds.setBounds(849, 421, 97, 25);
panel.add(lblNegativeBonds);

JLabel label = new JLabel("----------------------------------"
+ "--------------------------------------------------"
+ "--------------------------------------------------"
+ "------------------------------------------------------------------
-------"
+ "----------------------------------\r\n");
label.setBounds(0, 151, 990, 14);
panel.add(label);

JLabel label_5 = new JLabel("----------------------------------"
+ "--------------------------------------"
+ "--------------------------------------------------"
+ "------------------------------------------------------------"
+ "------------------------------------------------------------------
\r\n");
label_5.setBounds(0, 271, 990, 14);
panel.add(label_5);

JLabel label_6 = new JLabel("----------------------------------"
+ "--------------------------------------------------"
+ "--------------------------------------------------"
+ "--------------------------------------------------"
+ "------------------------------------------------------------\r\n");
label_6.setBounds(0, 394, 990, 14);
panel.add(label_6);

JLabel label_7 = new JLabel("----------------------------------------
---"
+ "--------------------------------------------------------"
+ "--------------------------------------------------------"
```

```
+ "-------------------------------------------------------------"
+ "-------------------------\r\n");
label_7.setVerticalAlignment(SwingConstants.TOP);
label_7.setBounds(0, 533, 990, 14);
panel.add(label_7);

JButton button_7 = new JButton("BACK");
button_7.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
            setVisible(false);
            frame.setBounds(100, 100, 633, 440);
            MainWindow.panel.setVisible(true);
      }
});
button_7.setForeground(Color.WHITE);
button_7.setFont(new Font("Times New Roman", Font.BOLD, 12));
            button_7.setBackground(SystemColor.activeCaption);
            button_7.setBounds(0, 0, 89, 23);
            panel.add(button_7);



      }
}
```

```
import java.awt.Color;
import java.awt.Desktop;
import java.awt.Font;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.SwingConstants;
import javax.swing.UIManager;
import javax.swing.border.EmptyBorder;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

import javax.swing.ImageIcon;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.awt.event.ActionEvent;
import java.awt.SystemColor;

public class Explanation extends JPanel {

      /**
 * Create the panel.
 */
public Explanation(JFrame frame) {
      setBounds(100, 100, 566, 399);
      setBackground(Color.WHITE);
```

```
		setBorder(new EmptyBorder(5, 5, 5, 5));

		setLayout(null);

		JTextArea textArea = new JTextArea();
		textArea.setFont(new Font("Times New Roman", Font.PLAIN, 15));
textArea.setBackground(Color.WHITE);
textArea.setEditable(false);
textArea.setBounds(37, 39, 405, 262);
String directions=" 1.Be sure that the Folder Obeo designer-Community
is saved \n    at the location C:Users/Your folder. \n\n 2.Click to
button 'Open Obeo designer' and then when a window \n will appear
click on 'Launch' button. \n\n 3.From the Obeo designer environment
Click Run from the toolbar. \n\n 4.Go to representation tab. \n\n
5.You can now draw the petri net from palette. \n\n 6.When you will
finish, save your Petri net \n diagram and click 'Finish' button.";

textArea.setText(directions);
add(textArea);

JLabel lblHowToDraw = new JLabel("How to draw your Petri Net");
lblHowToDraw.setHorizontalAlignment(SwingConstants.CENTER);
lblHowToDraw.setForeground(new Color(0, 0, 0));
lblHowToDraw.setBackground(UIManager.getColor("Separator.foreground"))
;
lblHowToDraw.setFont(new Font("Times New Roman", Font.BOLD, 18));
lblHowToDraw.setBounds(115, 0, 387, 28);
add(lblHowToDraw);

JButton btnNewButton = new JButton("Open Obeo Designer");
btnNewButton.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent arg0) {
		File exec=new File("C:\\Users\\"
		+ "Pantelina\\Downloads\\ObeoDesigner-
Community\\obeodesigner.exe");
try {
		Desktop.getDesktop().open(exec);
} catch (IOException e) {
		// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
});
btnNewButton.setFont(new Font("Times New Roman", Font.BOLD, 12));
btnNewButton.setBackground(new Color(175, 238, 238));
btnNewButton.setBounds(117, 323, 159, 34);
add(btnNewButton);

JButton btnNewButton_1 = new JButton("Finish");
btnNewButton_1.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
try {
Screen2 s2=new Screen2();
setVisible(false);
frame.getContentPane().add(s2);
s2.setVisible(true);
} catch (IOException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
} catch (ParserConfigurationException e1) {
```

```
// TODO Auto-generated catch block
e1.printStackTrace();
} catch (SAXException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
            }

      }
});
btnNewButton_1.setBackground(new Color(50, 205, 50));
btnNewButton_1.setFont(new Font("Times New Roman", Font.BOLD, 13));
btnNewButton_1.setBounds(374, 323, 103, 34);
add(btnNewButton_1);

JLabel label = new JLabel("");
label.setIcon(new
ImageIcon(Explanation.class.getResource("/images/hhhhh.png")));
label.setBounds(459, 137, 39, 34);
add(label);

JLabel label_1 = new JLabel("");
label_1.setBounds(184, 175, 372, 32);
add(label_1);
label_1.setIcon(new
ImageIcon(Explanation.class.getResource("/images/representation.png"))
);

JLabel label_2 = new JLabel("");
label_2.setIcon(new
ImageIcon(Explanation.class.getResource("/images/palette.png")));
label_2.setBounds(456, 218, 74, 25);
add(label_2);

JButton button = new JButton("BACK");
button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
            setVisible(false);
            frame.setBounds(100, 100, 633, 440);
            MainWindow.panel.setVisible(true);
      }
});
button.setForeground(Color.WHITE);
button.setFont(new Font("Times New Roman", Font.BOLD, 12));
button.setBackground(SystemColor.activeCaption);
button.setBounds(0, 0, 89, 23);
add(button);

JLabel label_3 = new JLabel("");
label_3.setIcon(new
ImageIcon(Explanation.class.getResource("/images/save.png")));
            label_3.setBounds(452, 265, 46, 25);
            add(label_3);
      }
}
```

```
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;

import java.awt.Color;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.ArrayList;
import java.awt.event.ActionEvent;
import javax.swing.SwingConstants;
import javax.swing.table.DefaultTableModel;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;

import org.xml.sax.SAXException;

import java.awt.SystemColor;
import java.awt.Font;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JComboBox;
import javax.swing.JTextArea;
import javax.swing.JTextPane;
import javax.swing.ComboBoxModel;
import javax.swing.DropMode;

public class Screen2 extends JPanel {
	public static boolean first=true;
	public static Object[] columns={"Place","Marking"};
public static int sum=0;


/**
 * Create the panel.
 * @throws SAXException
 * @throws ParserConfigurationException
 * @throws IOException
 */
public Screen2() throws IOException, ParserConfigurationException,
SAXException {
	setBackground(new Color(255, 255, 255));
	setLayout(null);
	setBounds(100, 100, 709, 447);
	JLabel lblForwardEnabledTransitions = new JLabel
			("<html>Forward-enabled  </b> transitions
are:</html>");
lblForwardEnabledTransitions.setHorizontalAlignment(SwingConstants.CEN
TER);
lblForwardEnabledTransitions.setFont(new Font("Times New Roman",
Font.BOLD, 12));
lblForwardEnabledTransitions.setBounds(10, 0, 107, 55);
add(lblForwardEnabledTransitions);
```

```
DefaultTableModel model=new DefaultTableModel();
JTable table=new JTable();
table.setBackground(Color.WHITE);
JScrollPane scroll=new JScrollPane();
add(scroll);
scroll.setViewportView(table);
model.setColumnIdentifiers(columns);
table.setModel(model);
table.setVisible(true);
scroll.setBounds(113, 200, 385, 204);


/**JTextPane textArea = new JTextPane();
textArea.setForeground(Color.BLACK);
textArea.setBackground(SystemColor.control);
textArea.setFont(new Font("Times New Roman", Font.PLAIN, 12));
textArea.setEditable(false);
textArea.setBounds(104, 203, 385, 244);
textArea.setAlignmentX(CENTER_ALIGNMENT);
add(textArea);*/

JButton btnNewButton = new JButton("<html> Find forward-  </b> enabled
transitions </html>");
btnNewButton.setVerticalAlignment(SwingConstants.TOP);
btnNewButton.setFont(new Font("Times New Roman", Font.BOLD, 14));
btnNewButton.setForeground(Color.WHITE);
btnNewButton.setBackground(SystemColor.activeCaption);

JComboBox <String>forward_enabled_tr = new JComboBox<String>();
forward_enabled_tr.setBackground(Color.WHITE);
forward_enabled_tr.setBounds(164, 4, 59, 32);


JLabel forward_lbl = new JLabel("");
forward_lbl.setFont(new Font("Tahoma", Font.PLAIN, 9));
forward_lbl.setBounds(233, 5, 126, 31);
add(forward_lbl);

JLabel backtrack_lbl = new JLabel("");
backtrack_lbl.setFont(new Font("Tahoma", Font.PLAIN, 9));
backtrack_lbl.setBounds(233, 51, 126, 31);
add(backtrack_lbl);

JLabel causal_lbl = new JLabel("");
causal_lbl.setFont(new Font("Tahoma", Font.PLAIN, 9));
causal_lbl.setBounds(233, 100, 126, 31);
add(causal_lbl);

JLabel ofc_lbl = new JLabel("");
ofc_lbl.setFont(new Font("Tahoma", Font.PLAIN, 9));
ofc_lbl.setBounds(245, 161, 114, 31);
add(ofc_lbl);

add(forward_enabled_tr);
if(first==true){
      Forward_algorithm.find_choice();
      first=false;
      }
```

```
ArrayList<String>choices=new ArrayList<String>();
btnNewButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
            forward_enabled_tr.removeAllItems();
            choices.clear();

      Forward_algorithm.forward_enabled(Forward_algorithm.forward_move
s.get(0));
            for(int i=0;i<Forward_algorithm.forward_moves.get(0).
                        transitions.size();i++){

      if(Forward_algorithm.forward_moves.get(0).transitions.
                              get(i).enabled_for_execution==true){

      choices.add(Forward_algorithm.forward_moves.get(0).

      transitions.get(i).transition_name);
                  }
            }

            forward_enabled_tr.insertItemAt("--", 0);
forward_enabled_tr.setSelectedIndex(0);
if(choices.size()==0){
      forward_lbl.setText("<html>There are no </b> forward enabled
transitions.</html>");
}
else if(choices.size()==1){
      forward_lbl.setText("<html>There is "+choices.size()
      +" </b>forward enabled transition</html>");
}
else{
      forward_lbl.setText("<html>There are "+choices.size()
      +" </b>forward enabled transitions</html>");
            }
            for(int i=0;i<choices.size();i++){
                  forward_enabled_tr.addItem(choices.get(i));

            }

      }


});

btnNewButton.setBounds(369, 0, 172, 49);
add(btnNewButton);



JLabel lblBacktrackenabledTransitionsAre = new JLabel
("<html>Backtrack-enabled </b> transitions are:</html>");
lblBacktrackenabledTransitionsAre.setHorizontalAlignment(SwingConstant
s.CENTER);
lblBacktrackenabledTransitionsAre.setFont(new Font("Times New Roman",
Font.BOLD, 12));
lblBacktrackenabledTransitionsAre.setBounds(10, 51, 107, 41);
add(lblBacktrackenabledTransitionsAre);

JLabel lblcausalenabledTransitionsAre = new JLabel
("<html>Causal-enabled  </b> transitions are:</html>");
```

```
lblcausalenabledTransitionsAre.setHorizontalAlignment(SwingConstants.C
ENTER);
lblcausalenabledTransitionsAre.setFont(new Font("Times New Roman",
Font.BOLD, 12));
lblcausalenabledTransitionsAre.setBounds(10, 90, 107, 49);
add(lblcausalenabledTransitionsAre);

JLabel lbloutOfCausalenabled = new JLabel
("<html>Out of causal-enabled  </b> transitions are:</html>");
lbloutOfCausalenabled.setHorizontalAlignment(SwingConstants.CENTER);
lbloutOfCausalenabled.setFont(new Font("Times New Roman", Font.BOLD,
12));
lbloutOfCausalenabled.setBounds(10, 137, 107, 55);
add(lbloutOfCausalenabled);

JComboBox<String> backtrack_comboBox = new JComboBox<String>();
backtrack_comboBox.setBackground(Color.WHITE);
backtrack_comboBox.setBounds(164, 51, 59, 32);
add(backtrack_comboBox);

JComboBox<String> causal_comboBox_1 = new JComboBox<String>();
causal_comboBox_1.setBackground(Color.WHITE);
causal_comboBox_1.setBounds(164, 101, 59, 32);
add(causal_comboBox_1);

JComboBox<String> ofc_comboBox_2 = new JComboBox<String>();
ofc_comboBox_2.setBackground(Color.WHITE);
ofc_comboBox_2.setBounds(164, 160, 59, 32);
add(ofc_comboBox_2);


JButton btnFindReversedEnabled = new JButton
("<html> Find reversed</b> enabled  transitions </html>");
ArrayList<String> backtrack=new ArrayList<String>();
ArrayList<String> causal=new ArrayList<String>();
ArrayList<String>out_of_causal=new ArrayList<String>();

btnFindReversedEnabled.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
            backtrack_comboBox.removeAllItems();
            causal_comboBox_1.removeAllItems();
            ofc_comboBox_2.removeAllItems();
            backtrack.clear();
            causal.clear();
            out_of_causal.clear();
            int
max=Backtracking_algorithm.backtracking_enabled(Forward_algorithm.forw
ard_moves.get(0));

      Causal_reversing.co_enabled(Forward_algorithm.forward_moves.get(
0));

      Out_of_causal.o_enabled(Forward_algorithm.forward_moves.get(0));

            for(int
i=0;i<Forward_algorithm.forward_moves.get(0).history.size();i++){

      if(Forward_algorithm.forward_moves.get(0).history.get(i).history
!=max){
                        Forward_algorithm.forward_moves.get(0).
```

```
                        history.get(i).tr.backtrack_enable=false;
                    }
                }
                for(int
i=0;i<Forward_algorithm.forward_moves.get(0).transitions.size();i++){

      if(Forward_algorithm.forward_moves.get(0).transitions.get(i).bac
ktrack_enable==true){

      backtrack.add(Forward_algorithm.forward_moves.get(0)

      .transitions.get(i).transition_name);
                  }

      if(Forward_algorithm.forward_moves.get(0).transitions.get(i).co_
enabled==true){

      causal.add(Forward_algorithm.forward_moves.get(0).

      transitions.get(i).transition_name);
                  }

      if(Forward_algorithm.forward_moves.get(0).transitions.get(i).o_e
nabled==true){

      out_of_causal.add(Forward_algorithm.forward_moves.get(0)

      .transitions.get(i).transition_name);
            }
            }

            backtrack_comboBox.insertItemAt("--", 0);
backtrack_comboBox.setSelectedIndex(0);
causal_comboBox_1.insertItemAt("--", 0);
causal_comboBox_1.setSelectedIndex(0);
ofc_comboBox_2.insertItemAt("--", 0);
ofc_comboBox_2.setSelectedIndex(0);

if(backtrack.size()==0){
      backtrack_lbl.setText("<html>No backtrack enable</b>
transitions.</html>");
}
else if(backtrack.size()==1){
      backtrack_lbl.setText("<html>There is "+backtrack.size()+
                  "</b> backtrack enabled transition.</html>");
}
else{
      backtrack_lbl.setText("<html>There are "+backtrack.size()+
            "</b> backtrack enabled transitions.</html>");
}

if(causal.size()==0){
      causal_lbl.setText("<html>No causal enabled
</b>transiotions.</html>");
}
else if(causal.size()==1){
      causal_lbl.setText("<html>There is "+causal.size()+
            "</b> causal enabled transition.</html>");
}
else{
```

```
		causal_lbl.setText("<html>There are "+causal.size()+
				"</b> causal enabled transitions.</html>");
	}

	if(out_of_causal.size()==0){
		ofc_lbl.setText("<html>No out_of causal enabled
	</b>transiotions.</html>");
	}
	else if(out_of_causal.size()==1){
		ofc_lbl.setText("<html>There is "+out_of_causal.size()
		+"</b> out of causal enabled transition.</html>");

	}
	else{
		ofc_lbl.setText("<html>There are "+out_of_causal.size()+
				"</b> out of causal enabled transitions.</html>");

				}
				for(int i=0;i<backtrack.size();i++){
					backtrack_comboBox.addItem(backtrack.get(i));
				}

				for(int i=0;i<causal.size();i++){
					causal_comboBox_1.addItem(causal.get(i));
				}

				for(int i=0;i<out_of_causal.size();i++){
					ofc_comboBox_2.addItem(out_of_causal.get(i));
				}
		}
	});

	JButton btnNewButton_1 = new JButton("Next move");
	btnNewButton_1.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent arg0) {
				model.getDataVector().removeAllElements();
				table.removeAll();
				String sel="";
	String get="--";
	String get1="--";
	String get2="--";
	String get3="--";
	sum=0;
	if(forward_enabled_tr.getSelectedItem()!=null){
	 get=forward_enabled_tr.getSelectedItem().toString();
	 if(forward_enabled_tr.getSelectedIndex()!=0){
			sum++;
			}

	}
	if(backtrack_comboBox.getSelectedItem()!=null){
	 get1=backtrack_comboBox.getSelectedItem().toString();
	 if(backtrack_comboBox.getSelectedIndex()!=0){
			sum++;
			}

	}
	if(causal_comboBox_1.getSelectedItem()!=null){
	 get2=causal_comboBox_1.getSelectedItem().toString();
	 if(causal_comboBox_1.getSelectedIndex()!=0){
```

```
        sum++;
        }

}
if(ofc_comboBox_2.getSelectedItem()!=null){
 get3=ofc_comboBox_2.getSelectedItem().toString();
 if(ofc_comboBox_2.getSelectedIndex()!=0){
        sum++;
        }

}
//System.out.println(sum);
if(sum>1){
JOptionPane.showMessageDialog(Screen2.this,
"You have to select only one move to execute at a time!",null,
JOptionPane.INFORMATION_MESSAGE);
}
else if(get.compareTo("--")!=0){
sel=forward_enabled_tr.getSelectedItem().toString();
try {
      Forward_algorithm.forward_execution(sel);
} catch (ParserConfigurationException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
} catch (TransformerException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
}

      for (int i = 0; i <Forward_algorithm.forward_moves.size(); i++)
{

      for (int j = 0; j <Forward_algorithm.
forward_moves.get(i).places.size(); j++) {
            Object[]table=new Object[2];
            String print="";
table[0]=Forward_algorithm.forward_moves.get(i).places.get(j).place_na
me ;
if (Forward_algorithm.forward_moves.get(i).places.get(j).bonds.size()
!= 0) {
      for (int k = 0; k <
Forward_algorithm.forward_moves.get(i).places.
                  get(j).bonds.size(); k++) {
      print+=Forward_algorithm.forward_moves.get(i).places.get(j)
                  .bonds.get(k).name + " ";
      }

} else {
      for (int k = 0; k <Forward_algorithm. forward_moves.get(i).
                  places.get(j).tokens.size(); k++) {
            print+=Forward_algorithm.forward_moves.get(i).
                        places.get(j).tokens.get(k).name + " ";
                  }

            }
            table[1]=print;
            model.addRow(table);
      }
      }
```

```
}
else if(get1.compareTo("--")!=0){
sel=backtrack_comboBox.getSelectedItem().toString();
Backtracking_algorithm.backtrack_execution(sel);
for (int i = 0; i <Forward_algorithm.forward_moves.size(); i++) {

      for (int j = 0; j < Forward_algorithm.forward_moves.get(i).
                  places.size(); j++) {
            Object[]table=new Object[2];
            String print="";
table[0]=Forward_algorithm.forward_moves.get(i).places.get(j).place_na
me;
if (Forward_algorithm.forward_moves.get(i).places.get(j).bonds.size()
!= 0) {
      for (int k = 0; k <Forward_algorithm.forward_moves.get(i)
                  .places.get(j).bonds.size(); k++) {
            print+=Forward_algorithm.forward_moves.get(i).places.
                        get(j).bonds.get(k).name + " ";
      }
} else {
      for (int k = 0; k < Forward_algorithm.forward_moves.get(i).
                  places.get(j).tokens.size(); k++) {
            print+=Forward_algorithm.forward_moves.get(i).
                        places.get(j).tokens.get(k).name + " ";
                  }
            }

      table[1]=print;
      model.addRow(table);
      }
}
}
else if(get2.compareTo("--")!=0){
sel=causal_comboBox_1.getSelectedItem().toString();
Causal_reversing.causal_execution(sel);

for (int i = 0; i <Forward_algorithm.forward_moves.size(); i++) {

      for (int j = 0; j <
Forward_algorithm.forward_moves.get(i).places.size(); j++) {
            Object[]table=new Object[2];
            String print="";
table[0]=Forward_algorithm.forward_moves.get(i).places.get(j).place_na
me;
if (Forward_algorithm.forward_moves.get(i).places.get(j).bonds.size()
!= 0) {
      for (int k = 0; k < Forward_algorithm.forward_moves.get(i).
                  places.get(j).bonds.size(); k++) {
            print+=Forward_algorithm.forward_moves.get(i).places.
                        get(j).bonds.get(k).name + " ";
      }

} else {
      for (int k = 0; k < Forward_algorithm.forward_moves.get(i).
                  places.get(j).tokens.size(); k++) {
            print+=Forward_algorithm.forward_moves.get(i).
                        places.get(j).tokens.get(k).name + " ";
                  }

            }
```

```
      table[1]=print;
      model.addRow(table);
      }

}
}
else if(get3.compareTo("--")!=0){
sel=ofc_comboBox_2.getSelectedItem().toString();
Out_of_causal.out_of_causal_execution(sel);
for (int i = 0; i <Forward_algorithm.forward_moves.size(); i++) {

      for (int j = 0; j <
Forward_algorithm.forward_moves.get(i).places.size(); j++) {
            Object[]table=new Object[2];
            String print="";
table[0]=Forward_algorithm.forward_moves.get(i).places.get(j).place_na
me;
if (Forward_algorithm.forward_moves.get(i).places.get(j).bonds.size()
!= 0) {
      for (int k = 0; k <
Forward_algorithm.forward_moves.get(i).places.get(j)
            .bonds.size(); k++) {

      print+=Forward_algorithm.forward_moves.get(i).places.get(j)
            .bonds.get(k).name + " ";
      }

} else {
      for (int k = 0; k <
Forward_algorithm.forward_moves.get(i).places.get(j)
                  .tokens.size(); k++) {

      print+=Forward_algorithm.forward_moves.get(i).places.get(j)
                        .tokens.get(k).name + " ";
                              }

                        }

                        table[1]=print;
                        model.addRow(table);
                        }
                  }
            }


      }
});

btnNewButton_1.setFont(new Font("Times New Roman", Font.BOLD, 14));
btnNewButton_1.setForeground(Color.WHITE);
btnNewButton_1.setBackground(SystemColor.activeCaption);
btnNewButton_1.setBounds(369, 137, 172, 49);
add(btnNewButton_1);

btnFindReversedEnabled.setVerticalAlignment(SwingConstants.TOP);
btnFindReversedEnabled.setForeground(Color.WHITE);
btnFindReversedEnabled.setFont(new Font("Times New Roman", Font.BOLD,
14));
btnFindReversedEnabled.setBackground(SystemColor.activeCaption);
```

```
btnFindReversedEnabled.setBounds(369, 65, 172, 49);
add(btnFindReversedEnabled);
      }
}
```

# Appendix D

## Parser's code

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Iterator;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class Parser {
      public static PetriNet initial_petrinet = new PetriNet();
      public static PetriNet initial_marking=new PetriNet();
      public static ArrayList<Token> bonds = new ArrayList<Token>();
      public static ArrayList<Token> neg_tokens = new
ArrayList<Token>();
      public static ArrayList<Token> neg_bonds = new
ArrayList<Token>();
      public static ArrayList<Object[]>use=new ArrayList<Object[]>();
      public static File file;

      public static void readArcs(Document doc) {
            String token = "";
String bond = "";
String ntoken = "";
String nbond = "";
int from;
int to;
String place = "";
String transition = "";
String arc_name = "";
```

```
int arc_id;
NodeList nList = doc.getElementsByTagName("arc");
for (int temp = 0; temp < nList.getLength(); temp++) {
      Node nNode = nList.item(temp);
      Connection arc=new Connection();
      Connection init=new Connection();
      try {
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                  Element eElement = (Element) nNode;
                  if (eElement.hasAttribute("token")) {
      token = eElement.getAttribute("token");
      int id=Integer.parseInt(token);
      for(int i=0;i<initial_petrinet.tokens.size();i++){
            if(initial_petrinet.tokens.get(i).id==id){
                  arc.tokens.add(initial_petrinet.tokens.get(i));
                  Token n=new Token();
                  n.id=initial_petrinet.tokens.get(i).id;
                  n.name=initial_petrinet.tokens.get(i).name;
                  init.tokens.add(n);
            }
      }
}
if (eElement.hasAttribute("bond")) {
      ArrayList<Integer> bond_list=new ArrayList<Integer>();
      bond = eElement.getAttribute("bond");
      if(bond.contains(" ")){
      String[]split=bond.split(" ");
      int id=0;
      for(int h=0;h<split.length;h++){
       id=Integer.parseInt(split[h])-200;
       bond_list.add(id);
      }
      for(int i=0;i<bonds.size();i++){
            for(int g=0;g<bond_list.size();g++){
            if(bonds.get(i).id==bond_list.get(g)){
                  arc.bonds.add(bonds.get(i));
                  init.bonds.add(bonds.get(i));
            }
            }
      }
}
      else{
            bond = eElement.getAttribute("bond");
            int  id=Integer.parseInt(bond)-200;


            for(int i=0;i<bonds.size();i++){
                  if(bonds.get(i).id==id){
                        arc.bonds.add(bonds.get(i));
                        init.bonds.add(bonds.get(i));

                  }
            }
      }
}
if (eElement.hasAttribute("negative_token")) {
      ArrayList<Integer>ntokens=new ArrayList<Integer>();
      ntoken = eElement.getAttribute("negative_token");
      if(ntoken.contains(" ")){
      String[]split=ntoken.split(" ");
```

```
		int id=0;
		for(int h=0;h<split.length;h++){
		 id=Integer.parseInt(split[h])-300;
		 ntokens.add(id);
		}
		for(int i=0;i<neg_tokens.size();i++){
			for(int k=0;k<ntokens.size();k++){
			if(neg_tokens.get(i).id==ntokens.get(k)){
				arc.negative_tokens.add(neg_tokens.get(i));
				init.negative_tokens.add(neg_tokens.get(i));
			}
		}
		}
}
		else{
			ntoken = eElement.getAttribute("negative_token");
			int  id=Integer.parseInt(bond)-300;
			for(int i=0;i<neg_tokens.size();i++){
				if(neg_tokens.get(i).id==id){
				arc.negative_tokens.add(neg_tokens.get(i));
				init.negative_tokens.add(neg_tokens.get(i));

				}
			}
		}
}
if (eElement.hasAttribute("negative_bond")) {
		ArrayList<Integer>nbonds=new ArrayList<Integer>();

		nbond = eElement.getAttribute("negative_bond");
		if(nbond.contains(" ")){
		String[]split=ntoken.split(" ");
		int id=0;
		for(int h=0;h<split.length;h++){
		 id=Integer.parseInt(split[h])-400;
		 nbonds.add(id);
		}

		for(int i=0;i<neg_bonds.size();i++){
			for(int k=0;k<nbonds.size();k++){
			if(neg_bonds.get(i).id==nbonds.get(k)){
				arc.negative_bonds.add(neg_bonds.get(i));
				init.negative_bonds.add(neg_bonds.get(i));
			}
			}
		}
}
		else{
			nbond = eElement.getAttribute("negative_bond");
			int  id=Integer.parseInt(bond)-400;
			for(int i=0;i<neg_bonds.size();i++){
				if(neg_bonds.get(i).id==id){
				arc.negative_bonds.add(neg_bonds.get(i));
				init.negative_bonds.add(neg_bonds.get(i));

				}
			}
		}
}
from = Integer.parseInt(eElement.getAttribute("from"));
```

```
to = Integer.parseInt(eElement.getAttribute("to"));

char fromc = (char) from;
char toc = (char) to;

arc.from=fromc;
arc.to=toc;
init.from=fromc;
init.to=toc;


if (eElement.hasAttribute("transition")) {
      transition = eElement.getAttribute("transition");
      String cut[]=transition.split("\\[10");
      cut=cut[1].split("\\]");
      int id=Integer.parseInt(cut[0]);
      for(int i=0;i<initial_petrinet.transitions.size();i++){
            if(initial_petrinet.transitions.get(i).transition_id==id){
                  arc.transition=initial_petrinet.transitions.get(i);
                  Transition t=new Transition();

      t.transition_id=initial_petrinet.transitions.get(i).transition_i
d;

      t.transition_name=initial_petrinet.transitions.get(i).transition
_name;
                  init.transition=t;
            }
      }
}
if (eElement.hasAttribute("place")) {
      place = eElement.getAttribute("place");
      String cut[]=place.split("\\[");
      cut=cut[1].split("\\]");
      int id=Integer.parseInt(cut[0]);

      for(int i=0;i<initial_petrinet.places.size();i++){
            if(initial_petrinet.places.get(i).place_id==id){
                  arc.place=initial_petrinet.places.get(i);
                  Place t=new Place();
                  t.place_id=initial_petrinet.places.get(i).place_id;

      t.place_name=initial_petrinet.places.get(i).place_name;
                  init.place=t;

            }
      }
}
arc_name = eElement.getAttribute("arc_name");
arc_id = Integer.parseInt(eElement.getElementsByTagName("arc_id").
            item(0).getTextContent());
                        arc.connection_id=arc_id;
                        init.connection_id=arc_id;
                        initial_petrinet.arcs.add(arc);
                        initial_marking.arcs.add(init);
                  }

            } catch (Exception e) {
                  e.printStackTrace();
```

```
			}
		}
	}

	public static void readPlaces(Document doc) {

		NodeList nList = doc.getElementsByTagName("place");

	String token = "";
	String bond = "";
	String arc_id;
	String place_name = "";
	int place_id;
	for (int temp = 0; temp < nList.getLength(); temp++) {
		Node nNode = nList.item(temp);
		Place newp = new Place();
		Place init=new Place();
		try {
			if (nNode.getNodeType() == Node.ELEMENT_NODE) {
				Element eElement = (Element) nNode;

				if (eElement.hasAttribute("token")) {
		ArrayList<Integer>tokens=new ArrayList<Integer>();
		token = eElement.getAttribute("token");
		int id=0;
		if(token.contains(" ")){
		String g[]=token.split(" ");
		for(int k=0;k<g.length;k++){
			id = Integer.parseInt(g[k]);
			tokens.add(id);
		}
		for (int i = 0; i < initial_petrinet.tokens.size(); i++) {
			for(int j=0;j<tokens.size();j++){
			if (initial_petrinet.tokens.get(i).id == tokens.get(j)) {
				newp.tokens.add(initial_petrinet.tokens.get(i));
				init.tokens.add(initial_petrinet.tokens.get(i));
			}
		}
	}
		}
		else{
			id = Integer.parseInt(token);

		for (int i = 0; i < initial_petrinet.tokens.size(); i++) {

			if (initial_petrinet.tokens.get(i).id == id) {
				newp.tokens.add(initial_petrinet.tokens.get(i));
				init.tokens.add(initial_petrinet.tokens.get(i));

		}

		}
	}
	}
	if (eElement.hasAttribute("bond")) {
		bond = eElement.getAttribute("bond");
		if(!bond.contains(" ")){
		int id = Integer.parseInt(bond)-200;
		for (int i = 0; i < bonds.size(); i++) {
			if (bonds.get(i).id == id) {
```

```
                        newp.bonds.add(bonds.get(i));
                        init.bonds.add(bonds.get(i));
                  }
            }
      }
            else{
                  int id=0;
                  ArrayList<Integer>bonds1=new ArrayList<Integer>();
                  String f[]=bond.split(" ");
                  for(int k=0;k<f.length;k++){
                              id = Integer.parseInt(f[k]);
                              bonds1.add(id);
                  }
                  for (int i = 0; i < bonds.size(); i++) {
                        for(int j=0;j<bonds1.size();j++){
                              if(bonds.get(i).id==bonds1.get(j)){
                                    newp.bonds.add(bonds.get(i));
                                    init.bonds.add(bonds.get(i));
                              }


                  }
            }
            }
      }
      if (eElement.hasAttribute("arc")) {
            arc_id = eElement.getAttribute("arc");
            String c1[];
            if(arc_id.contains(" ")){
            c1=arc_id.split(" ");
            for(int j=0;j<c1.length;j++){
            String cut[]=c1[j].split("//@");
            cut=cut[1].split("arc.");
            int arcid=Integer.parseInt(cut[1])+1;
            newp.arc_id.add(arcid);
            init.arc_id.add(arcid);
            }
            }
            else{
                  String cut[]=arc_id.split("//@");
                  cut=cut[1].split("arc.");
                  int arcid=Integer.parseInt(cut[1])+1;
                  newp.arc_id.add(arcid);
                  init.arc_id.add(arcid);
            }
      }
      place_name = eElement.getElementsByTagName("place_name").item(0)
            .getTextContent();
      newp.place_name=place_name;
      place_id = Integer.parseInt(eElement.getElementsByTagName("place_id")
                                          .item(0).getTextContent());
                              newp.place_id=place_id;
                              initial_petrinet.places.add(newp);
                              init.place_id=place_id;
                              init.place_name=place_name;
                              initial_marking.places.add(init);
                        }

                  } catch (Exception e) {
                        e.printStackTrace();
```

```
			}
		}
}

public static void readTokens(Document doc) {

		NodeList nList = doc.getElementsByTagName("token");
String name = "";
int id;
for (int temp = 0; temp < nList.getLength(); temp++) {
		Node nNode = nList.item(temp);
		Token token1 = new Token();
		try {
			if (nNode.getNodeType() == Node.ELEMENT_NODE) {
				Element eElement = (Element) nNode;

				id =
Integer.parseInt(eElement.getAttribute("token_id"));
token1.id = id;

name = eElement.getAttribute("token_name");
					token1.name = name;
					initial_petrinet.tokens.add(token1);
					Token newt = new Token();
					newt.name = name;
					initial_marking.tokens.add(newt);
					Object[]newe=new Object[2];
					newe[0]=id;
					newe[1]=name;
					use.add(newe);

				}

			} catch (Exception e) {
				e.printStackTrace();
			}
		}
}

public static void readBonds(Document doc) {

		NodeList nList = doc.getElementsByTagName("bond");
String name = "";
int id;
for (int temp = 0; temp < nList.getLength(); temp++) {
		Node nNode = nList.item(temp);
		Token bond = new Token();
		try {
			if (nNode.getNodeType() == Node.ELEMENT_NODE) {
				Element eElement = (Element) nNode;
				id =
Integer.parseInt(eElement.getAttribute("bond_id"));
bond.id = id-200;
name = eElement.getAttribute("bond_name");
					bond.name = name;
					bonds.add(bond);

				}

			} catch (Exception e) {
```

```
                  e.printStackTrace();
            }
      }
}

public static void readNTokens(Document doc) {
      NodeList nList = doc.getElementsByTagName("negative_token");
String name = "";
int id;
for (int temp = 0; temp < nList.getLength(); temp++) {
      Node nNode = nList.item(temp);
      Token neg = new Token();
      try {
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                  Element eElement = (Element) nNode;

                  id =
Integer.parseInt(eElement.getAttribute("token_id"))-300;
neg.id = id;
String p[]=eElement.getAttribute("token_name").split("-");
                        name =p[1];
                        neg.name =name;
                        neg_tokens.add(neg);
                  }

            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}

public static void readNBonds(Document doc) {
      NodeList nList = doc.getElementsByTagName("negative_bond");
String name = "";
int id;
for (int temp = 0; temp < nList.getLength(); temp++) {
      Node nNode = nList.item(temp);
      Token neg_bond = new Token();
      System.out.println("\nCurrent Element :" + nNode.getNodeName());
try {
      if (nNode.getNodeType() == Node.ELEMENT_NODE) {
            Element eElement = (Element) nNode;
            id = Integer.parseInt(eElement.getAttribute("bond_id"))-
400;
neg_bond.id = id;
String[]p=eElement.getAttribute("bond_name").split("-");
                        name = p[1];
                        neg_bond.name = name;
                        neg_bonds.add(neg_bond);
                  }

            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}

public static void readTransitions(Document doc) {
      NodeList nList = doc.getElementsByTagName("transition");
String name = "";
```

```
int id;
String arc = "";
for (int temp = 0; temp < nList.getLength(); temp++) {
      Node nNode = nList.item(temp);
      Transition tr=new Transition();
      Transition init=new Transition();
      try {
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                  Element eElement = (Element) nNode;

                  if (eElement.hasAttribute("arc")) {
      String arc_id = eElement.getAttribute("arc");
      arc_id = eElement.getAttribute("arc");
      String c1[];
      if(arc_id.contains(" ")){
      c1=arc_id.split(" ");
      for(int j=0;j<c1.length;j++){
      String cut[]=c1[j].split("//@");
      cut=cut[1].split("arc.");
      int arcid=Integer.parseInt(cut[1])+1;
      tr.arc_id.add(arcid);
      init.arc_id.add(arcid);
      }
      }
      else{
            String cut[]=arc_id.split("//@");
            cut=cut[1].split("arc.");
            int arcid=Integer.parseInt(cut[1])+1;
            tr.arc_id.add(arcid);
            init.arc_id.add(arcid);
      }

}
String id2=eElement.getElementsByTagName("transition_id")
            .item(0).getTextContent();
String split[]=id2.split("10");

tr.transition_id=Integer.parseInt(split[1]);
init.transition_id=Integer.parseInt(split[1]);;
name = eElement.getElementsByTagName("transition_name")
                                    .item(0).getTextContent();
                        tr.transition_name=name;
                        init.transition_name=name;
                        Cell cell=new Cell();
                        Cell cell1=new Cell();
                        cell.tr=tr;
                        cell1.tr=init;
                        initial_petrinet.transitions.add(tr);
                        initial_petrinet.history.add(cell);
                        initial_marking.transitions.add(init);
                        initial_marking.history.add(cell1);

                  }

            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

```
public static void fillArcs(){
      for(int i=0;i<initial_petrinet.places.size();i++){
            for(int
k=0;k<initial_petrinet.places.get(i).arc_id.size();k++){
            if(initial_petrinet.places.get(i).arc_id.get(k)!=0){
                  for(int j=0;j<initial_petrinet.arcs.size();j++){

      if(initial_petrinet.arcs.get(j).connection_id==initial_petrinet.
places
                                          .get(i).arc_id.get(k)){

      initial_petrinet.arcs.get(j).place=initial_petrinet.places.get(i
);

      initial_marking.arcs.get(j).place=initial_marking.places.get(i);
                                    break;
                              }
                        }
                  }
      }
      }
      for(int i=0;i<initial_petrinet.transitions.size();i++){
            for(int
k=0;k<initial_petrinet.transitions.get(i).arc_id.size();k++){
            if(initial_petrinet.transitions.get(i).arc_id.get(k)!=0){
                  for(int j=0;j<initial_petrinet.arcs.size();j++){

      if(initial_petrinet.arcs.get(j).connection_id==initial_petrinet.
transitions.
                                          get(i).arc_id.get(k)){

      initial_petrinet.arcs.get(j).transition=initial_petrinet.transit
ions.get(i);

      initial_marking.arcs.get(j).transition=initial_marking.transitio
ns.get(i);
                                    break;
                              }
                        }
                  }
      }
      }

}

public static void readFile(File file) throws IOException,
ParserConfigurationException, SAXException {
DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(file);
doc.getDocumentElement().normalize();
readTokens(doc);
readBonds(doc);
readNTokens(doc);
readNBonds(doc);
readPlaces(doc);
readTransitions(doc);
readArcs(doc);
fillArcs();
```

```
}

public static void mainF ()throws IOException,
ParserConfigurationException, SAXException {
 file = new File("C:\\Users\\Pantelina\\Downloads\\"
+ "ObeoDesigner-Community\\runtime-
New_configuration\\org.eclipse.sample.petrinets.sample\\My.petrinets")
;
readFile(file);

      }
}
```

## Reverse parser's code

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class ReverseParser {

public static void update_xml(PetriNet p, ArrayList<String[]> add,
ArrayList<String[]> remove)
throws ParserConfigurationException, SAXException, IOException,
TransformerException {
DocumentBuilderFactory docFactory =
DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
Document doc = docBuilder.parse(Parser.file);

ArrayList<String[]> check = new ArrayList<String[]>();
for (int j = 0; j < p.places.size(); j++) {
            String[] s = new String[2];
            s[1] = "";
```

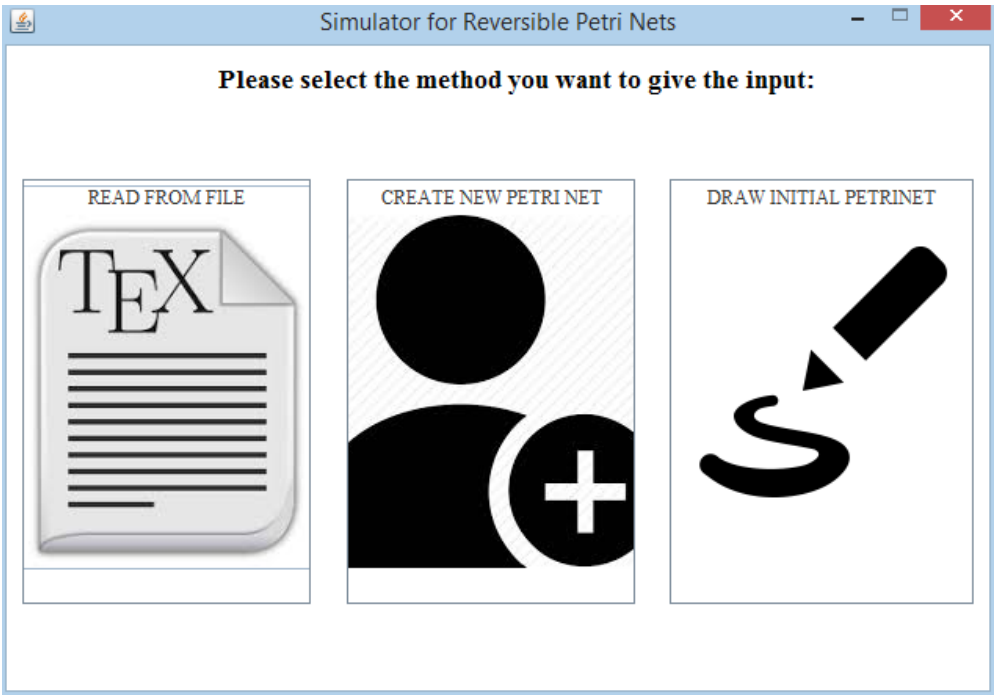
```
            s[0] = "";
for (int i = 0; i < add.size(); i++) {
      s[0] = p.places.get(j).place_name;
      if (p.places.get(j).place_name.compareTo(add.get(i)[0]) == 0) {
            s[1] += add.get(i)[1] + " ";

            }
      }
      if (s != null) {
            check.add(s);
      }
}

for (int i = 0; i < check.size(); i++) {

if (check.get(i)[1].compareTo("") != 0 && !check.get(i)[1]
      .contains("-")) {

Node place = doc.getElementsByTagName("place").item(i);
String split[] = check.get(i)[1].split(" ");
for (int j = 0; j < split.length; j++) {
      for (int k = 0; k < Parser.use.size(); k++) {
            if (split[j].compareTo(Parser.use.get(k)[1].toString()) ==
0) {
                  check.get(i)[0] = Parser.use.get(k)[0].toString();
            }
      }
}

Attr token = doc.createAttribute("token");
      String h = check.get(i)[0];
      token.setValue(h);
      ((Element) place).setAttributeNode(token);

} else if (check.get(i)[1].compareTo("") != 0 &&
check.get(i)[1].contains("-")) {
Node place = doc.getElementsByTagName("place").item(i);
String[] split = check.get(i)[1].split(" ");
String in = "";
int k = 0;
for (int j = 0; j < split.length; j++) {
      for (int h = 0; h < Parser.bonds.size(); h++) {
            if (split[j].compareTo(Parser.bonds.get(h).name) == 0) {
                  k = Parser.bonds.get(h).id + 200;
                  in += Integer.toString(k) + " ";
            }
      }

}
Attr token = doc.createAttribute("bond");
            in = in.substring(0, in.length() - 1);
            token.setValue(in);
            ((Element) place).setAttributeNode(token);
      }
}

for (int i = 0; i < remove.size(); i++) {
      for (int j = 0; j < p.places.size(); j++) {
      if (p.places.get(j).place_name.compareTo(remove.get(i)[0]) == 0)
      {
```

```
Node place1 = doc.getElementsByTagName("place")
            .item(p.places.get(j).place_id - 1);
NamedNodeMap nodes = place1.getAttributes();
for (int k = 0; k < nodes.getLength(); k++) {
      String id = nodes.item(k).getNodeName();
      if (id.equals("bond")) {
String r = "";
for (int y = 0; y < p.places.get(j).bonds.size(); y++) {
      int rep = p.places.get(j).bonds.get(y).id + 200;
      r += Integer.toString(rep);
}
if (r.compareTo("") != 0) {

            nodes.item(k).setNodeValue(
                        nodes.item(k).getNodeValue().
                        replace(nodes.item(k).getNodeValue(), r));
      } else {

            Element el = ((Attr) nodes.item(k)).getOwnerElement();
            el.removeAttribute(id);
      }

} else if (id.equals("token")) {
String r = "";
for (int y = 0; y < p.places.get(j).tokens.size(); y++) {
      int rep = p.places.get(j).tokens.get(y).id;
      r += Integer.toString(rep);
}

if (r.compareTo("") != 0) {
      nodes.item(k).setNodeValue(
      nodes.item(k).getNodeValue().replace(nodes.item(k)
                  .getNodeValue(), r));
} else {

Element el = ((Attr) nodes.item(k)).getOwnerElement();
el.removeAttribute(id);
                              }
                        }

                  }
            }
      }
}

TransformerFactory transformerFactory =
TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(new
File(Parser.file.getAbsolutePath()));
transformer.transform(source, result);

      }
}
```

# Appendix E

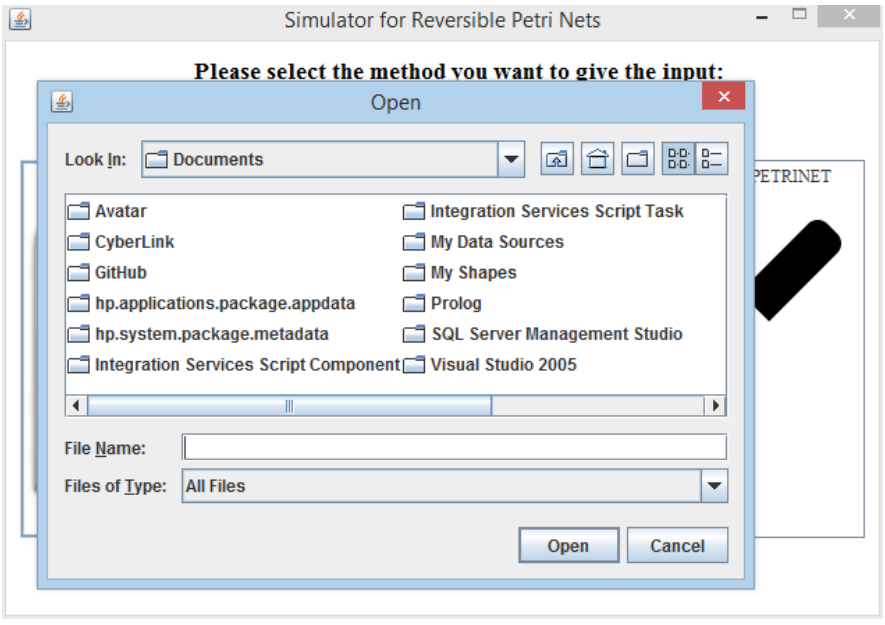
## Simulator manual

When the program starts the user has to choose by which method the input for the simulator is going to be imported. The program provides three methods by which the user can import the input.




If the user chooses the first option, then another window will appear, which will ask the user to give a text file as the input.

### First method – Import file:



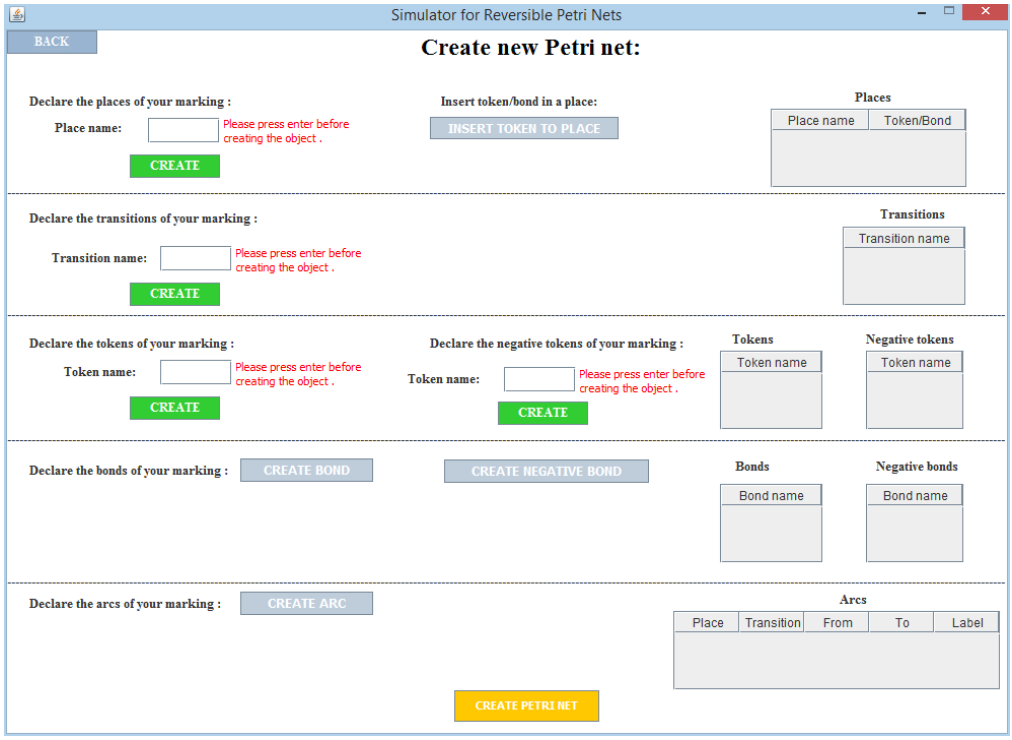
Since the user inserts the text file, and its type is correct, the program will continue to the next screen. If a file that is not at the correct form, a warning message will appear and then the user will have the opportunity to try again.


In order for the simulator to be able to read the file and create the appropriate Petri net structure, the text file has to look like the following example:

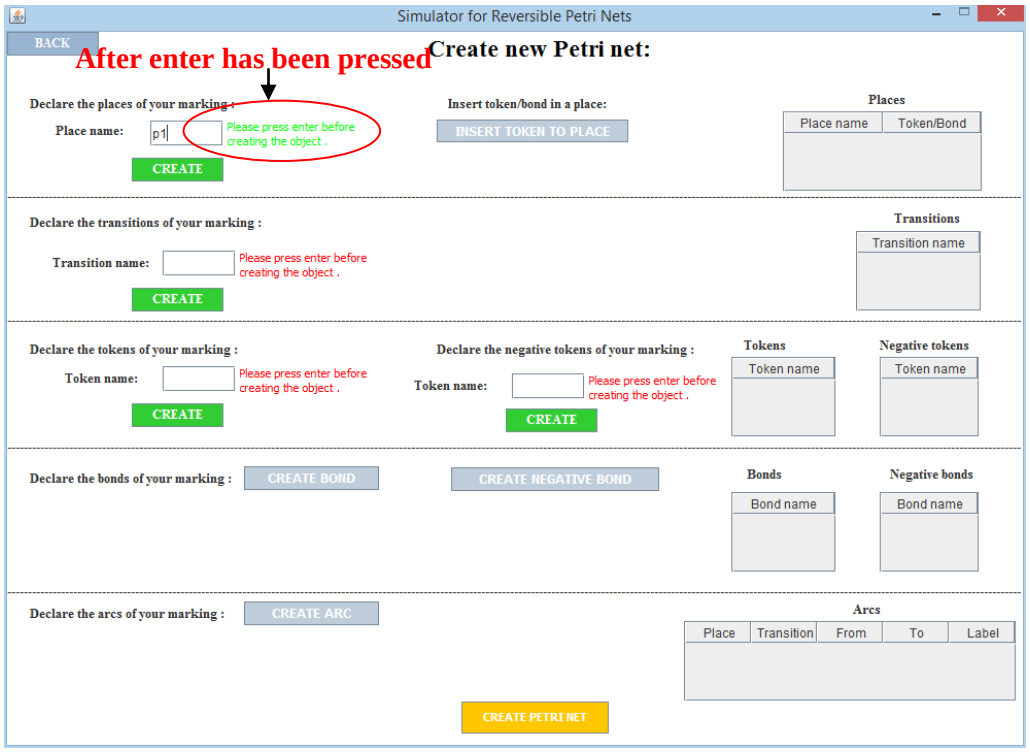
```
Tokens:
[a,b,c]
Places:
[p1,p2,p3,p4,p5]
Transitions:
[t1,t2]
Arcs:
[
(p1,t1)={a}
(p2,t1)={b}
(t1,p3)={a-b}
(p3,t2)={b}
(p4,t2)={c}
(t2,p5)={b-c}
]
Initial marking:
[
p1->a
p2->b
p3->0
p4->c
p5->0
]
```

## Second method- Create new Petri net:

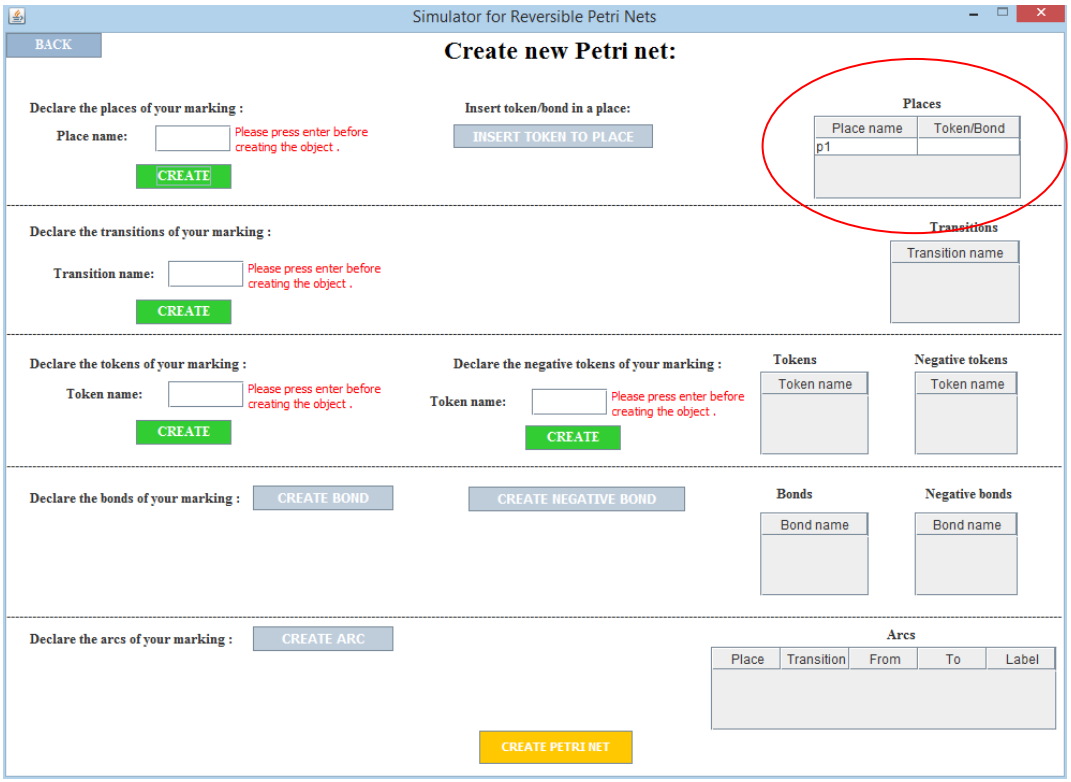
If the user wants to create a new Petri net structure through the program, instead of importing a file, the second choice must selected, namely "Create new Petri net". Since it has been selected, the next screen will appear.



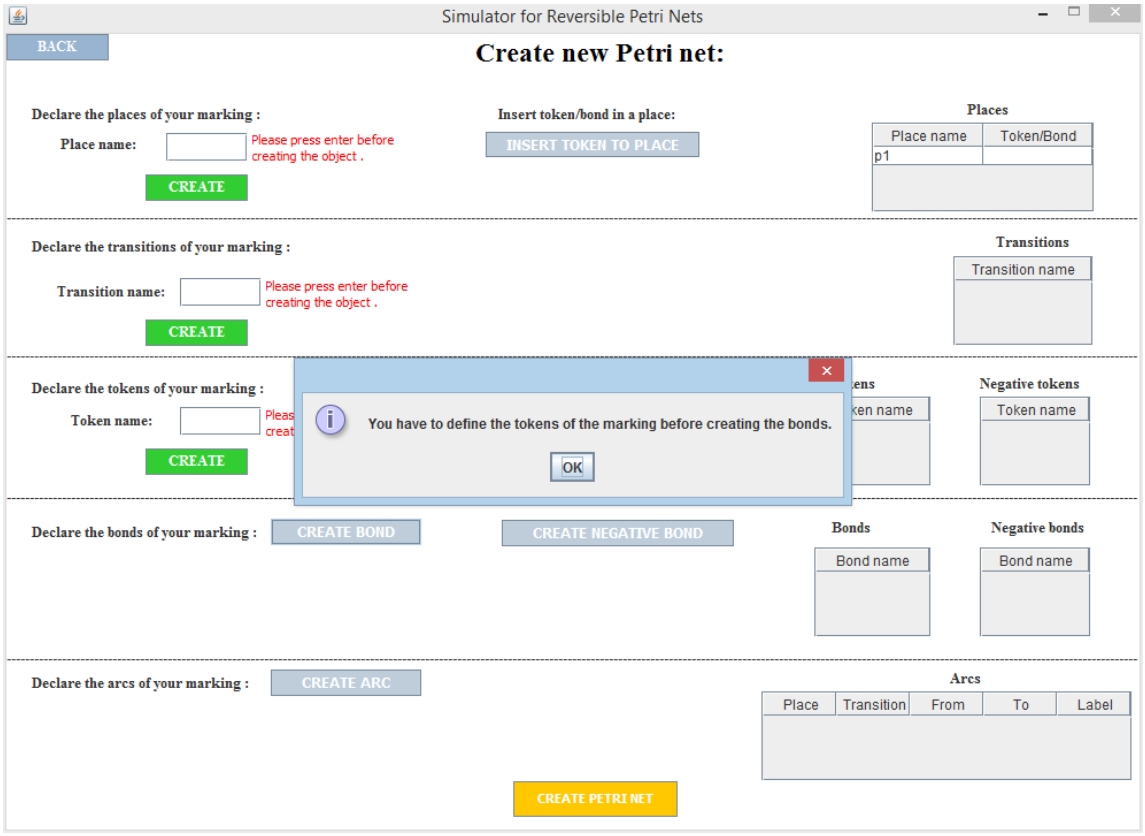

Through this window the user can create all the necessary elements of the initial marking. Firstly the user has to create places and transitions. When the user wants to create a component, after typing its name, enter key from keyboard has to be entered, before clicking "CREATE".

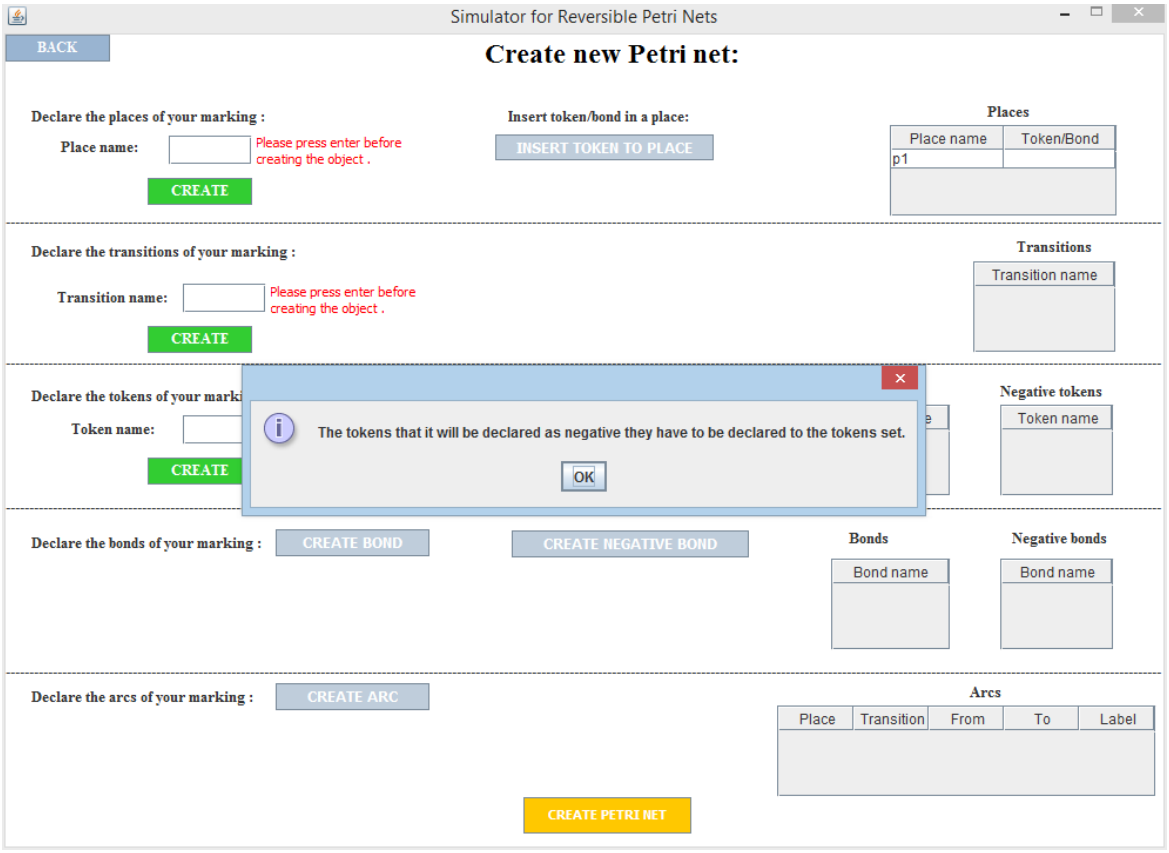


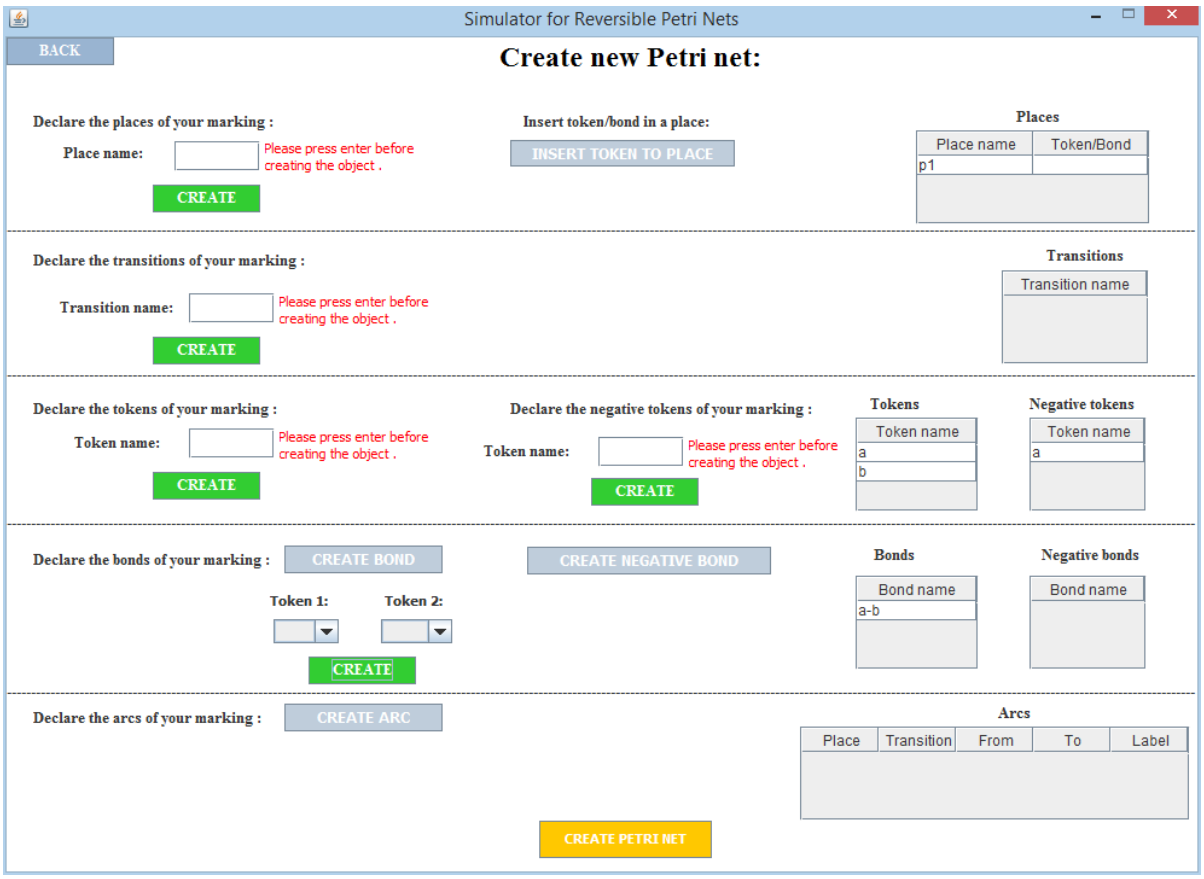
Since the user has pressed “enter” and then “CREATE” button, the element will appear at the table, on the right side of the window, accordingly.

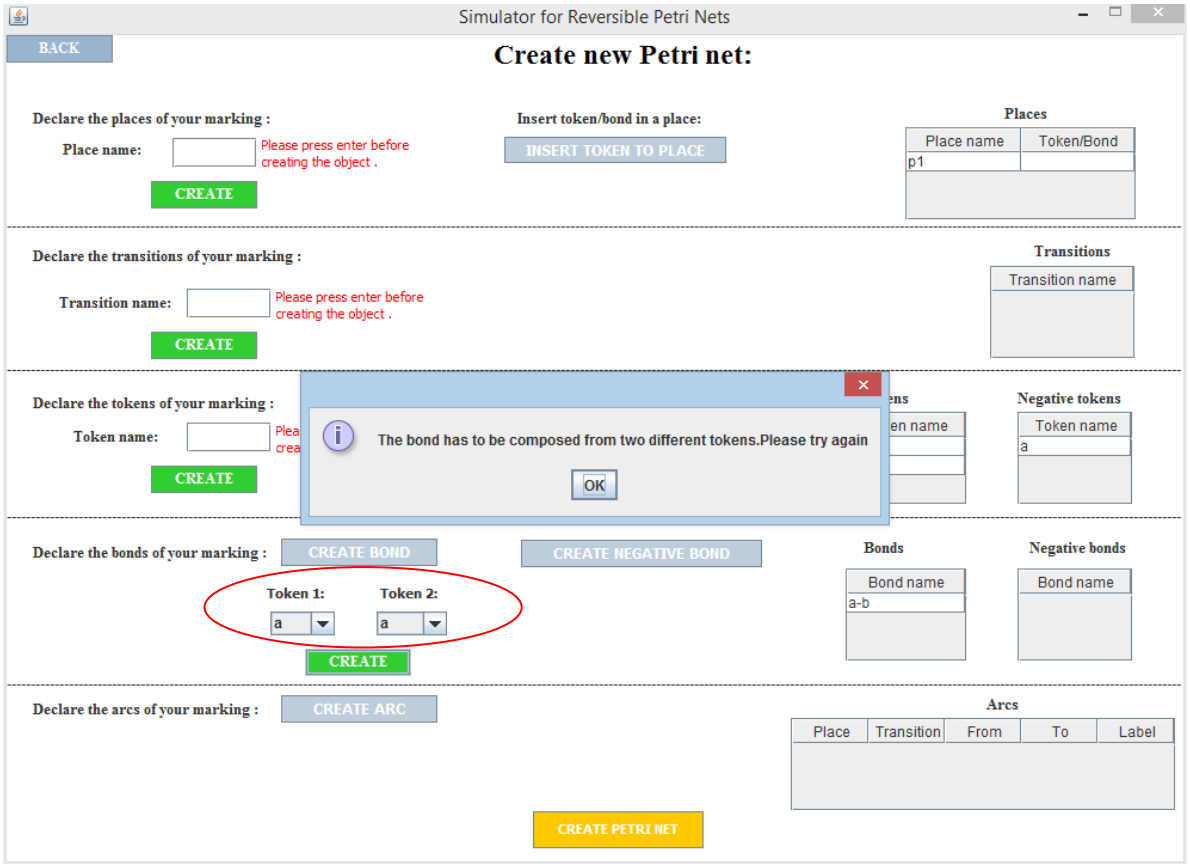



After the creation of places and transitions, the user has to define tokens and bonds appearing in the initial marking. Tokens must be defined first, and then bonds, so bonds will contain exactly two tokens. If the user tries to define a bond when no tokens have been defined, or a bond with two identical tokens, a warning message will appear. Negative tokens and negative bonds can be defined in the same way. A negative token has to refer to a defined token to be legitimate.

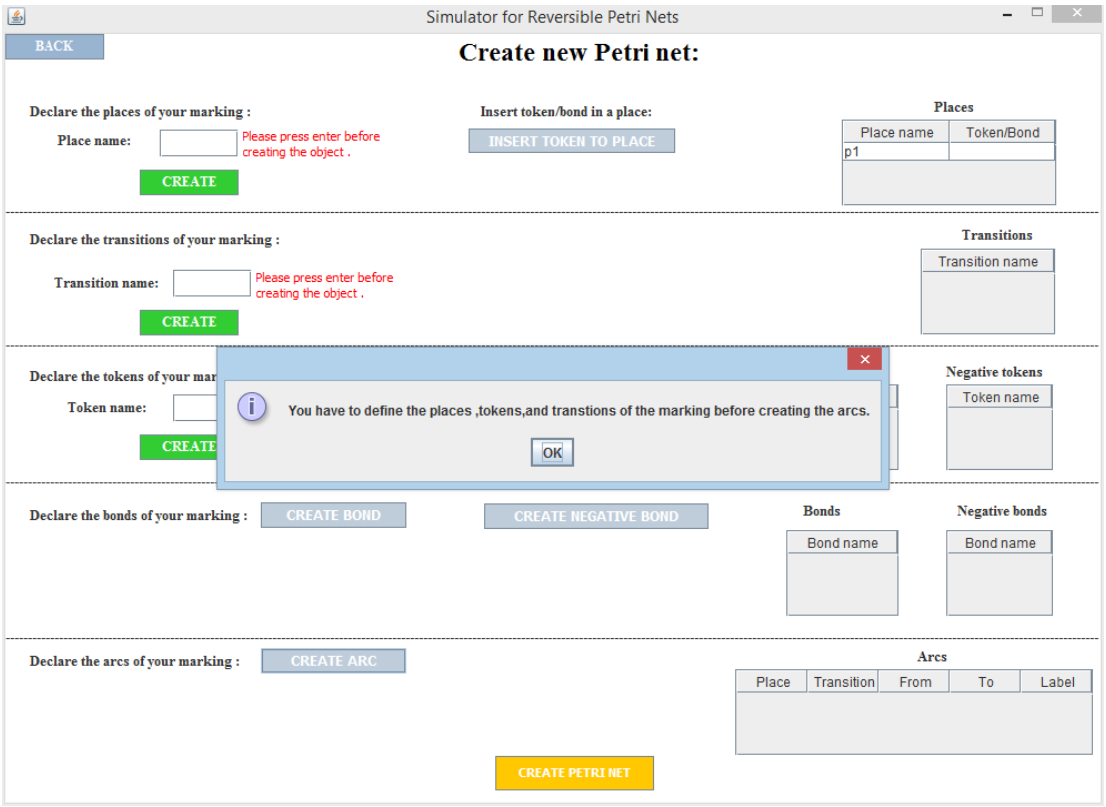

Simulator for Reversible Petri Nets
BACK
Create new Petri net:
Declare the places of your marking :
Place name:
Please press enter before creating the object .
CREATE
Insert token/bond in a place:
INSERT TOKEN TO PLACE
Places
Place name
Token/Bond
p1
Declare the transitions of your marking :
Transition name:
Please press enter before creating the object .
CREATE
Transitions
Transition name
Declare the tokens of your marking :
Token name:
CREATE
You have to define the tokens of the marking before creating the bonds.
OK
Negative tokens
Token name
Declare the bonds of your marking :
CREATE BOND
CREATE NEGATIVE BOND
Bonds
Bond name
Negative bonds
Bond name
Declare the arcs of your marking :
CREATE ARC
Arcs
Place
Transition
From
To
Label
CREATE PETRI NET

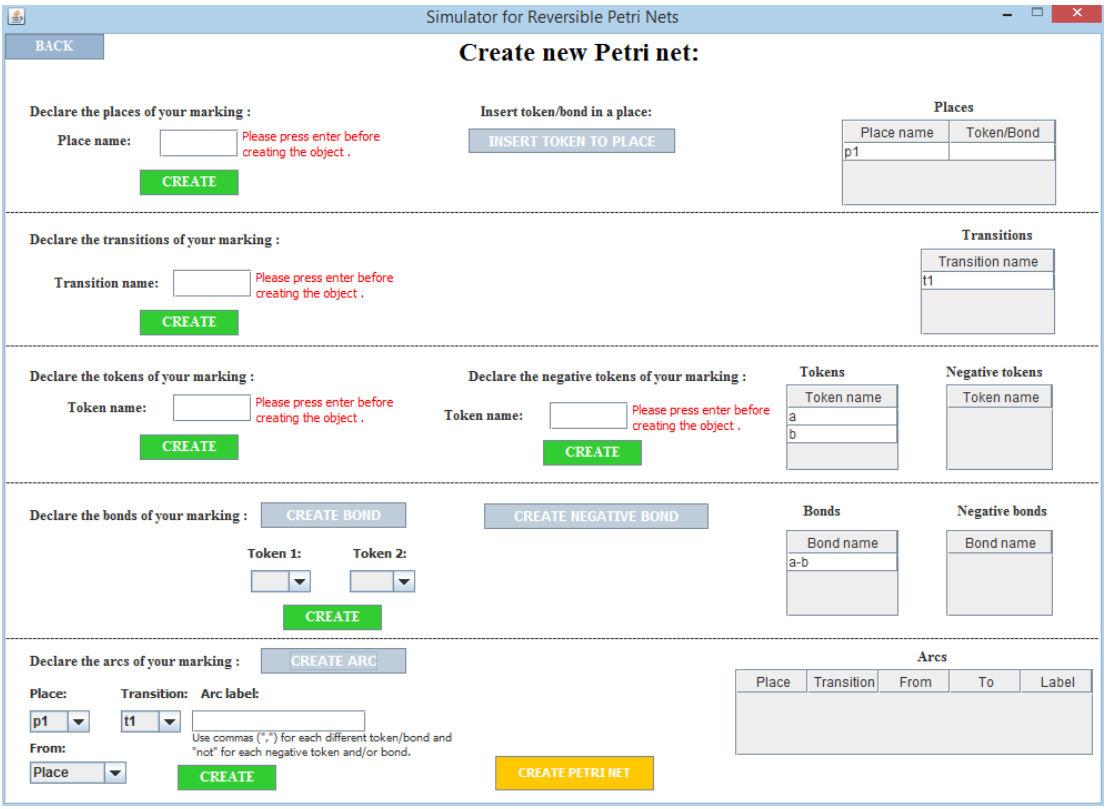

Simulator for Reversible Petri Nets
BACK
Create new Petri net:
Declare the places of your marking :
Place name:
Please press enter before creating the object .
CREATE
Insert token/bond in a place:
INSERT TOKEN TO PLACE
Places
Place name
Token/Bond
p1
Declare the transitions of your marking :
Transition name:
Please press enter before creating the object .
CREATE
Transitions
Transition name
Declare the tokens of your marki
Token name:
CREATE
The tokens that it will be declared as negative they have to be declared to the tokens set.
OK
Negative tokens
Token name
Declare the bonds of your marking :
CREATE BOND
CREATE NEGATIVE BOND
Bonds
Bond name
Negative bonds
Bond name
Declare the arcs of your marking :
CREATE ARC
Arcs
Place
Transition
From
To
Label
CREATE PETRI NET


Simulator for Reversible Petri Nets
BACK
Create new Petri net:
Declare the places of your marking :
Place name:
Please press enter before creating the object .
CREATE
Insert token/bond in a place:
INSERT TOKEN TO PLACE
Places
Place name
Token/Bond
p1
Declare the transitions of your marking :
Transition name:
Please press enter before creating the object .
CREATE
Transitions
Transition name
Declare the tokens of your marking :
Token name:
Please press enter before creating the object .
CREATE
Declare the negative tokens of your marking :
Token name:
Please press enter before creating the object .
CREATE
Tokens
Token name
a
b
Negative tokens
Token name
a
Declare the bonds of your marking :
CREATE BOND
CREATE NEGATIVE BOND
Token 1:
Token 2:
CREATE
Bonds
Bond name
a-b
Negative bonds
Bond name
Declare the arcs of your marking :
CREATE ARC
Arcs
Place
Transition
From
To
Label
CREATE PETRI NET

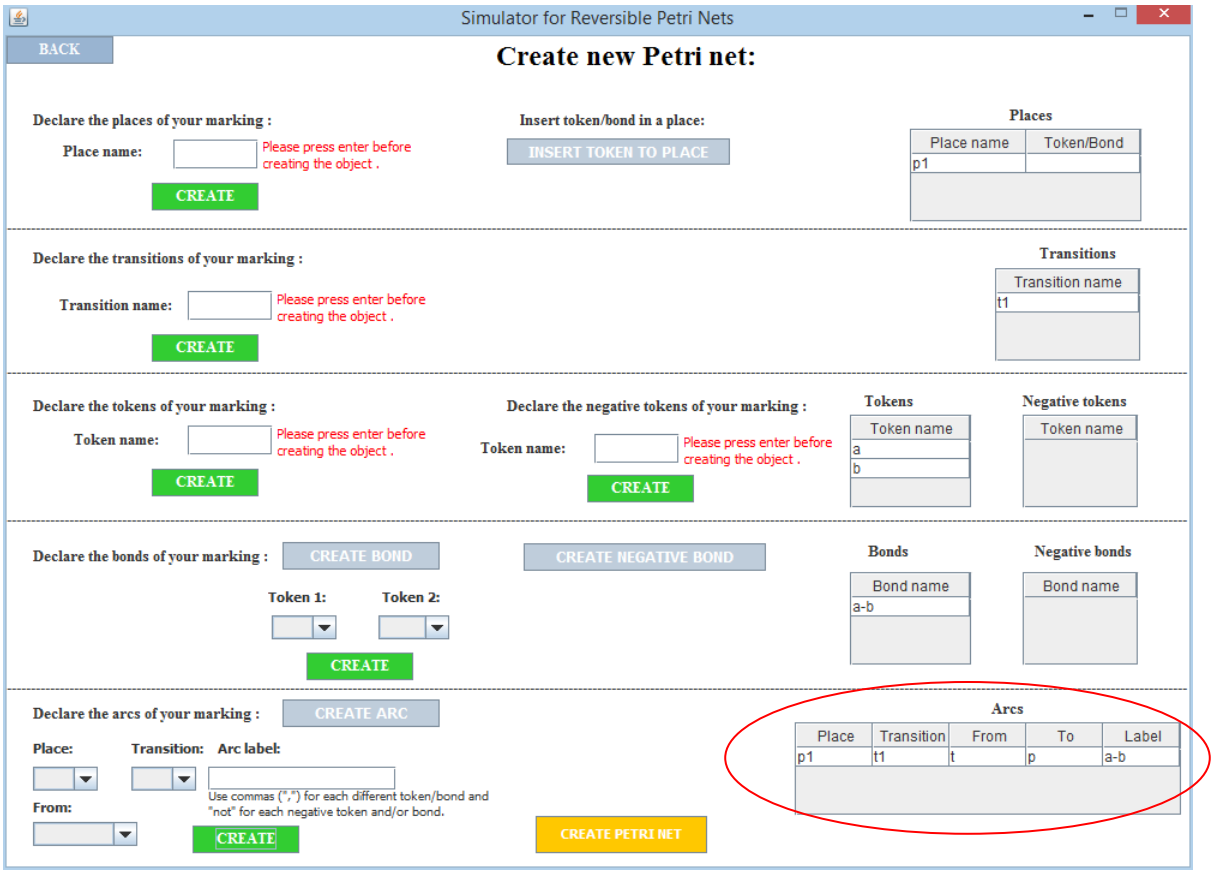

Simulator for Reversible Petri Nets
BACK
Create new Petri net:
Declare the places of your marking :
Place name:
Please press enter before creating the object .
CREATE
Insert token/bond in a place:
INSERT TOKEN TO PLACE
Places
Place name
Token/Bond
p1
Declare the transitions of your marking :
Transition name:
Please press enter before creating the object .
CREATE
Transitions
Transition name
Declare the tokens of your marking :
Token name:
CREATE
The bond has to be composed from two different tokens.Please try again
OK
Negative tokens
Token name
a
Declare the bonds of your marking :
CREATE BOND
CREATE NEGATIVE BOND
Token 1:
Token 2:
a
a
CREATE
Bonds
Bond name
a-b
Negative bonds
Bond name
Declare the arcs of your marking :
CREATE ARC
Arcs
Place
Transition
From
To
Label
CREATE PETRI NET

Since the desired tokens, bonds, negative tokens, and negative bonds have been created, the user can create arcs for the initial marking. In order to create arcs, the users has to ensure that both places and transitions needed have been defined. Morever, all tokens, bonds, negative tokens and/or negative bonds needed have been declared too.



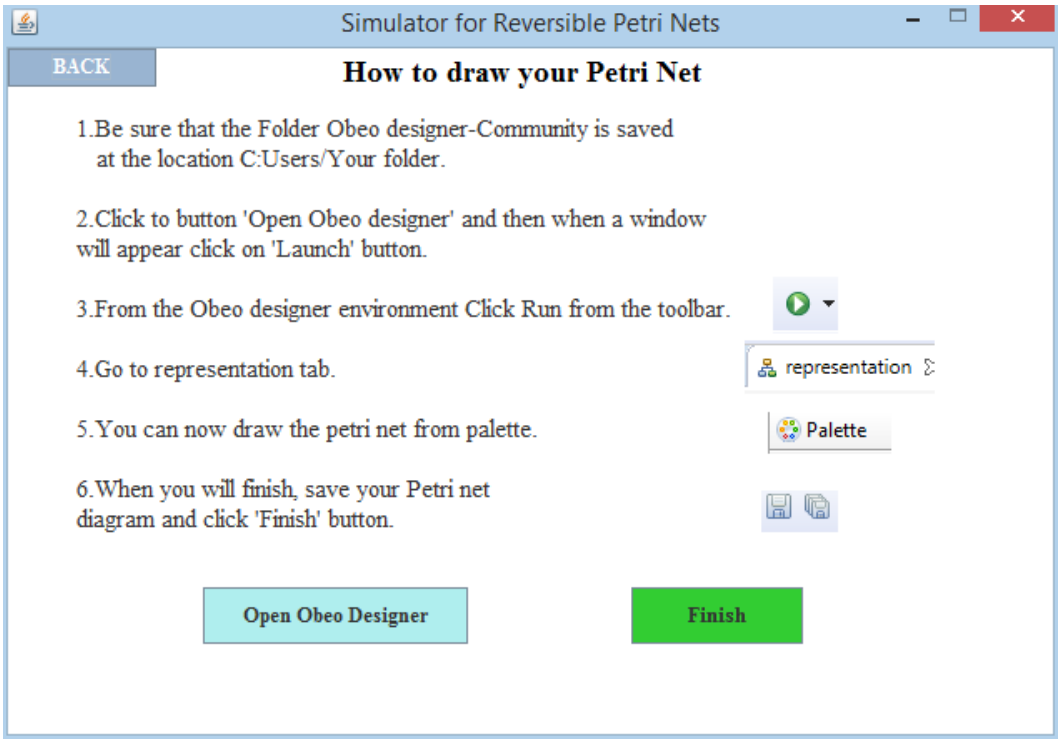
The user in order to create an arc, has to choose the place and the transition which this arc connects, if the arc is from place or from transition, and define the label of this arc (pre-condition or post-condition.



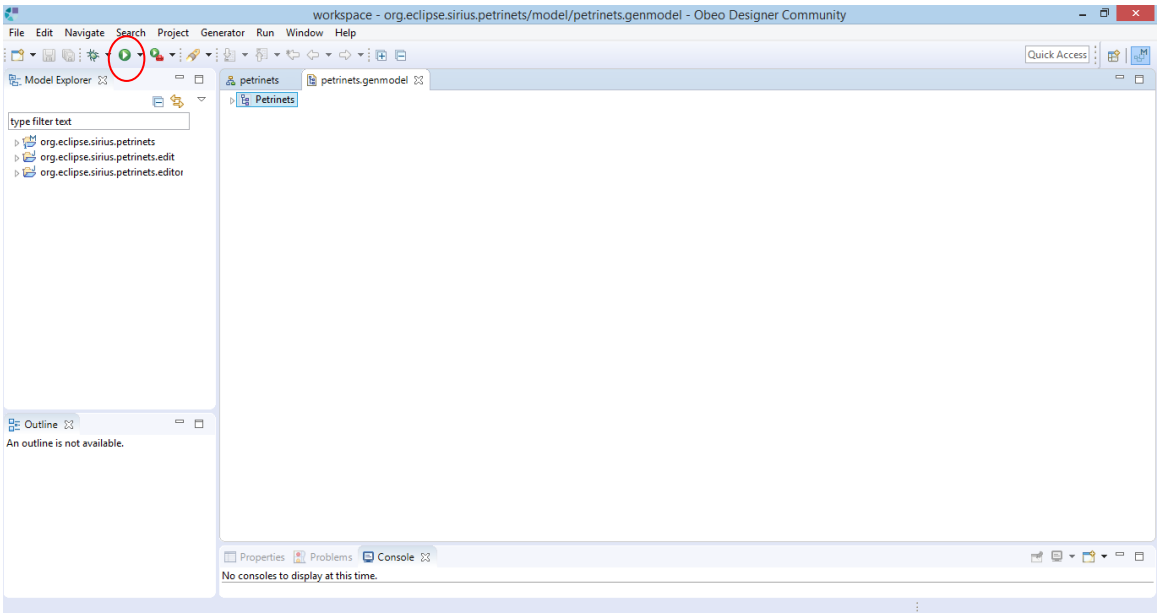

When the user has defined all the necessary components of the initial marking, he/she can click on “CREATE PETRI NET” button to complete the process.

## Third method – Draw a Petri net diagram

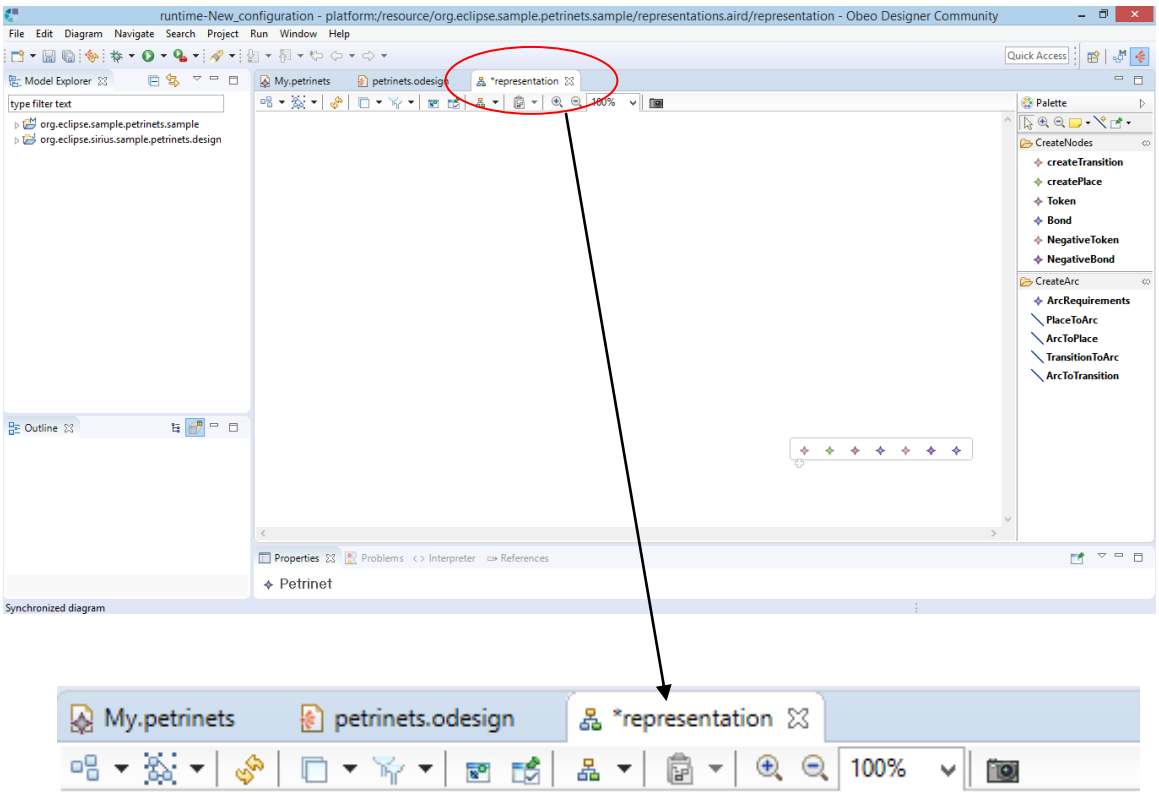
The last method to insert an input to the simulator is by draw the Petri net diagram visually. The first step of the process is to select “Draw initial Petri net” option from the initial screen. Then an explanation window will appear.




The user has to follow the instructions appeared in the window, to be able to draw the initial marking. The diagram will be created, through another software, Obeo Designer, so there are some steps to go there. Firstly, the user has to ensure that the Obeo Designer's folder, which has been downloaded with the simulator, is at the inidicated location. Therefore, the user can select "Open Obeo Designer" to launch the software. Now, the user must see the following screen:



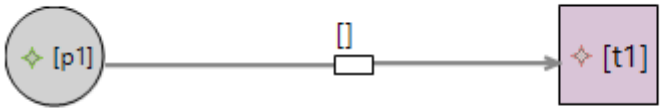

There, the user has to click on Run button, without closing the current window. Afterwards, the user is going to redirected in another window. On the next screen the user has to go at the representation task, where she/he will be able to draw the initial marking, through the palette.



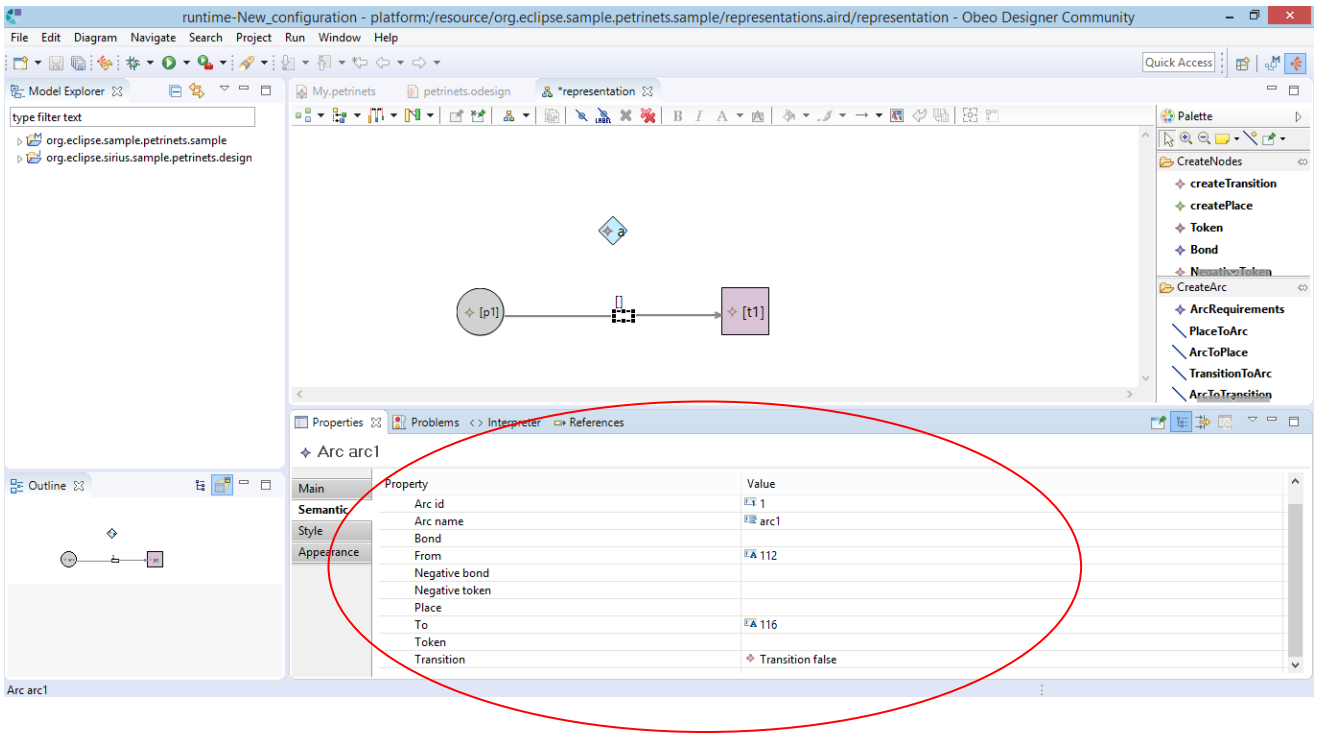




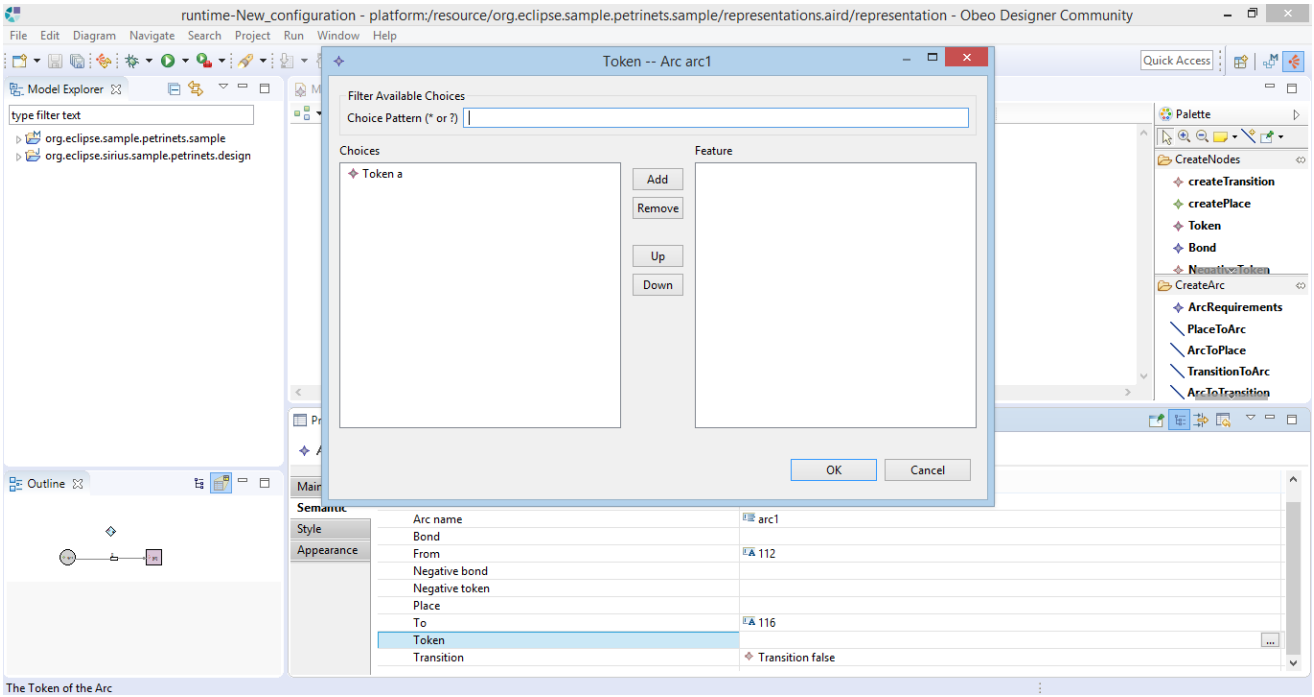
Now, the user can define the initial marking, by creating all the necessary elements. For the creation of an arc the user has to create the arc object at first, and then define the necessary edges from and to the corresponding place/transition.

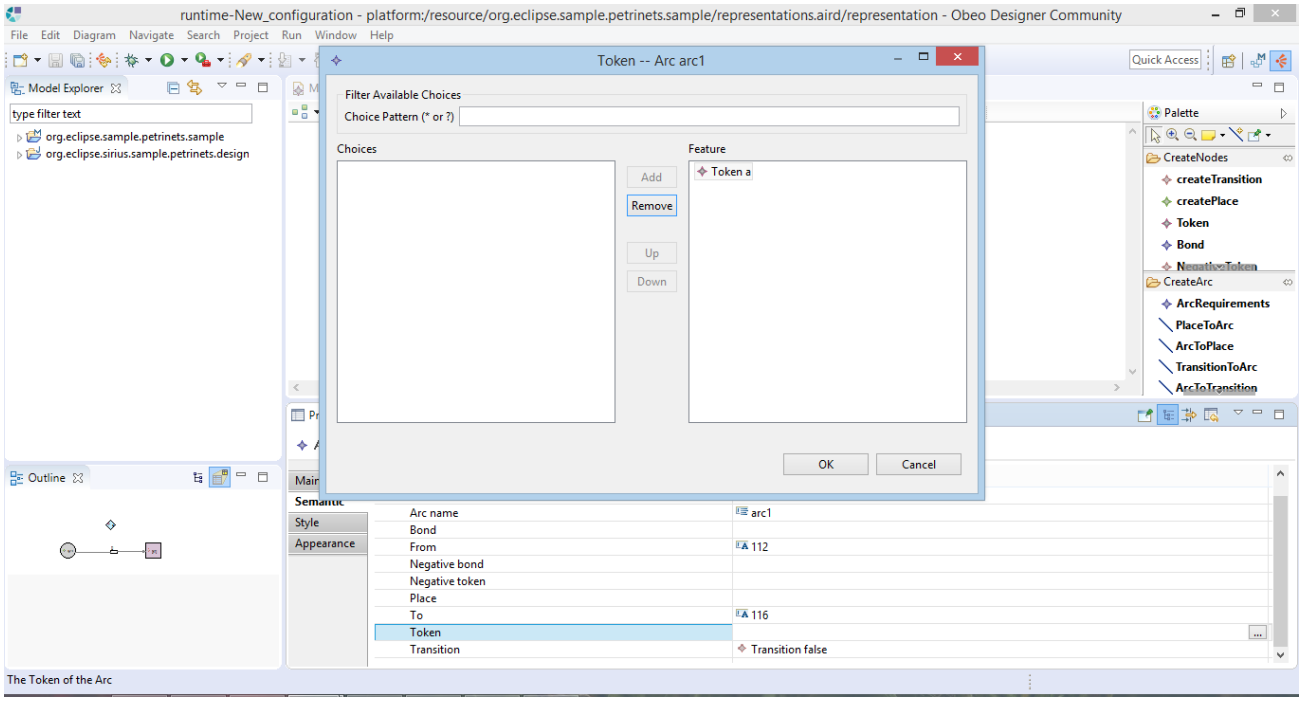


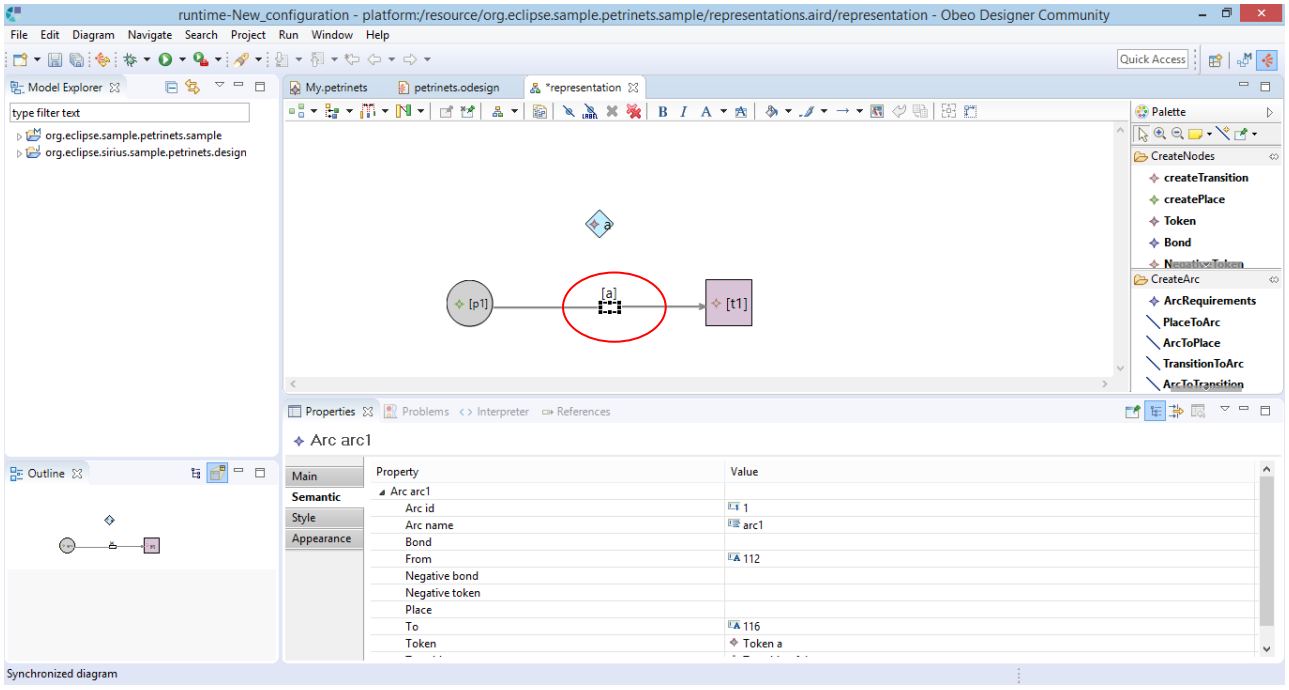

The braces above the arc object indicate the arc label (pre-condition/post-condition). So, in order to insert tokens/bonds into places or tokens/bonds/negative tokens/negative bonds to the arc label, the user has to find the sematics area of the corresponding element.



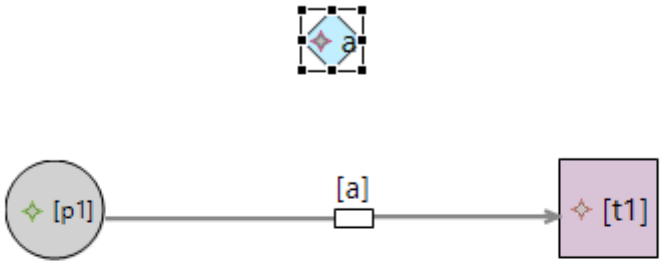

At the semantic area, the user can define possible tokens/bonds into places or tokens/bonds/negative tokens/negative bonds to the arc label, by just press on "…" option at the right side of the desired object, and then all options will appear.

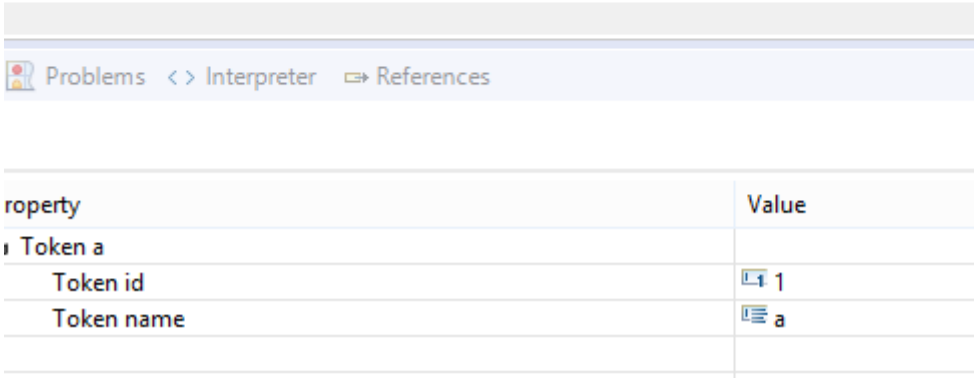



runtime-New_configuration - platform:/resource/org.eclipse.sample.petrinets.sample/representations.aird/representation - Obeo Designer Community
File Edit Diagram Navigate Search Project Run Window Help
Token -- Arc arc1
Filter Available Choices
Choice Pattern (* or ?)
Choices
Feature
Token a
Add
Remove
Up
Down
OK
Cancel
Model Explorer
type filter text
org.eclipse.sample.petrinets.sample
org.eclipse.sirius.sample.petrinets.design
Palette
CreateNodes
createTransition
createPlace
Token
Bond
CreateArc
ArcRequirements
PlaceToArc
ArcToPlace
TransitionToArc
ArcToTransition
Outline
Style
Appearance
Arc name
arc1
Bond
From
112
Negative bond
Negative token
Place
To
116
Token
Transition
Transition false
The Token of the Arc

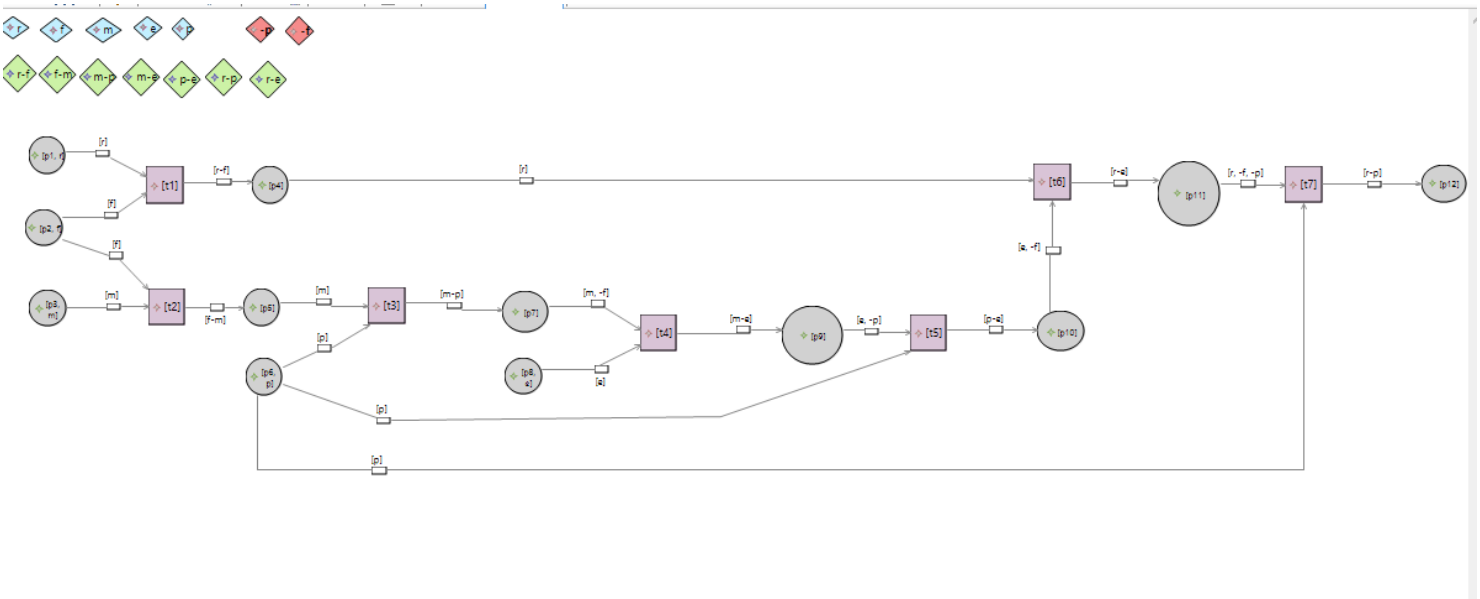

runtime-New_configuration - platform:/resource/org.eclipse.sample.petrinets.sample/representations.aird/representation - Obeo Designer Community
File Edit Diagram Navigate Search Project Run Window Help
My.petrinets
petrinets.odesign
*representation
Model Explorer
type filter text
org.eclipse.sample.petrinets.sample
org.eclipse.sirius.sample.petrinets.design
Palette
CreateNodes
createTransition
createPlace
Token
Bond
CreateArc
ArcRequirements
PlaceToArc
ArcToPlace
TransitionToArc
ArcToTransition
[p1]
[a]
[t1]
Properties
Problems
Interpreter
References
Arc arc1
Outline
Main
Semantic
Style
Appearance
Property
Value
Arc arc1
Arc id
1
Arc name
arc1
Bond
From
112
Negative bond
Negative token
Place
To
116
Token
Token a
Synchronized diagram

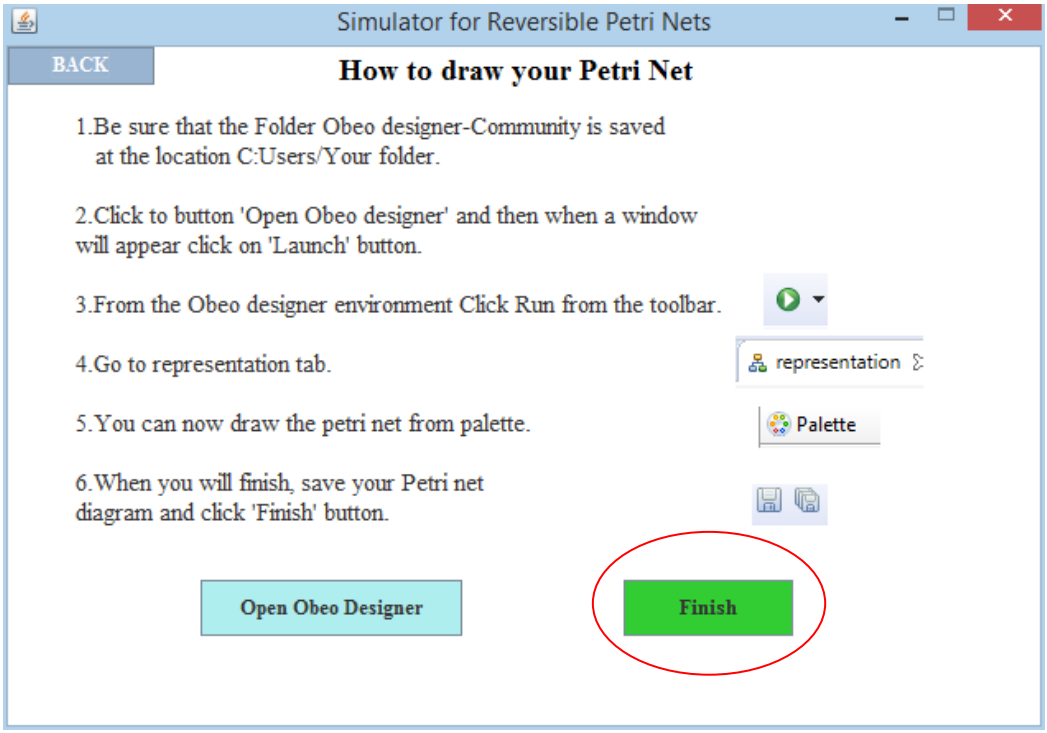
From the semantics area the user can also define token/bond/negative token/negative bond's name respectively.



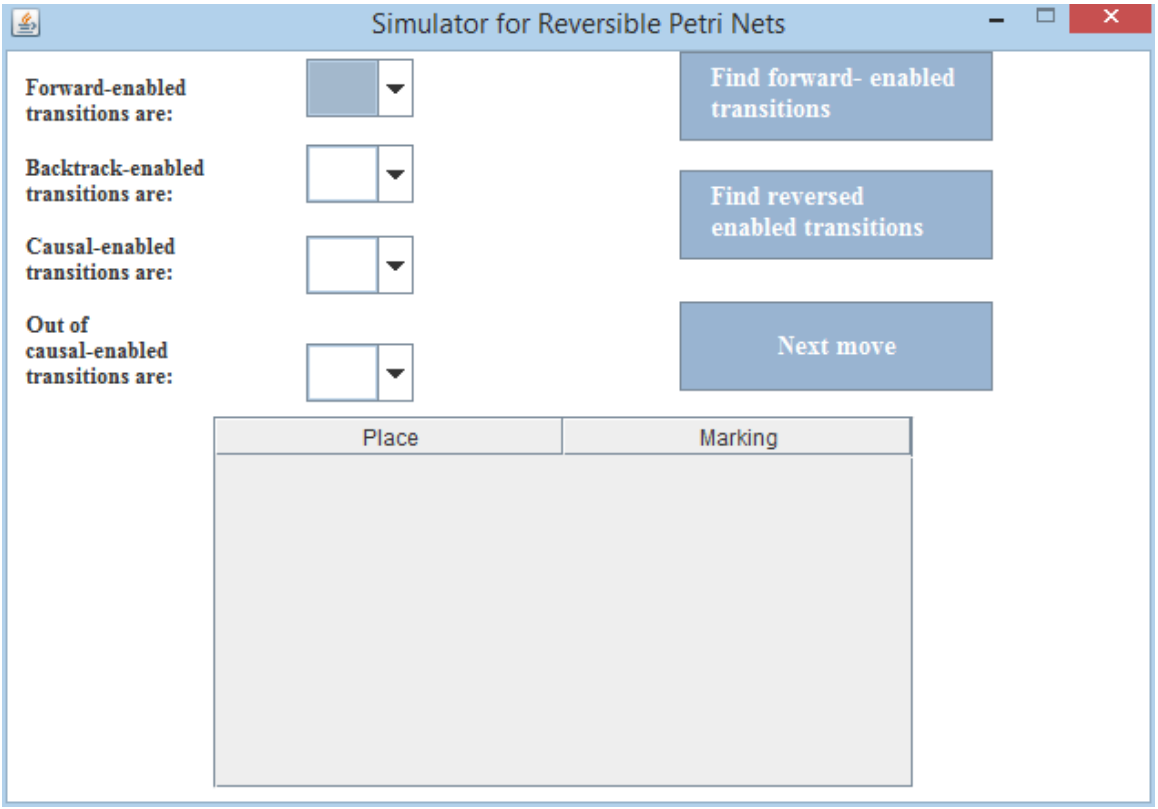

When the initial marking is ready, the user has to save it and then return to the simulator. There he/she must click "Finish" button.



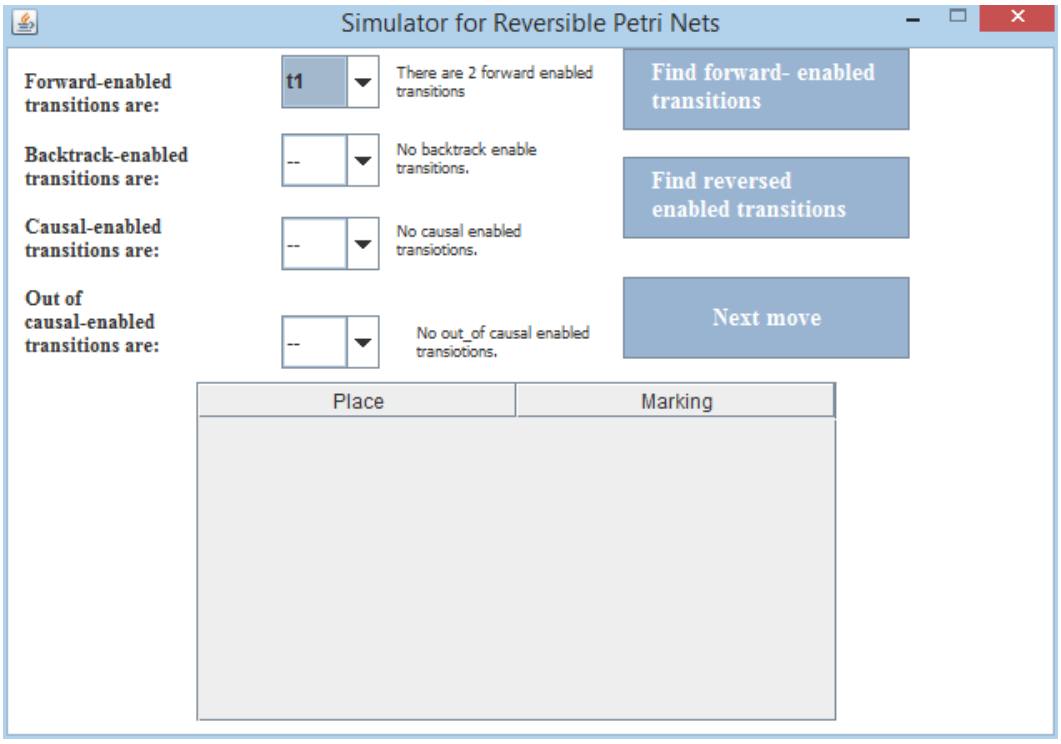



## Execution of transitions:

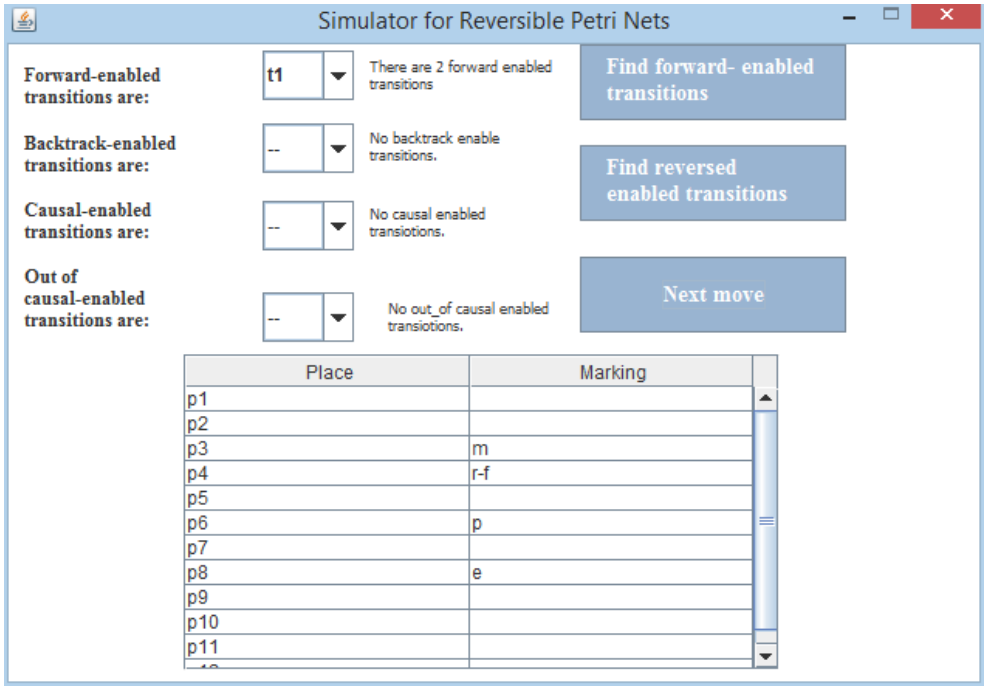
Since the input has been imported the user is going to redirected to the next screen, where the user can execute transtions, either in forward or in reverse order.



The user can select to search for the forwars-enabled and/or the reversed enabled transitions, according to the given marking. Then the user can choose a transition and a form of execution (forward, backtrack, causal, out-of-causal), one at a time to be executed.




Then the user can press “Next move” button to execute the transition. After the execution the new marking will appear to the table below.



## Additional feature for third Method:

If the user has chosen the third method to insert the input, another feature is available. The user can observe the changes happening on the marking, on the diagram too, from the graphical representation tool. In order to achieve this the user has to keep open the Obeo Designer's windows open, and then after each execution update the source file of the diagram. To update the source file, the user has to just click on the file "My.petrinet", which is at the toolbar located in the left side of the window. Afterwards, the changes will be available on the screen.