Democratic and Popular Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Mohamed Khider - BISKRA
Faculty of Exact Sciences, Natural Sciences and Life
Computer Science Department

Order Number: GLSD2/M2/2019

Thesis

Presented to obtain the diploma of academic Master in

Computer Science

Option: **Software Engineering and Distributed Systems**

A Tool for Modeling and Simulating High Level Petri Nets

By:
Merabti Samah

Defended the 07/07/2019, in front of the jury composed of:

Djaber Khaled	MAA	President
Kahloul Laid	MCA	Supervisor
Bendahmane Asma	MAA	Examiner

University Year: 2018/2019

Acknowledgements

Praise to ALLAH, the Compassionate, the Merciful. Peace and blessing on the Messenger of Allah, Muhammad the prophet (Peace Be Upon Him). I wish to express my gratitude to ALLAH for His blessing and inspiration leading me to finish this work.

My special thanks and appreciation to my supervisor **Dr. Kahloul Laid** for his continuous encouragement, guidance and for his endless patience and precious advice.

I would like to express my deepest thanks to the members of the jury for reading and evaluating my dissertation.

I never forget to thank all my teachers And my colleagues at Computer Science Department especially amina and saouab .

Finally, my gratitude is deeply paid to my mother and my father, my sister and my brother, and specially to my late dear friend "imane" and to all my friends and members of my family.

Abstract

The Reconfigurable Object Petri Nets (RONs) are formal models of High-Level modeling for dynamic systems. In recent years, the model has been proposed and used for modeling several reconfigurable systems ; however, it suffers like most high-level formalism from the lack of implementation. There have been some attempts to provide tools to bridge this gap. Nevertheless, the proposed tools are not effective, as they often contain bugs or simply unavailable for developers. The objective of this project is to provide a tool to model, to simulate and to verify the RON formalism.

Résumé

Les RdPs objets reconfigurables sont des modèles formels de haut niveau pour la modélisation des systèmes dynamiques. Ce modèle proposé depuis quelques années a été utilisé pour la modélisation de plusieurs systèmes reconfigurables, cependant il souffre comme la majorité des formalismes de haut niveau du manque au niveau des implémentations. Il existe quelques tentatives pour offrir des outils pour combler à ce manque, cependant les outils proposés ne sont pas efficaces, contiennent souvent des bugs ou tout simplement indisponibles aux développeurs. L'objectif de ce projet est de réaliser un outil pour modéliser, simuler et vérifier le formalisme RONs.

Contents

List of Tables

List of Figures

Introduction

Computer, science is growing up since the last 50 years and one of it's major branches is the verification [K.J00]. The most objective of verification is to help the development of systems reliably, to carry out verification, we first realize a model of the system that is a mathematical presentation in which we reason, verify and check properties which interest us. Formal modeling of a system can be used as a basis for simulating, analyzing and ensure it's good functioning. There is different systems to model like reconfigurable systems which are systems with dynamic and open structure that can be changed or reconfigured instead of being replaced. Such a model is realized with a formalism of modeling like Petri nets. There are two classes of Petri nets :low level and high level Petri Nets. RONs is an extension of high level Petri net which can model and specify reconfigurable systems.

There are tools that automate modeling and simulation of RONs but these are confronted with bugs and shortcomings like lack of analysis. The goal of this work to realize a tool for modeling, simulation and analysis of RONs.

This thesis starts with an introduction that presents the problem, then it is composed of two parts, the first part focuses on the theoretic level(State of the art), the second part shows our contribution in this work.
Part I is divided into two chapters:

chapter 1 Review some basic definitions describing Petri nets, their analysis and the properties that can be modeled with this approach. Finally, it is realized that the low level PNs are not sufficiently expressive and that a high-level extension will be needed to model the systems that can be reconfigured.
chapter 2 In this chapter we will introduce an extension of high level PNs that allows the modeling of transformation system and gives the basic concepts of Reconfigurable Object Nets, and the introduce the steps followed to the appearance of this formalism.

Part II This part concentrates on the development of our tool for modeling, simulation and analysis of High Level Petri Nets, it contains two chapters.

chapter 3 Illustrates the design of our application in the two levels (the global and the detailed design)

chapter 4 This chapter is the last chapter. it introduces the tools and techniques used to develop this project and the tool realized by specifying its interface and its functionalities.

The thesis ends with a general conclusion that evaluates the result, and discuss some perspectives of this work.

Part I

State of the art

Chapter 1:

Low Level Petri Nets

Chapter 1

Low Level Petri Nets

Introduction

Petri nets are a graphical and mathematical modeling tool applicable to many systems [Mur89]. There are two classes of Petri nets: Low and High level Petri nets. Low level Petri nets are used to represent systems in a simple way. They are called Place/Transition nets (P/T nets) [Wol08]. P/T nets are a mathematical modeling language for the description of distributed systems. In this chapter we will give a formal and informal definition of P/T net, it's execution semantics, the analysis of the properties of petri net and what can we model with P/T nets.

1.1 Informal definition

A Petri net is a directed bipartite graph, in which the nodes represent places(i.e., conditions, represented by circles) and transitions (i.e., events that may occur, represented by bars). The directed arcs describe which places are pre- and/or post conditions for which transitions (modeled by arrows).

1.2 Formal definition

The following is the formal definition of a Petri net [Mur89] [Rei85] [CL09].

Definition 1. A Petri net is a five-tuple:(P,T,A,W,M_0) where:
•P is a finite set of places
•T is a finite set of transitions
•$A \subseteq$ (P $\times$ T) $\cup$ (T $\times$ P): is a set of arcs
•$W : A \to \{1, 2, 3, \dots\}$: is a weight function
•$M_0 : P \to z^+$:is the initial marking

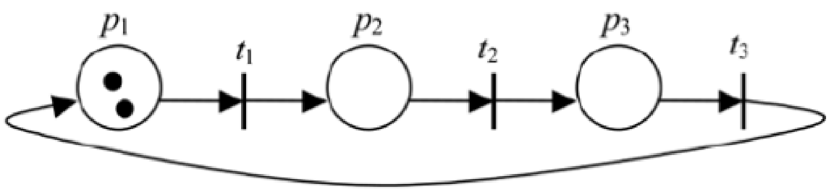


Figure 1.1: Petri Net example

Definition 2. (Marking) The marking M_i of a place $P_i \in P$ is a non-negative quantity representing the number of tokens in the place P_i at a given state of the Petri net. The marking of the Petri net is defined as the function:

$$M : P \rightarrow z^+$$

that maps the set of places to the set of non-negative integers. It is also defined as a vector:

$$M_j = (m_1, m_2, \ldots, m_n)$$

where: $m_i = M_j(p_i)$ which represents the j^{th} state of the net. M_j contains the marking of all the places and the initial marking is denoted by M_0.

Definition 3. (Incidence matrices)

Pre incidence matrix:

$$Pre(p,t) = \left\{ \begin{array}{cc} w(p,t), & \text{if p} \in ^{\circ}\text{t} \\ 0, & \text{otherwise} \end{array} \right\}$$

Post incidence matrix:

$$Post(p,t) = \left\{ \begin{array}{cc} w(t,p), & \text{if p} \in t^{\circ} \\ 0, & \text{otherwise} \end{array} \right\}$$

Incidence matrix

$$C = Post - Pre$$

1.3 Execution semantics

1.3.1 Enabled transition

A transition t is enabled (it may fire) in M if there are enough tokens in its input places $^{\circ}t$ for the consumptions to be possible, it means that the number of tokens in input places of transition t are greater or equal t° the weight of output arcs of places $^{\circ}t$.

Definition 4. Formally a transition t is enabled in M if:

$$\forall p \in^{\circ} t : M(p) \geq W(p,t)$$

where :
-$M(p)$ is the marking of place p
-$W(p,t)$:the weight of arc between t and its input place p

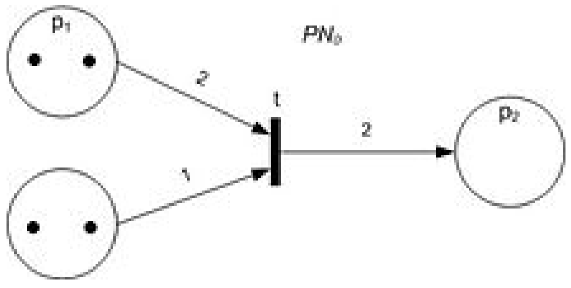


Figure 1.2: A Petri Net with an enabled Transition

1.3.2 Firing a transition

•Firing a transition t in a marking M consumes $W(p,t)$ tokens from each of its input places p, and produces $W(t,p)$ tokens in each of its output places s.
-removing $w(p,t)$ tokens from each $p \in^{\circ} t$
-adding $w(t,p)$ tokens to each $p \in t^{\circ}$ where:
t°: is the set of output places of t.

•Firing a transition leads to a new marking that enables another transitions.

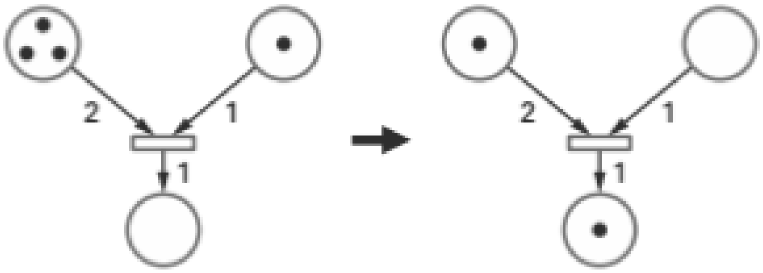


Figure 1.3: Firing a Transition

Definition 5. Firing a transition t from a marking M to M' is formalized by:

$$\forall (p) M'(p) = M(p) - pre(p,t) + post(p,t)$$

1.4 Analysis

One thing makes Petri net interesting is that they provide a balance between modeling power and analyzability. The analysis of Petri nets consists in checking some properties on the model. There are two methods of analysis: static analysis and dynamic analysis.

1.4.1 Mathematical properties of Petri net

Two types of properties are generally distinguished: the behavioral properties, related to a Petri net marked initially, and structural properties related to a Petri net independently of any initial marking [Mor02].

1.4.1.1 Examples of properties

Liveness: A transition t is said live if it can always be made enabled starting from any reachable marking, i.e.,$\forall M \in R(M_0)$, ∃ M' $\in R(M)$ such that $M'(t >$

A Petri net is said live if all transitions are live.

Reversibility: A Petri net is said reversible if the initial marking remains reachable from any reachable marking, i.e., $M_0 \in R(M), \forall M \in R(M0)$

Deadlock-free: A marking M is said a deadlock or dead marking if no transition is enabled at M.
A Petri net is said deadlock-free if it does not contain any deadlock.

Boundness: A Petri net P is k-bounded if there exists a positive integer k, such that for every marking M and for every place p of P:

$$M(p) < k$$

Safety: A Petri net is safe if it is 1-bounded. [VAL78]

1.4.2 Dynamic analysis methods

The dynamic analysis of petri net may classified into the following: coverability(reachability) tree/graph method.

Definition 6. (Reachability tree:) The reachability tree, also called marking graph, of a Petri net (N, M0) is a graph in which nodes corresponds to reachable markings from an initial marking, arcs correpond to feasible transitions.
We use the Algorithm 1 below to draw reachability tree:

Algorithm 1 Reachability Tree

```
Algorithm ReachabilityTree  Root← $M_0$ while $\exists t \in T : M_0 \xrightarrow{t} M$ do
    Calculate $M : M_0 \xrightarrow{t} M$
    Add M to tree nodes
    Add ($M_0 \xrightarrow{t} M$ to tree arcs)
    ReachabilityTree($M_0$);
end
```

Definition7. (Reachability graph): If the reachability tree is infinite but the accessible marking finite(loop case), it is necessary to draw reachability graph to hqve a finite representation. We use the Algorithm 2 below to draw reachability graph:

Algorithm 2 Reachability Graph

```
Algorithm ReachabilityGraph
Root← $M_0$
while $\exists t \in T : M_0 \xrightarrow{t} M$ do
    if $M \notin$ to the set of nodes then
        Add $M$ to the set of nodes
        Add ($M \xrightarrow{t} M$) to graph's arcs
        ReachabilityGraph($M$);

    else
        Add ($M \xrightarrow{t} M$) to graph's arcs
    end
end
```

If this graph is calculable all properties becomes decidable

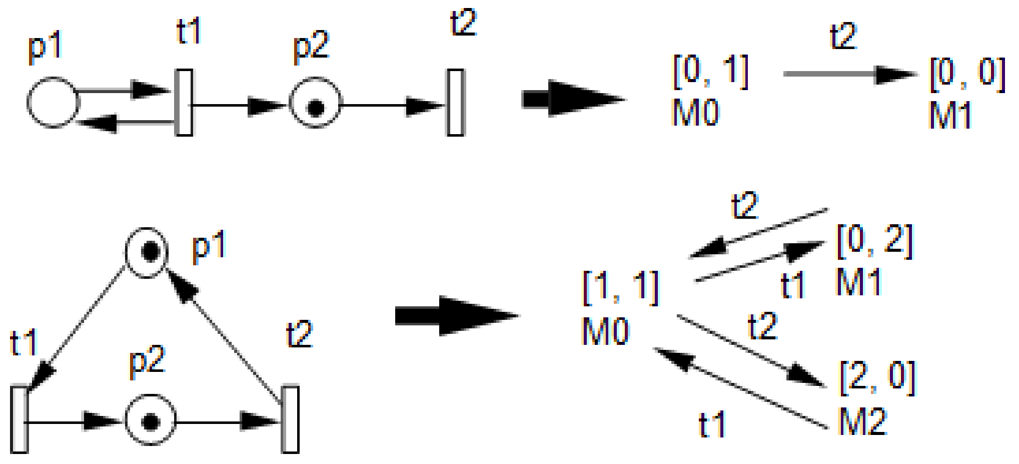


Figure 1.4: Reachability tree of Petri nets

Definition 8. (Coverability tree): [Mur89] Given a Petri net (N, M_O), from the initial marking M_O, we can obtain many "new" markings as the number of the enabled transitions. From each new marking, we can again reach more markings. This process results in a tree representation of the markings. Nodes represent markings generated from MO (the root) and its successors, and each arc represents a transition firing, which transforms one marking to another.

For bounded PN coverability tree is a reachability graph since it contains all possible reachable markings.

The above tree representation, however, will grow infinitely large if the net is unbounded. To keep the tree finite, we introduce a special symbol $'w'$, which can be thought of as "infinity." It has the properties that for each integer n, $w > n, w + n = wandw - n = w$. The coverability tree for a Petri net (N, M_0) is constructed by the following algorithm:
1: Label the initial marking M_0 as the root and tag it "new"
2: While "new" markings exist, do the following:
2.1: Select a new marking
2.2: If M is identical to a marking on the path from the root to M, then tag M "old" and go to another new marking.
2.3: If no transitions are enabled at M, tag M "dead-end".
2.4: While there exist enabled transitions at M, do the following for each enabled transition t at M:
2.4.1: Obtain the marking M' that results from firing t at M.
2.4.2: On the path from the root to M if there exists a marking M" such that $M'(p) \geq M"(p)$ for each place p and $M' \neq or \neq$ M", i.e., M" is coverable, then replace $M'(p)$ by w for each p such that $M'(p) > M"(p)$.
2.4.3: introduce M' as a node, draw an arc with label t from M to M', and tag M' "new."

Examples: Consider the net shown in the figure below :

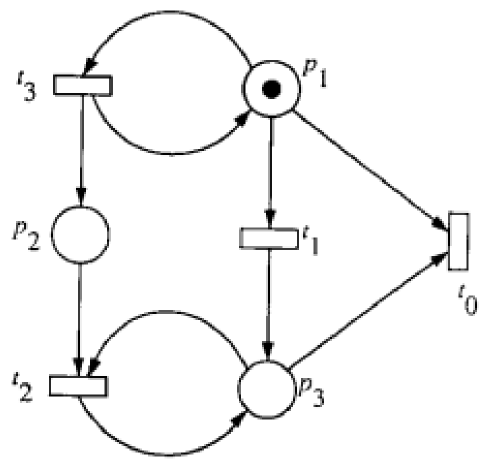


Figure 1.5: Example 1 Petri net

For the initial marking M_O = (1,0,0), the two transitions t_1, and t_3 are enabled. Firing t_1, transforms M_O to M_1 = (0,0,1), which is a "dead-end" node, since no transitions are enabled at M_1. Now, firing t_3 at M_O results in M_3' = (1,1,0), which covers M_O = (1,0,0).Therefore, the new marking is M_3= (1,w, O), where two transitions t_1 and t_3 are again enabled. Firing t_1 transforms M_3 to M_4 = (0 w 1), from which t_2 can be fired, resulting in an "old" node M_5=M_4. Firing t_3 at M_3 results in an "old" node M_6=M_3. Thus, we have the coverability tree of Petri net in figure1.5 shown in figure1.6

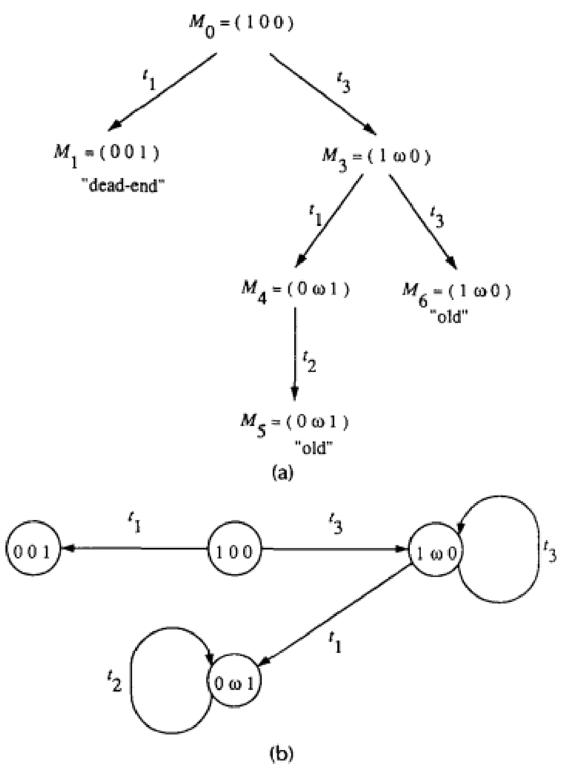


Figure 1.6: (a) The coverability tree of the net shown in Figure1.5.(b) The coverability graph of the net shown in Figure1.5

Consider the two nets shown in the figure below :

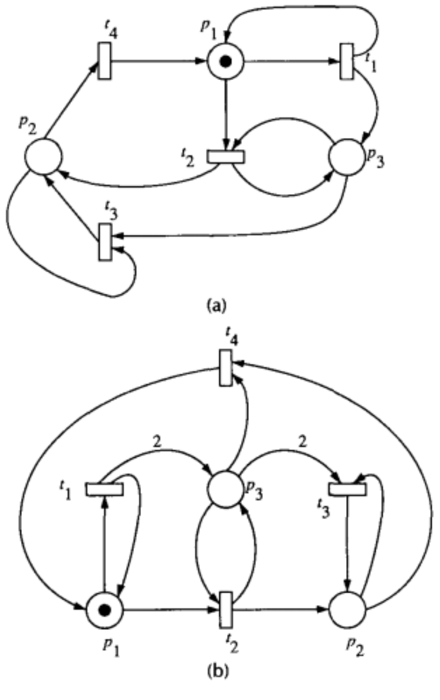


Figure 1.7: Two Petri nets having the same coverability tree. (a) A live Petri net. (b) A non-live Petri net [Mur89]

The two different Petri nets shown in Figure1.7 (a) and (b) have the same coverability as shown in Figure 1.8. The net shown in Figure1.7(a) is a live Petri net, while the net shown in Figure1.7 (b) is not live since no transitions are enabled after firing $t1$,t_2, and t_3.

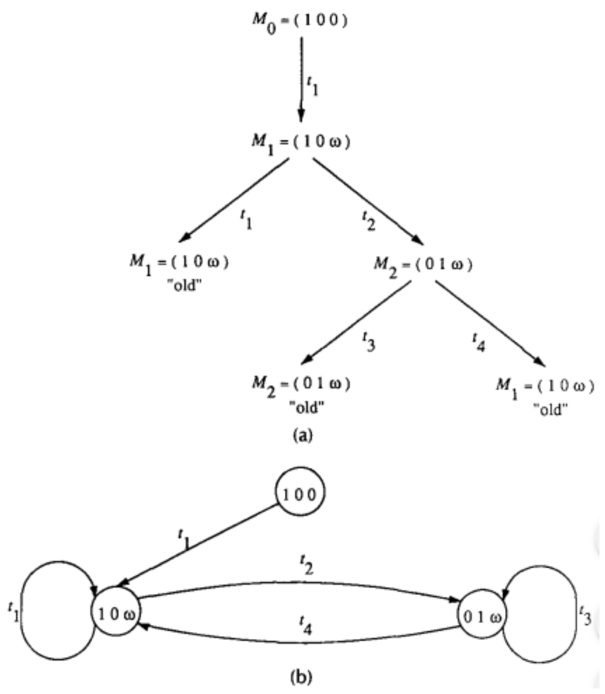


Figure 1.8: The coverability tree for both Petri nets shown in Figure1.7(a) and (b)[Mur89].

1.5 Petri Net models of key characteristics

Due to P/T net we can model: parallel process, Synchronisation, Shared ressources, Precedence relation... [CL09]

1.5.1 Parallel process

It represents the possibility that many processes are evolving together within the same system.

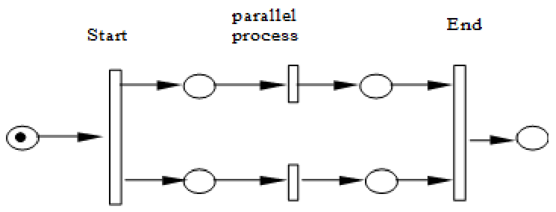


Figure 1.9: Petri Net specifying Parallel Process

1.5.2 Synchronisation

Synchronize the operations of two or more processes.

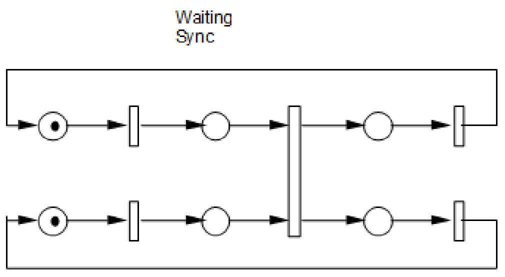


Figure 1.10: A Petri net specifying synchronization process

1.5.3 Shared ressources

Within the same system serval processes share the same resource.

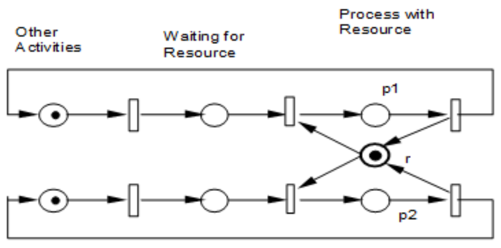


Figure 1.11: A Petri net specifying shared ressources

1.5.4 Precedence relation

Represent the precedence relation between activities in the same system.

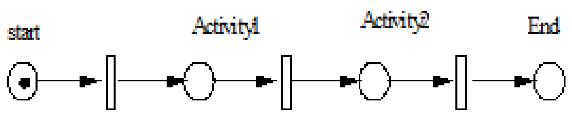


Figure 1.12: A Petri Net specifying precedence relation

Conclusion

Petri nets are well known low-level formalism very limited for modelling and verifying distributed and concurrent systems. The major drawback of low level Petri nets formalism is their inability to represent complex data which influences the behavior of the system, because it can model and specify only systems with discrete events systems with less expressivity, but it does not offer a direct way to address some modeling issues like dynamic changes. This interest will be the aim of the next chapter 2 when we will present a particular class of high level Petri nets which can specify systems with dynamic and changeable structure.

Chapter 2

Reconfigurable Object Nets

Chapter 2

Reconfigurable Object Nets

Introduction

The evolution in software and hardware systems from classical systems with rigid structures to open, dynamic, and flexible structures has inspired the extension of Petri nets to reconfiguration. The idea of reconfiguring Petri nets was launched in the early nineties and since then has been developed by several researchers at different levels of formalization.

Researchers in this field have achieved a large amount of theoretical results and practical applications.As we know petri nets (which presented in chapter 1) are a formalisms to model, analyze, simulate, control and evaluate the behavior of distributed and concurrent systems [Wol08], but this formalism does not offer a direct way to model reconfigurable systems, then we have to use an extension of petri nets called reconfigurable petri nets (RONs). Reconfigurable petri nets are proposed to make a very frequent changes in the model itself, there are many variants of reconfigurable PNs, and in this part we will focus on reconfigurable object nets (RONs or RdPORs in French). RONs are high-level nets with two types of tokens: object nets (place/transition nets) and net transformation rules (a dedicated type of graph transformation rules).

This chapter is composed of two parts: a first theoretical part which begin by defining the concept of net objects (also called net within nets), then introduce the concept of graph transformation applied to the PNs, a second part which mentions some avaible tools that were proposed for the simulation of these high level formalisms.

2.1 Object Nets

Tokens in a Petri net place can be interpreted as objects. Object nets or Nets-Within-Nets [Val04] are well suited for the modelling of distributed systems under the particular aspects of: hierarchy[1], mobility[2], encapsulation[3] with Petri nets, the Nets-Within-Nets paradigm provides an innovative modelling technique by giving tokens themselves the structure of a Petri net. These nets, called token nets or object nets, also support the object oriented modelling technique. In Nets-Within-Nets, one can distinguish between two levels in the Net: the system level and the token level. Tokens are two kinds: token-net and ordinary-token .

In system level places can contain tokens in form of P/T nets it called token Net, and in token level places contains tokens as ordinary tokens like in(chapter 1). In many applications objects not only belong to a specific environment but are also able to switch to a different one, and this what made the nets-within-nets very useful for modeling interesting applications in many domains as in the field of mobile agents [PBM06] [Pad08].

2.2 Reconfigurable Object Nets

Reconfigurable object nets [Bie08] are high level Petri nets and a type of reconfigurable nets which enriched Petri nets with the concepts of orientation, encapsulation and reconfiguration. RONs[4] have two types of tokens: net and rule tokens. In RON not only the follower marking can changed but also the structure can be changed by rule application to obtain a new P/T net. Net places are a P/T nets which can move from a place to another place in the system, where it can change its marking and structure also, rule places are production by double pushout rules and it will never be changed or move in the system it just uses by transform transition to made changes in the token net, and rules are based on a partial morphism between left-hand and right-hand side of the rule. Transitions in the system level decide about the movement of token-net from one place to another. These transitions decide if the marking or the structure of a token-net must be changed. To change the marking of a token-net, the transition in the system level triggers a transition in the token-net level. However, to change the structure of a token-net, a transition, in the system level requires a token-rule. The token-rule decides how the structure of the token-net must be changed when some transition, in the system level is fired. In RONs, reconfiguration of the structure concerns only the token-net and not the system level. This reconfiguration is defined by a set of token rules, based on the graph transformation techniques.

The two examples in the figures below explain the difference between fire and transform transition. The figure 2.1 explain the fire transition where in token-net1 (Token level "a P/T net ") the transition T_1 is enabled, so the fire Transition in the system level will fired T_0 and provide the same object with new marking of Token-net1

[1]dividing the net into a number of sub-nets to break down the complexity of a large model.
[2]The object net moved inside the system
[3]encapsulation consists of masking the details of the implementation of an object, by defining an interface.
[4]Reconfigurable Object Nets.

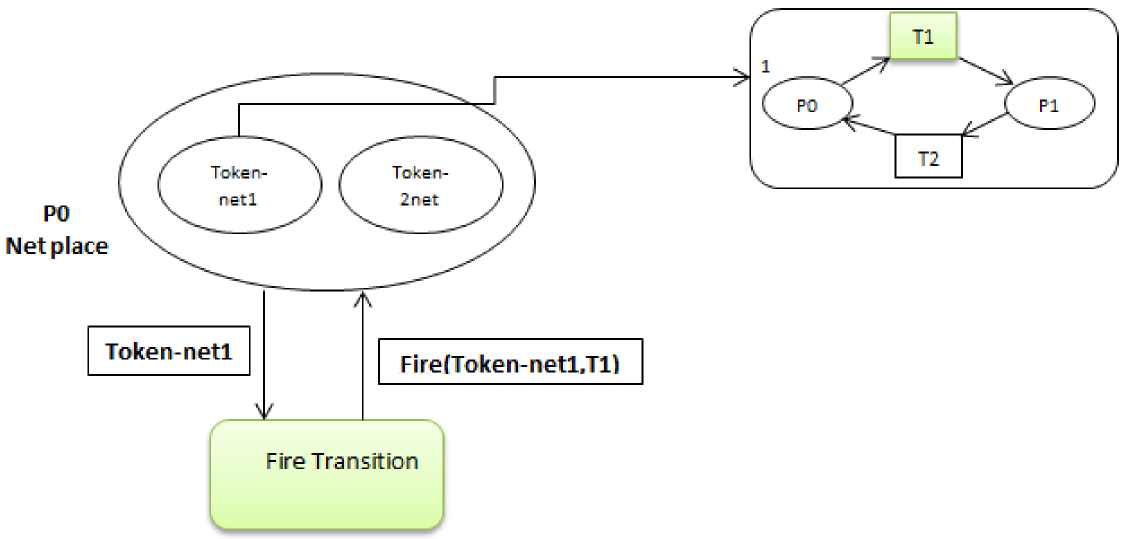


Figure 2.1: Example of a Fire transition

Figure 2.2 explain the Transform transition where Token-net1 (in token level "P/T net") in place P_0 corresponds to token-rule1 in place P_1, so the Transform transition in system level will transform the token-net1 according to token-rule1 and provide a new Token-net in place P_2 with different structure.

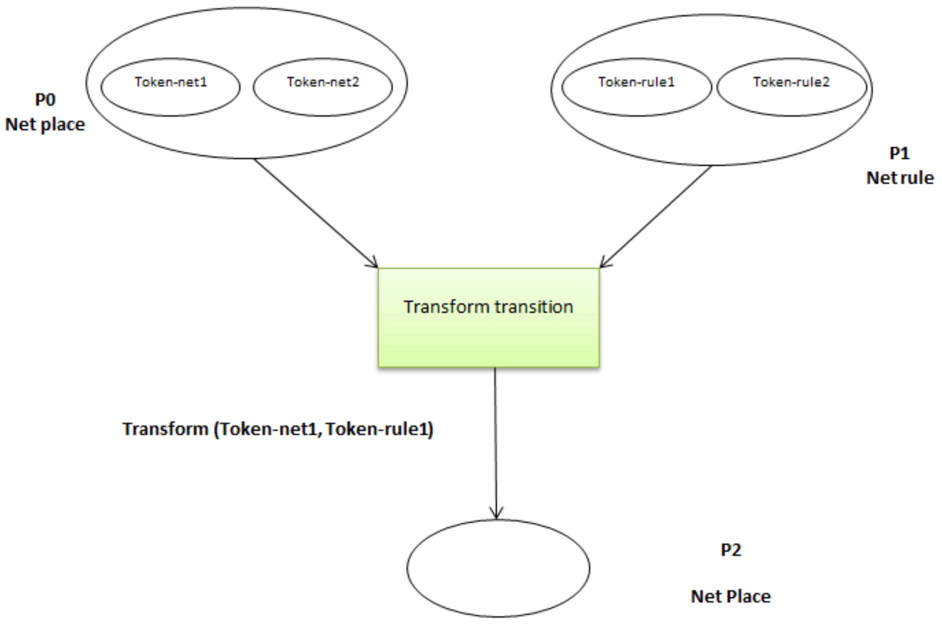


Figure 2.2: Example of Transform transition

2.3 Transformation techniques

Transformation techniques [Kö18] are inspired from graph transformation of two basic constructions: union and transformation on Place/Transition nets (P/T nets). The union construction takes two Nets N1 and N2 and yields another net N3, but the transformation construction takes one P/T net N1 and yields another net N2. These two constructions are the two basic reconfigurable techniques for P/T nets. Union and transformation are based on morphism concept defined over P/T nets.

Definition 6.(Graph)
A graph G is a tuple $G = (V, E, s, t, l)$, where:
•V is a set of nodes,
•E is a set of edges,
•s: E $\longrightarrow$V is the source function
•t: E $\longrightarrow$ V is the target function
•l: E $\longrightarrow$ V is the labelling function
Given a graph G, we denote its components by VG, EG, sG, tG, lG Given an edge e $\in$ EG, the nodes sG(e), tG(e) are called incident to e.

Central notion in graph rewriting[5] is a graph morphism. Just as a function is a mapping from a set to another set, a graph morphism is a mapping from a graph to a graph. It maps nodes to nodes and edges to edges, while preserving the structure of a graph. This means that if an edge is mapped to an edge, there must be a mapping between the source and target nodes of the two edges. Furthermore, labels must be preserved. Graph morphisms are needed to identify the match of a left-hand side of a rule in a (potentially larger) host graph. As we will see below, they are also required for other purposes, such as graph gluing and graph transformation rules

Definition 6.(Graph transformation) A graph transformation system is a tuple $G = (G_0, R)$ where:
– G_0 is an initial graph or start graph
– R is a set of graph transformation rules.

2.4 Morphisms over P/T nets

Net morphisms are given as a pair of mappings for the places and the transitions preserving the structure and the marking. Given two P/T nets:
N1$(P_1, T_1, Pre_1, Post_1, M_1)$ and N2$(P_2, T_2, Pre_2, Post_2, M_2)$, morphism f between the two nets $N1$ and $N2$ is a function $f : (N_1 \to N_2)$. We have: $f = (f_P, f_T)$, such that: $f_T(T_1 \to T_2)$, and $f_P(P_1 \to P_2)$ are two morphisms which:
Map transitions into transition and places into places, respectively. f_P and fTt satisfy:

$$(1) \quad \forall\, p'_1 \in {}^o t' \implies \exists p_1 \in {}^o t \;:\; f_P(p_1) = p'_1 and \quad \forall\, p'_1 \in t'^o \implies \exists p_1 \in t^o \;:\; f_P(p_1) = p'_1$$

$$(2) \quad f_P(M_1(p)\,) \;\leq\; M_2(f_p(p)\,)$$

[5]Graph transformation

The labels and the capacity need to remain the same when mapping one net to another.

Example(Morphism) In the example below we have two P/T nets PN_1 with blue color and PN_2 in red color, T_1' is the image of T_1 and the source of T_1' is P_1' which is in PN_2 the image of P_1 the source of T_1 in PN_1, also the target of T_1' is P_2' in PN_2 which is the image of P_2 the target of T_1 in PN_1 .

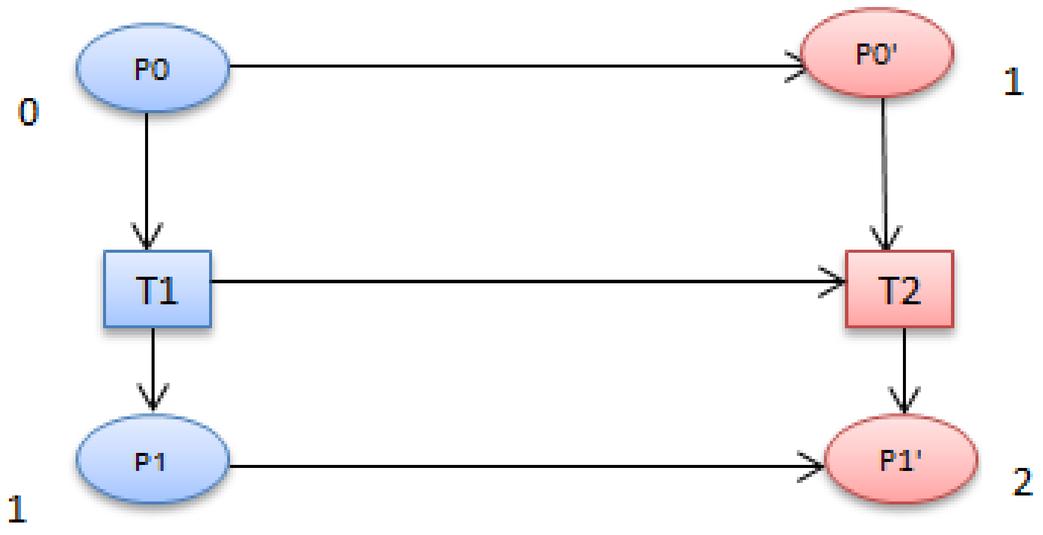


Figure 2.3: Example of Morphism

2.5 Union P/T nets as a pushout

Based on the morphisms on P/T nets, it is possible to define a specific construction which is the pushout (or union) of two P/T nets [KCD+14]. Let $N_1(P_1, T_1, Pre_1, Post_1)$ and $N_2(P_2, T_2, Pre_2, Post_2)$ be two nets, with the two morphisms:
$f : I \to N_1$ and $g : I \to N_2$. The net I is said a common interface between N_1 and N_2. The union of N_1 and N_2 is the Net $N(P, T, Pre, Post)$ defined using the two morphisms: $f' : N_1 \to N$ and $g' : N_2 \to N$. We write $N = N_1 +I\, N_1$. The operator $+I$ is called the pushout construction or the gluing (union) operator. [KCD+14]

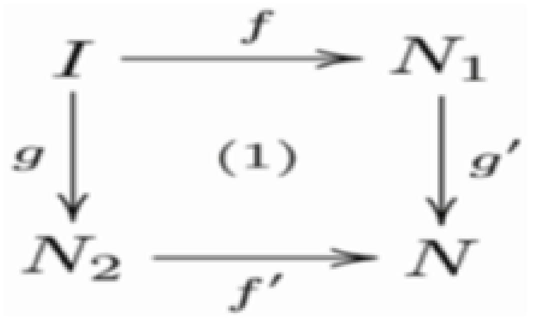


Figure 2.4: Pushout diagram [Kö18]

Example Below (Figure 2.5), an example of a gluing[6] construction.

Let the two graph morphisms:
$\varphi_1 I \to G_1$ and $\varphi_2 : I \to G_2$ given, since the interface I is present in both graphs G_1 and G_2, we can glue the two graphs together to construct a graph $G_1 + IG_2$.

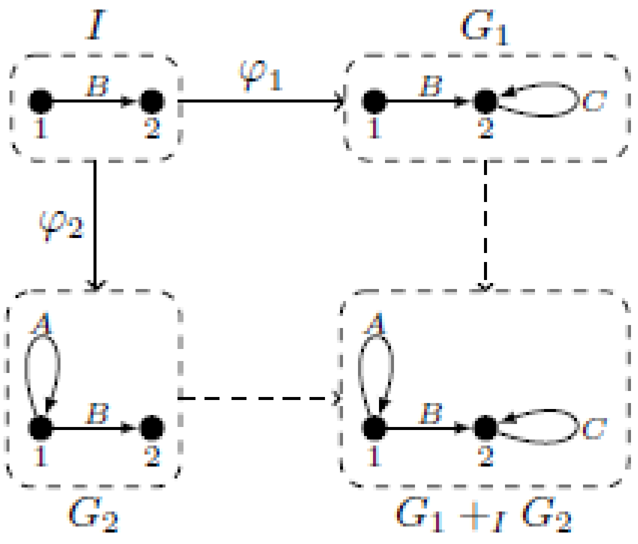


Figure 2.5: Example of gluing [Kö18]

2.6 Rules and transformations

Based on the P/T gluing construction, the P/T transformation is constructed as a DPO [7]. Let L, K, R and C be four P/T nets. A transformation $f : N1 \to N2$ transforms the P/T net N_1 to the P/T net N_2 using the rule $r = (L, K, R)$ and the match $m : L \to N_1$ and we write $f = (r, m)$, below in(Figure 2.6) an example of double pushout, k_1, k_2, m, c, and n are morphisms. The P/T net C is called the context of transformation.[Kah16].

[6]Union
[7]Double pushout

$$\begin{tikzcd}
L \arrow[d, "m"'] & K \arrow[l, "k_1"'] \arrow[r, "k_2"] \arrow[d, "c"] \arrow[l, phantom, "(1)", shift right=6ex] \arrow[r, phantom, "(2)", shift right=6ex] & R \arrow[d, "n"] \\
N_1 & C \arrow[l] \arrow[r] & N_2
\end{tikzcd}$$

Figure 2.6: Double pushout [Kah16]

2.7 Avaible tools

2.7.1 Tina

Tina *(TIme petri Net Analyzer)*is a toolbox for the editing and analysis of Petri Nets, with possibly inhibitor and read arcs, Time Petri Nets, with possibly priorities and stopwatches, and an extension of Time Petri Nets with data handling called Time Transition Systems. TINA has been developed in the OLC, then VerTICS, research groups of LAAS/CNRS [thpb].

2.7.2 PIPE

PIPE(*Platfome Independent Petri net Editor*)is an open source, platform independent tool for creating and analysing Petri nets including Generalised Stochastic Petri nets. Petri nets are a popular way for modelling concurrency and synchronisation in distributed systems.

PIPE2 began life in 2002/3 as an MSc. Group Project at the Department of Computing, Imperial College London called "The Platform Independent Petri net Editor PIPE" [thpa].

2.7.3 ReConNet

ReConNet(*ReCongurable Net*)is a visual editor for the Reconfigurable Petri net, it was implemented on Java 6 by ten students from the year 2010 and nine students In 2011, these students were working with the Double-Pushout approach but with another construction technique called Cospan where the left side and the right side of the rule are integrated in the interface (L K R). The ReConNet tool has been developed to model and simulate the reconfigurable networks is handled appropriately [Mar12].

Conclusion

This chapter introduced the new extension of high level Petri nets, which is *Reconfigurable Object Nets(RONs)*, that can modify their own structures by rewriting some of their components, so that it can model and specify the dynamic of the reconfigurable systems. IT gives some basic concepts to finally give the complete definition RONs.

In the next chapter we will use RONs to develop a tool that can model and specify reconfigurable nets and explain the different steps to realise this tool.

Part II

A Tool for Reconfigurable Petri Nets

Chapter 3

Analysis & Design

Chapter 3

Analysis and Design

Introduction

After having learned the necessary theoretical points, we specify the aim of our project which is to realize a tool for the modeling, simulation and analysis of the RONs, we pass to the application development.

This chapter is composed of three sections. The first section of analysis specify our needs. In the second section, the global design is raised, in which we show an abstract solution that satisfied our needs and specifies functionalities of the tool we are trying to achieve. The third part present the detailed design in which we detail our solution with UML[1] diagrams.

3.1 Analysis

As we see in the first part, the reconfiguration can make systems more complex and impose new kinds of errors and anomalies, so we need a sophisticated verification process to ensure reliability of these systems.

Knowing that there is some tools to model, analyze and simulate these kind of systems but stills limit, complex, not suitable neither sufficient and specially a lack of analysis. For this reason we decided to develop a tool which can model, simulate and analyze the reconfigurable systems more easy than others with reconfigurable object nets(Rons) already presented in (chapter 2 part I).

3.2 Design

After studying the project aims, we have to identify the global architecture of the application, the set functionalities, and the relation between them in a detailed way.

[1]The Unified Modeling Language is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

3.2.1 Global Design

Figure 3.1 illustrates the global architecture of the application in the figure below .

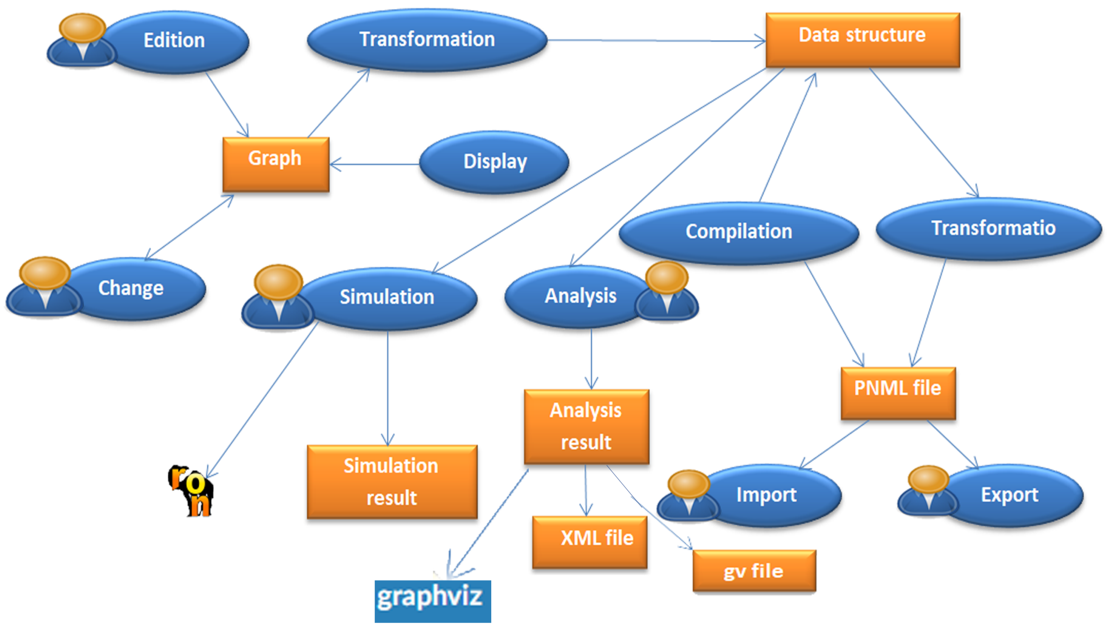


Figure 3.1: Global Architecture of the application

According to the global design, the user can create and edit Petri nets graphically, and at the same time these graphs will be automatically transformed into data structures (.pnml[2], .mml[3], .poml[4], .xml[5]) in order to store it and import it in other tools or on the same tool and import files with these extentions from other tools also when he wants. The user can also analyze,simulate and display the data structure with the same tool or by using other tools like PIPE, TINA.

3.2.2 Detailed Design

In order to release our application, we used object oriented method for the design and modeling of the different modules of our system for more expressivity and details we introduce some class diagrams with UML.

[2]Petri Nets Markup Language
[3]Morphisms Markup Language
[4]Pushouts Markup Language
[5]Extensible Markup Language

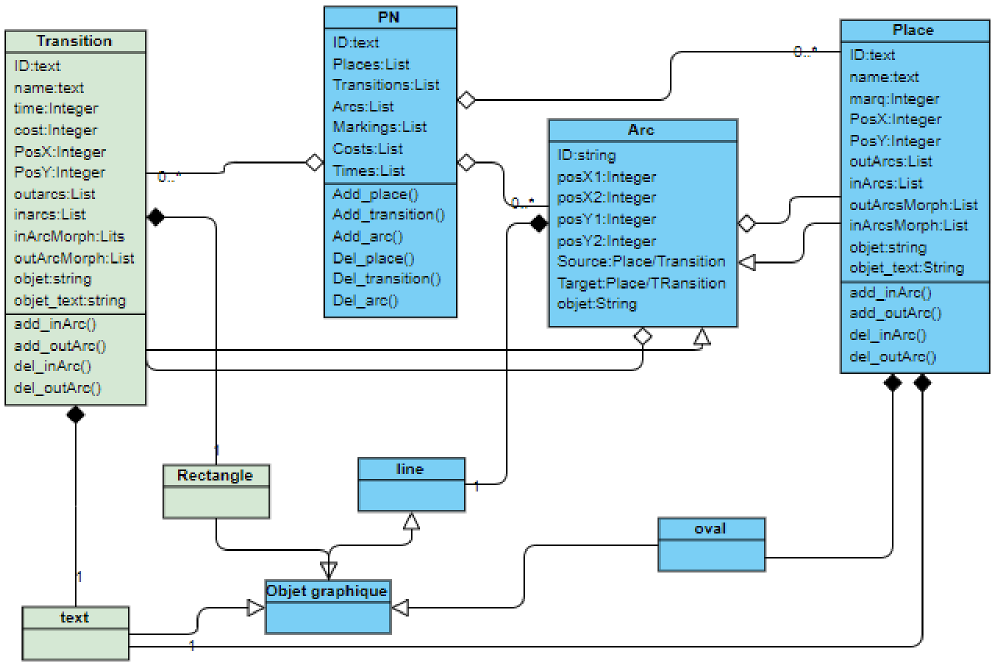


Figure 3.2: Class Diagram (P/T nets)

The figure 3.2 shows the main classes that are needed to make a P/T net, The Class PN represent the P/T net model it contains all the necessary attributes which needed to define a P/T net, a list of places object, a list of transitions objects and a list of arcs objects with the necessary operations that needed to model the Petri net, addition and remove of different components.The Place class contains the two positions PosX and PosY which indicate where the object will be drawn, a reference on a oval object (a graphical object that represent a place), a list of inArcs and outArcs plus the main methods to add or remove in and out arcs.The same with Transition class except the reference is on a rectangle object. The Arc Class represent the the link between two objects place-transition or transition-place it contains the following attribute: PosX1, PosY1, PosX2 and PosY2 represent the graphical position of the line, source object(Place/Transition), and the target object (Place/Transition).

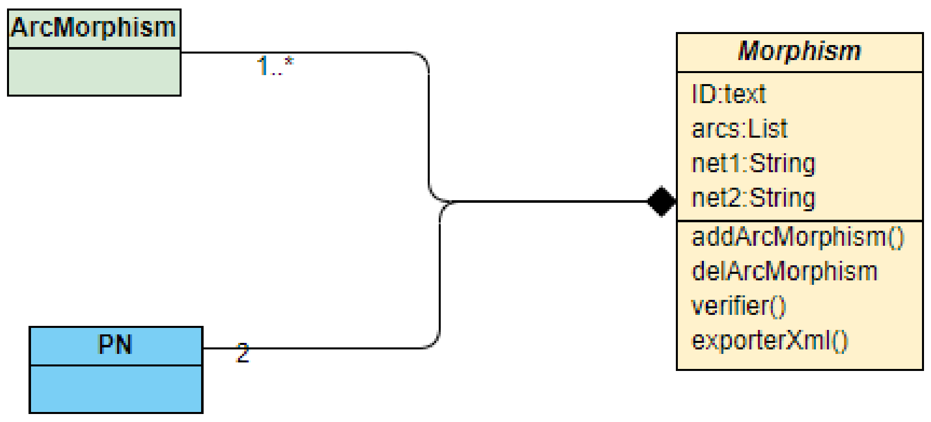


Figure 3.3: Class Diagram (Morphism)

The previous figure 3.3 shows the main classes that are needed to edit morphisms over P/T nets, Morphism class contains two objects of class PN and list of object of class ArcMorphism.

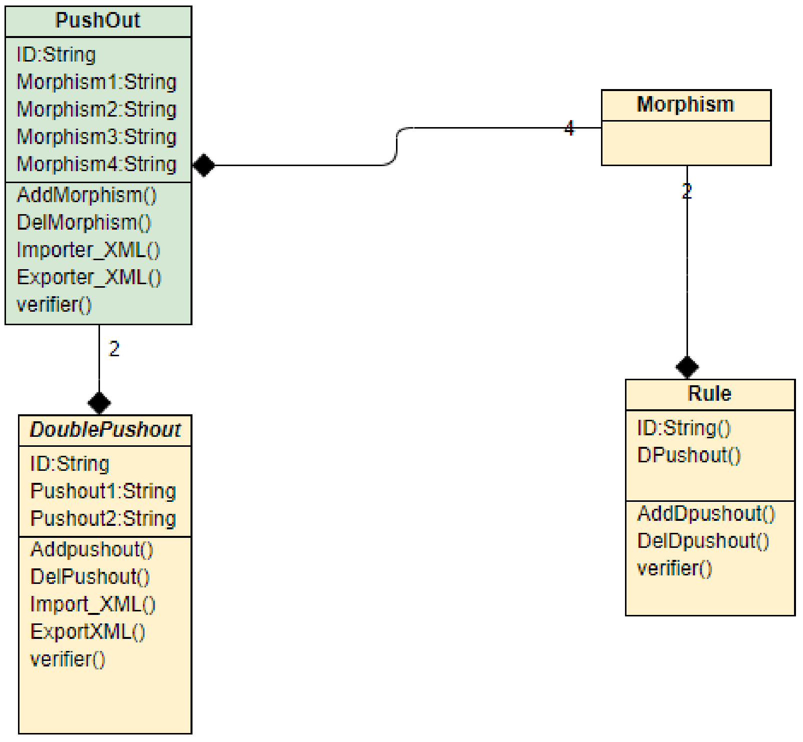


Figure 3.4: class Diagram (Rule)

The previous figure 3.4 shows the main classes needed for creation a Pushout, a Double-Pushout and a production rule $p = (L, K, R)$. The Pushout class contains four attributes that represent morphisms (class Morphism defined in the previous figure3.3) and the necessary methods for adding, deletion, verification, export and import under file (.PNML). The Double-Pushout class contains two attributes that represent two pushouts as well as adding, deleting, importing and exporting under file (.dpml) methods. The Rule class contains two objects of the Morphism class, three objects of PN class with adding, deleting, and verifying methods.

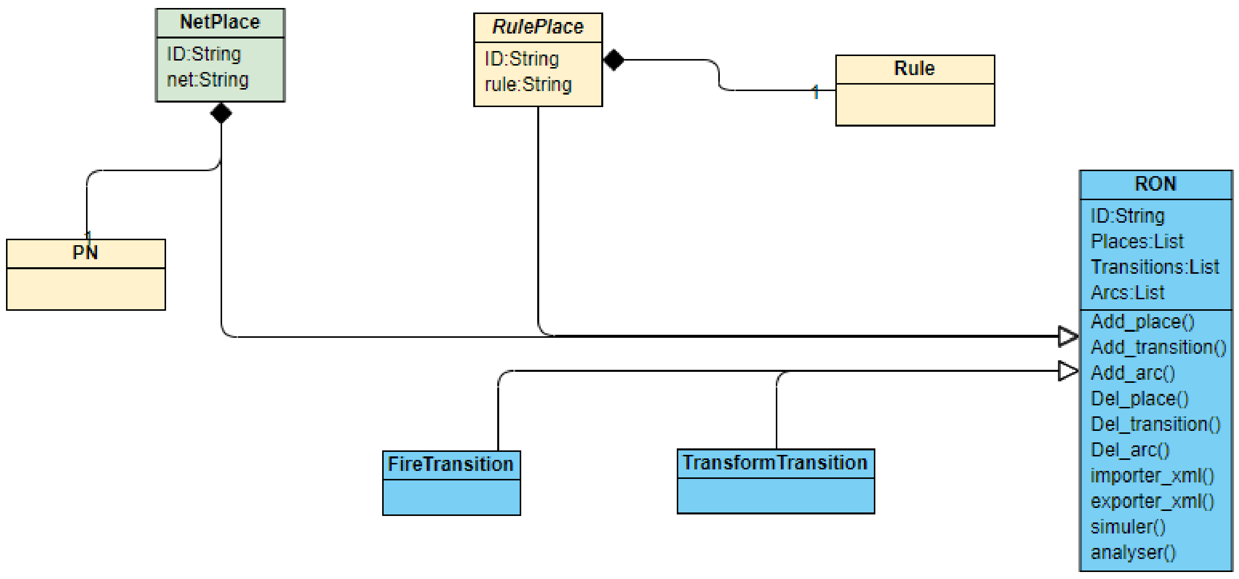


Figure 3.5: class Diagram (RON)

The previous figure 3.5 contains the principle class which we needed to create and edit Reconfigurable object nets, it contain Ron class which compose of list of places which can be an object of class NetPlace or RulePlace, list of transitions which can be an object of class Fire or Transform transition and list of arcs with some methods add and delete place, transition and arc, import and export files, simulation and analysis. NetPlace class contain a list of object from the class PN, RulePlace class contain a list of object from the class Rule.

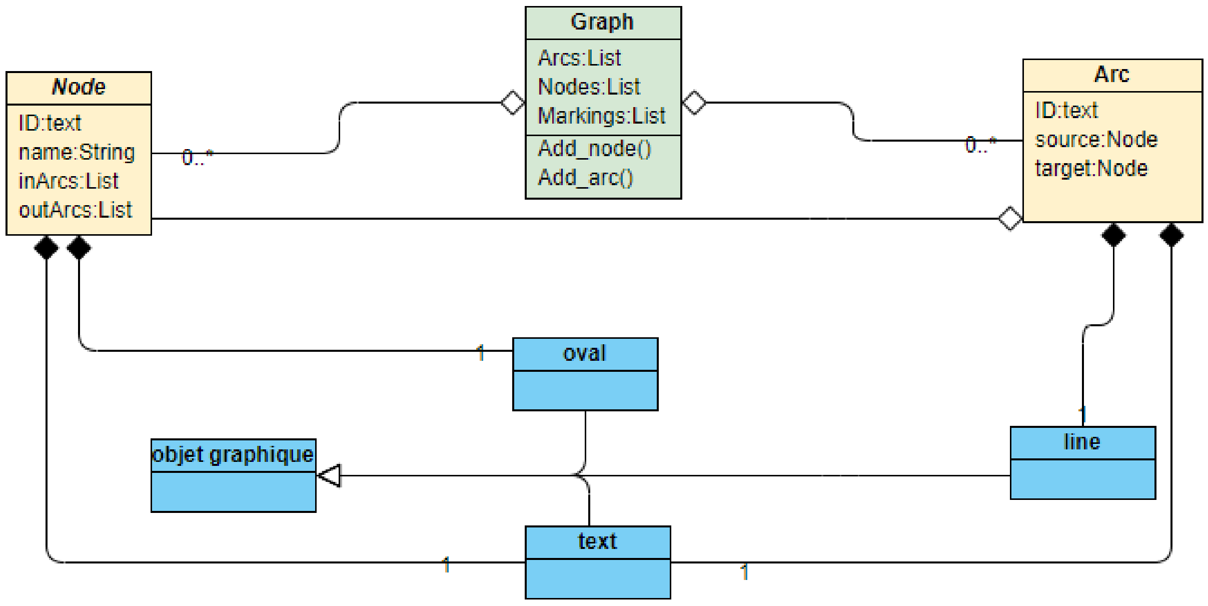


Figure 3.6: class Diagram (Graph)

The previous figure 3.6 contain the primary classe to create a reachability graph of a petri net. Graph class contain a list of arcs, nodes and marking states. Node class contains two object of class arc and a reference to a oval object(graphic object that represents a node). Arc class contains two object of class node source and target nodes.

Conclusion

In this chapter, we have presented analysis that has directed us to design our tool. Then, we passed to the conception phase which is divided in two steps (global conception, and the detailed conception). The first step illustrates a global design of our application. The second step presented a detailed description of the coding/decoding techniques of process plan to move to the next stage.

The next steps of the project are an implementation of the proposed design, testing and discussing some experimental results. These steps will be the aim of the next chapter4 which is the last chapter.

Chapter 4

Implementation

Chapter 4

Implementation

Introduction

After analysis and design steps that are mentioned in the previous chapter (chapter 3), we have to pass to the next steps of the project, which are coding and test. These phases aim to implement a tool for modeling and simulation of reconfigurable object nets.

This chapter includes two sections. The first section introduces briefly the development tools and languages that we have learned and exploit them in the realisation of our project. The second section presents the main implementation results of our final application.

4.1 Development Tools and Languages

In this section, we present different tools and languages, that help us during the realisation of our project in the two levels (programming level, and theoretical level).

4.1.1 Python programming language

Python is an intelligent programming language that we have used it in the implementation of our application. It is easy to learn, because it is flexible, and its syntax doesn't hard to learn. A Python program is short than other languages' programs, because of the availability of many implemented functions. Python is an open source and untyped programming language. It is available for all these operating system (Windows, LINUX, Mac OS).

4.1.2 PyCharm Programming Editor

PyCharm is an open source Integrated Development Environment (IDE), used for python programming. It is a powerful coding assistant, it can highlight errors and introduces quick fixes based on an integrated Python debugger. It is a suitable editor for writing and testing many lines of code and classes, since it offers a structural project view, and a quick files navigation.

4.1.3 Tool Kit Interface "Tkinter" Package

Tool Kit Interface in short "*Tkinter*" [?], it is an open source *Graphical User Interface* (*GUI*) package. It is intended for Python programming language. We have preferred the *Tkinter* toolkit for developing GUIs of our application, because it is simple to learn it , and it is a powerful toolkit. It is available on both operating systems (Windows, Linux, and Mac OS).

4.1.4 XML

(*Extensible Markup Languge*)XML is a file extension for an Extensible Markup Language (XML) file format used to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. We used this language to store our files (P / T, Morphisms, Rules, Pushout, Double-Pushouts, RON, Reachability Graphs) from our tool.

4.1.5 PNML

(*Petri Net Markup Language)* PNML is a proposal of an XML-based interchange format for Petri nets. Originally, the PNML was intended to serve as a file format for the Java version of the Petri Net Kernel . But, it turned out that currently several other groups are developing an XML-based interchange format too. So, the PNML is only one contribution to the ongoing discussion and to the standardization efforts of an XML-based format.

4.1.6 Graphviz

(*for Graph Visualization Software*) is a package of open-source tools initiated by ATT Labs Research for drawing graphs specified in DOT language scripts. It also provides libraries for software applications to use the tools. Graphviz is free software licensed under the Eclipse Public License.

4.1.7 Document Preparation System LaTeX

LaTeX [1] is a powerful and flexible typesetting system for producing high quality technical and scientific papers. It based on the tags language. It follows the design philosophy of separating presentation from content, thus authors focus on what they are writing, not on what is displayed, because the appearance is handled by LaTeX. The appearance incl+udes many aspects, document structure (part, chapter, section, ..etc), figures, cross-references and bibliographies. It is more familiar to a computer programmer, because it follows the code-compile-execute cycle.

4.1.8 Typesetting Editor (TeX MAKER)

TeX MAKER [?] is a free and open source editor for drafting papers, based on LaTeX system. It supports a powerful spell-checker, code auto-completion, and a *pdf* displayer. We have used TeX MAKER to draft our report and make our presentation, because it produces high quality papers and talks.

4.2 Implementation

Moreover mentioned above (section 1.4), we used the oriented object programming (OOP) paradigm to implement our tool, and using a set of software and hardware which are summarized in the following table 4.1.

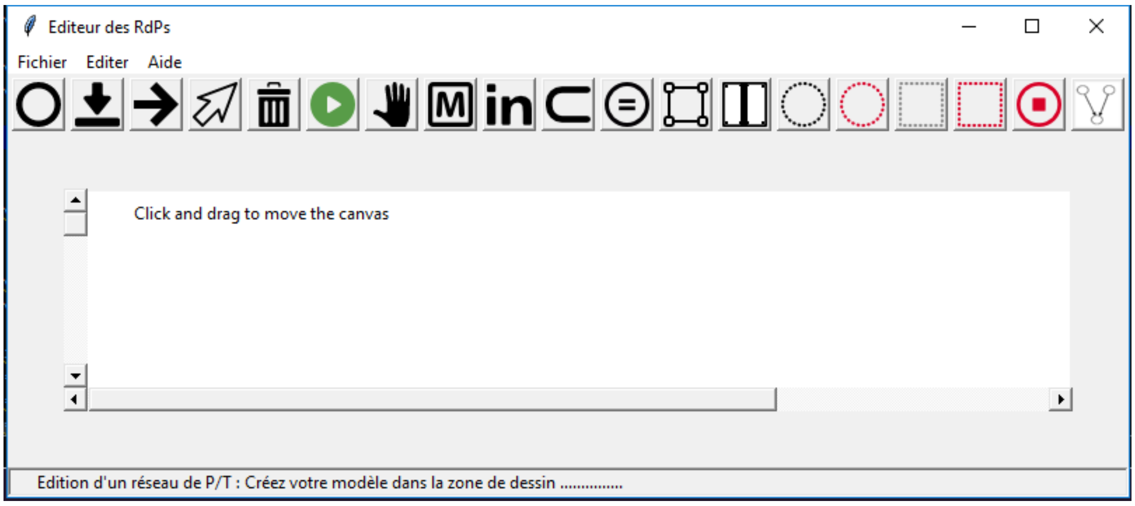
Software/Hardware	**Version**
OS	Microsoft Windows 10 Home, 64bits
CPU	Intel(R) Pentium(R) CPU 3525U@1.90GHz
RAM	4.00Go
Python Interpreter	3.5.0
PyCharm	2018.3.3
Tkinter	8.6

Table 4.1: Software/Hardware versions

4.2.1 Application Home

After having followed several stages of development, we have implemented a graphical tool that allows to model, simulate and verify Reconfigable Petri nets using Double-Pushout approach. This tool is not only for the RONs but it also allows the modeling and simulation of the low level petri nets (P / T nets). It allows the editing and the verification of the different properties and conditions of morphisms, Pushouts and DoublePushouts, as well as the ability to export them to XML files with a specific grammar we've proposed to which extensions have been given (.MML, .PML, DPML). The tool has been designed to be easy to use with simple interface (see Figure 4.1). It consists of a menu bar, a toolbar, a drawing area with scrollbars and a status bar. A menu bar contains the different publishers that can operate independently of each other. The options for a new creation are: New P / T Net, New Morphism, New Pushout, New Double Pushout and New RON. The options for opening files (also called import) are: Open a P / T net, Open a morphism, Open a Pushout, Open a Double-Pushout, Open a RON.

Figure 4.1 depicts our final application.



Edition d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.1: Application Home

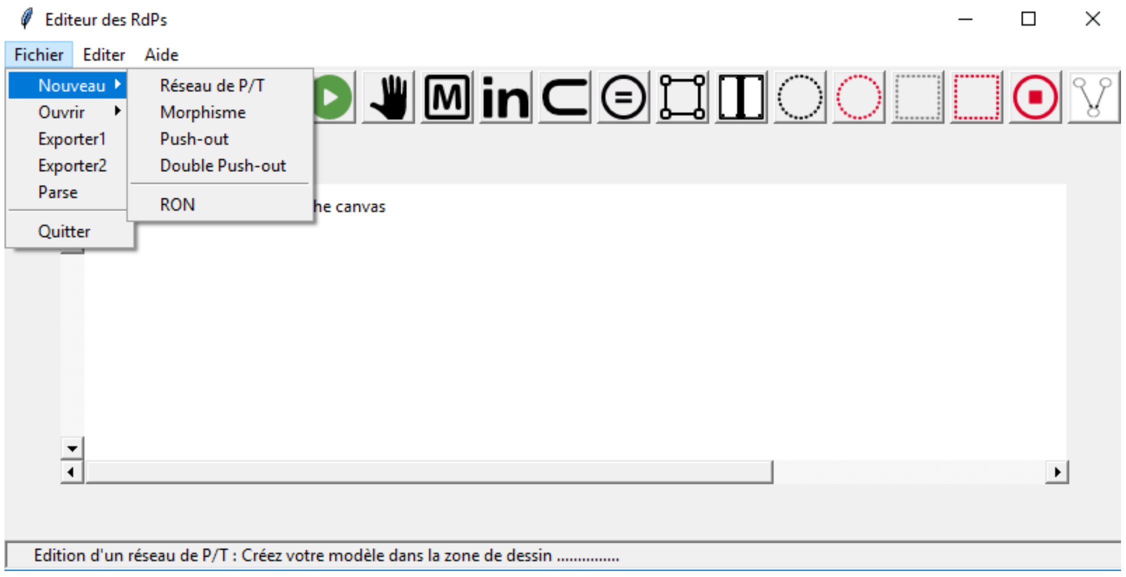
4.2.1.1 Menu Bar

We developed many GUIs (Graphical User Interface) to facilitate the use of the application. The application contains a menu bar with three menus (File, Edit and Help), the main menu is the file menu.

We will give the description of the principle menu "File menu".Figure 4.2 illustrate file menu and its commands.

File Menu:

File menu contains two sub menus: new and open menu. The sub menu new allows the user to open new drawing area for Petri net, Morphism, Pushout, Double Pushout and RON. The sub menu open allows user to recover his models for editing and simulation. If the user click at the exporte1 label in the master menu, it's allows to save their models, in their extensions (.pnml, .pml, .dpml), but export2 allows to save P/T nets in the extention .pnml with two new properties in transitions time and cost.



Edition d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.2: Menu Bar

4.2.1.2 Tool Bar

The toolbar in figure4.3, the button ○, used to draw places *(P / T nets)*, the button ⬇, used to draw transition *(P / T nets)*, the button ➔, used to create arcs between places and transition in the drawing low level net or RON, the button ⬀ is used to drag the model of petri net or reconfigurable object nets in the drawing area, delete button 🗑, is used to delete a graphical object from the drawing area, run simulation button ▶, use to run simulator; for Petri net model; or Ron_s models, move button ✋, used to drag or move the graphical object in drawing area, the button ⬚, used to draw fire transition in the ron editor, the button ⬚ using to draw transform transition in the ron editor, the button ◌ used to draw rule-place for ron, the button ◌ used to draw net-place in the drawing area of the ron editor. The last four buttons run just in the drawing area of the RON,stop simulation button ⊙

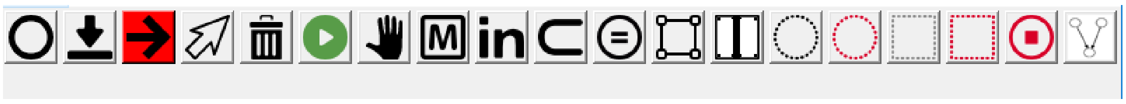


Figure 4.3: Toolbar

4.2.2 Application features

Our tool consists of four modules: P/T net editor, Morphism editor, Pushout editor, Double-Pushout editor that has been made independently. Below is the description of the features of each editor.

4.2.2.1 P/T nets editor

This editor allows the creation and modification of a Petri net in the drawing area using the toolbar. Eventually, the net P / T editor can generate a PNML code that can be exported to other tools, or compiled and displayed as a P/T graph net.

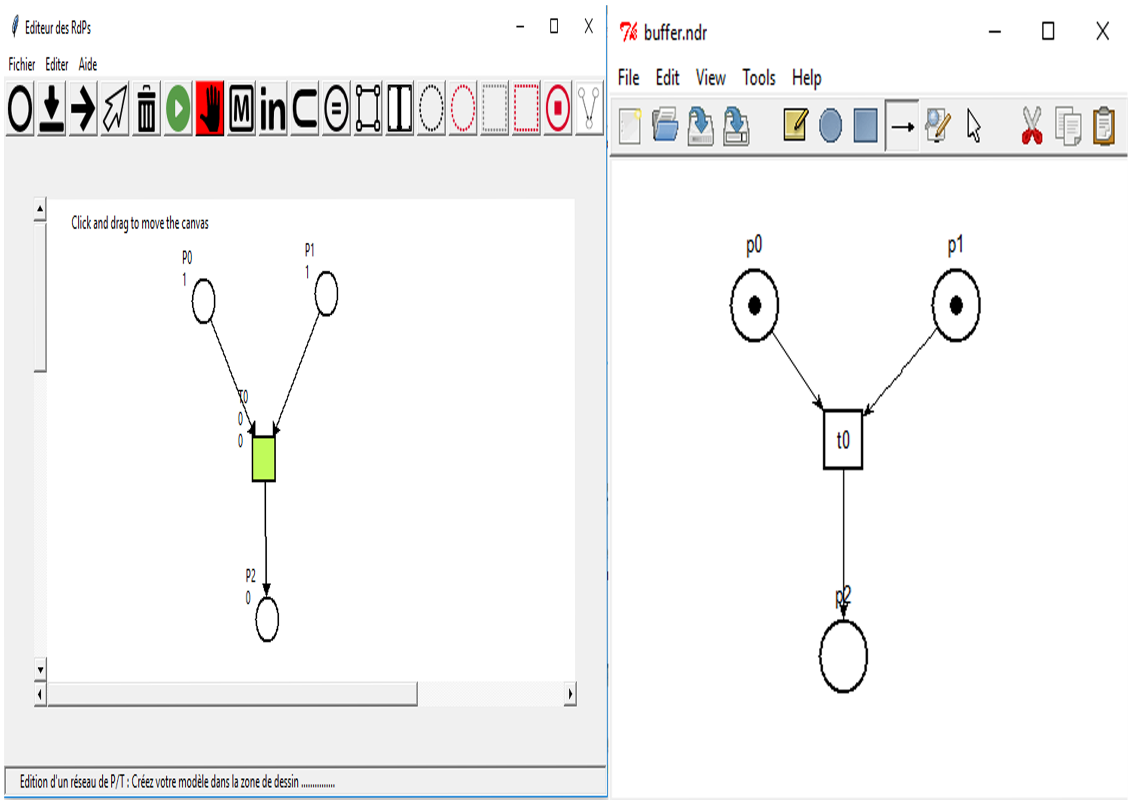
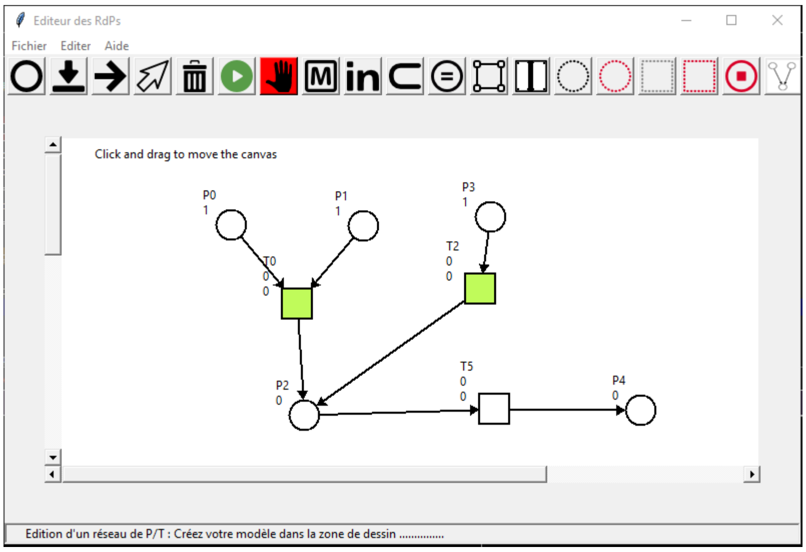

Editeur des RdPs
Fichier Editer Aide
Click and drag to move the canvas
P0
1
P1
1
T0
0
0
P2
0
Edition d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....
buffer.ndr
File Edit View Tools Help
p0
p1
t0
p2

Figure 4.4: Exporting/Importing in Tina

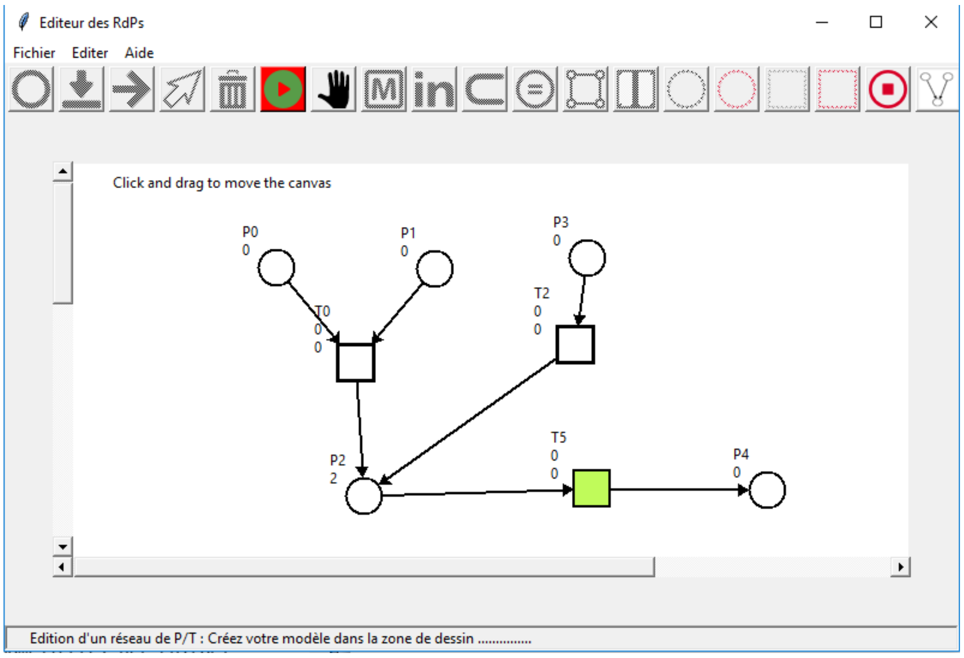
Figure 4.4 represent in the left side a P/T net editing with our tool and in the right side the same model exported in PNML form and re-imported it in Tina tool.
With our tool we can always see the enabled transitions,in which if there is an enabled transition, it will represented with green color but with Tina We noticed that it does not be colored as To in PN shown in figure 4.4.In transition also we noticed that the is no information with Tina tool while with our tool we can define the time and the cost.



Edition d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.5: Petri net editor

The P/T nets editor can also simulate P/T nets, the transition T0 and T2 in figure 4.5 are enabled, and in the figure4.6 shows the PN after firing the two transitions.



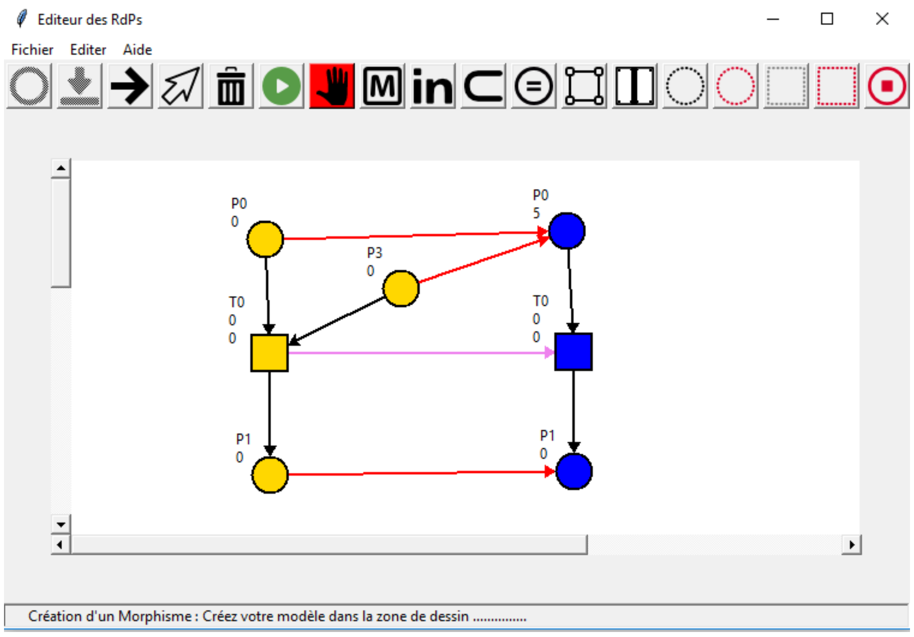
Edition d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.6: Petri net editor(simulation)

In figure 4.6 the two transitions T0 and T2 become not enabled while T5 become an enabled transition. We can't modify the PN only when the simulation is stopped by the user by pressing in stop simulation button ⦿ and back to the initial marking and reactive the toolbar again.

4.2.2.2 Morphism editor

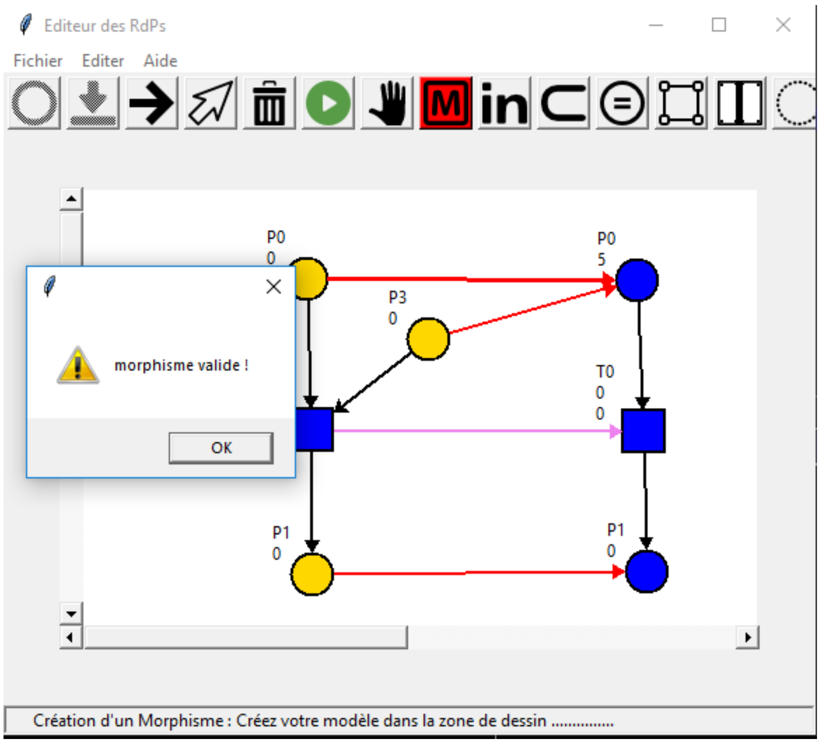
This tool allows to create, modify and verify morphism between two P/T nets, and generate a mml code to export and re-import it other time.



Création d'un Morphisme : Créez votre modèle dans la zone de dessin ....

Figure 4.7: Morphism editor

To create a morphism we need two P/T nets (the blue and yellow PNs) and then connect them with arcs between nodes(place or transition) of the first P/T net with their images in the second P/T net.The arcs with red color represent *Fp* function and whose with purple color represent *Ft* function. The morphism editor allows to verify if the morphism is valid or not and display a message which indicates why. The same thing with the verification if is a injectif ,strict or inclusif morphism.Figure4.8 represent the verification if the morphism is valid or not. As shown in figures 4.8 4.9

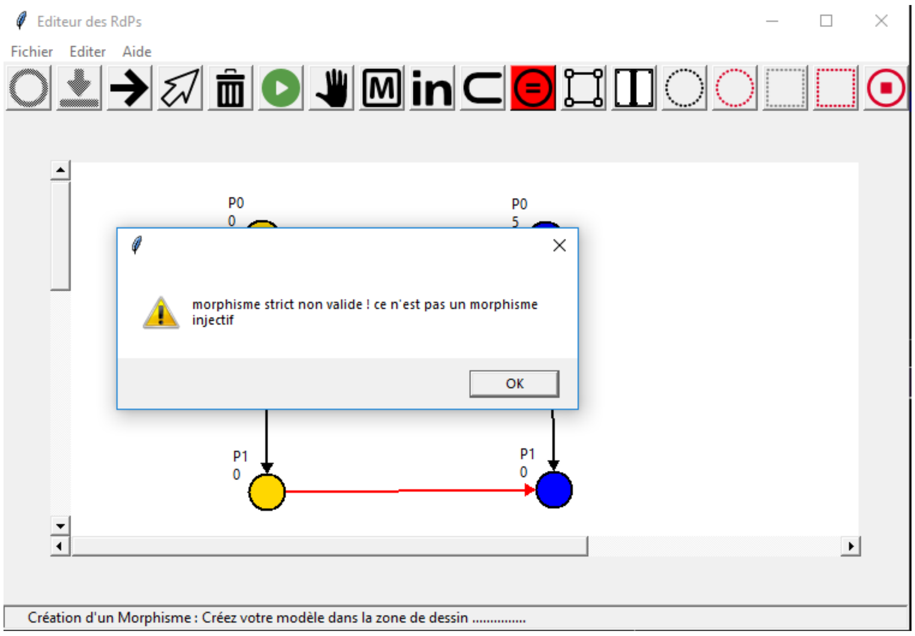


morphisme valide !

Création d'un Morphisme : Créez votre modèle dans la zone de dessin ....

Figure 4.8: Morphism editor(verifying morphism valid)

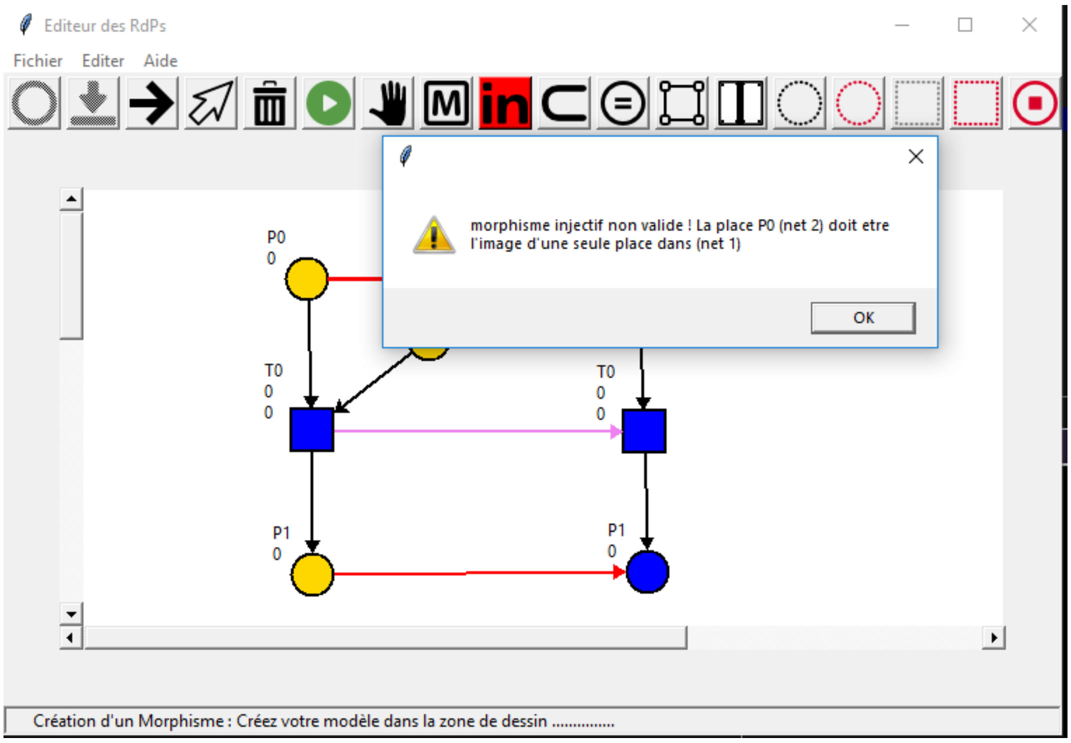
The same thing with The verification of morphism inclusif, strict or injectif. As shown in figure 4.9, 4.10, 4.11



morphisme strict non valide ! ce n'est pas un morphisme injectif

Création d'un Morphisme : Créez votre modèle dans la zone de dessin ....

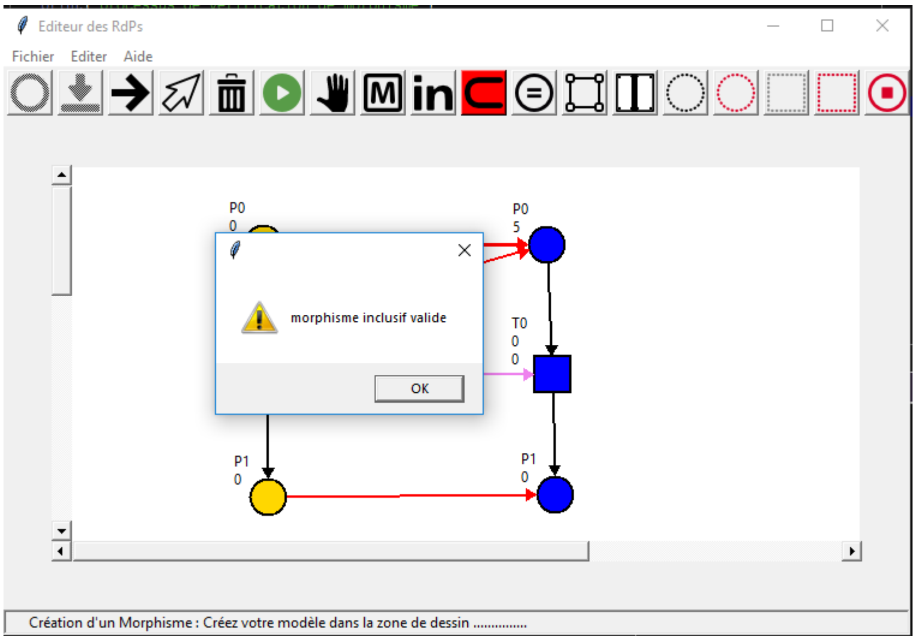
Figure 4.9: Verification if morphism strict



morphisme injectif non valide ! La place P0 (net 2) doit etre l'image d'une seule place dans (net 1)

Création d'un Morphisme : Créez votre modèle dans la zone de dessin ....

Figure 4.10: Verification if morphism injectif



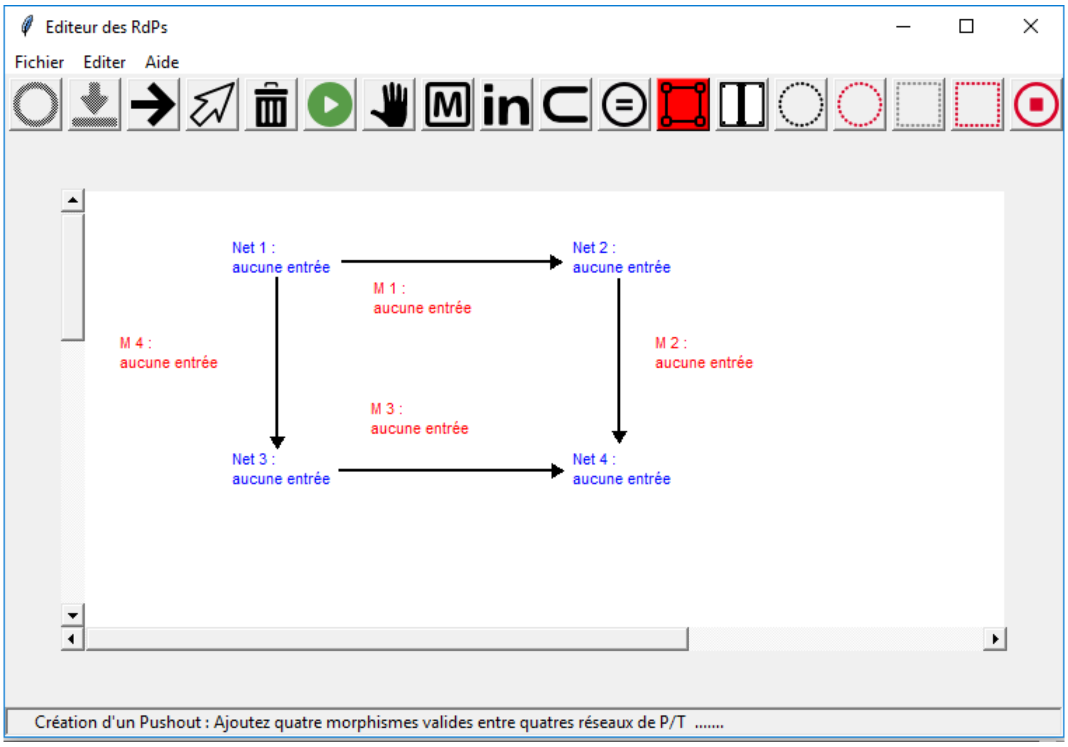
morphisme inclusif valide

Création d'un Morphisme : Créez votre modèle dans la zone de dessin ....

Figure 4.11: Verification if morphism inclusif

4.2.2.3 Pushout editor

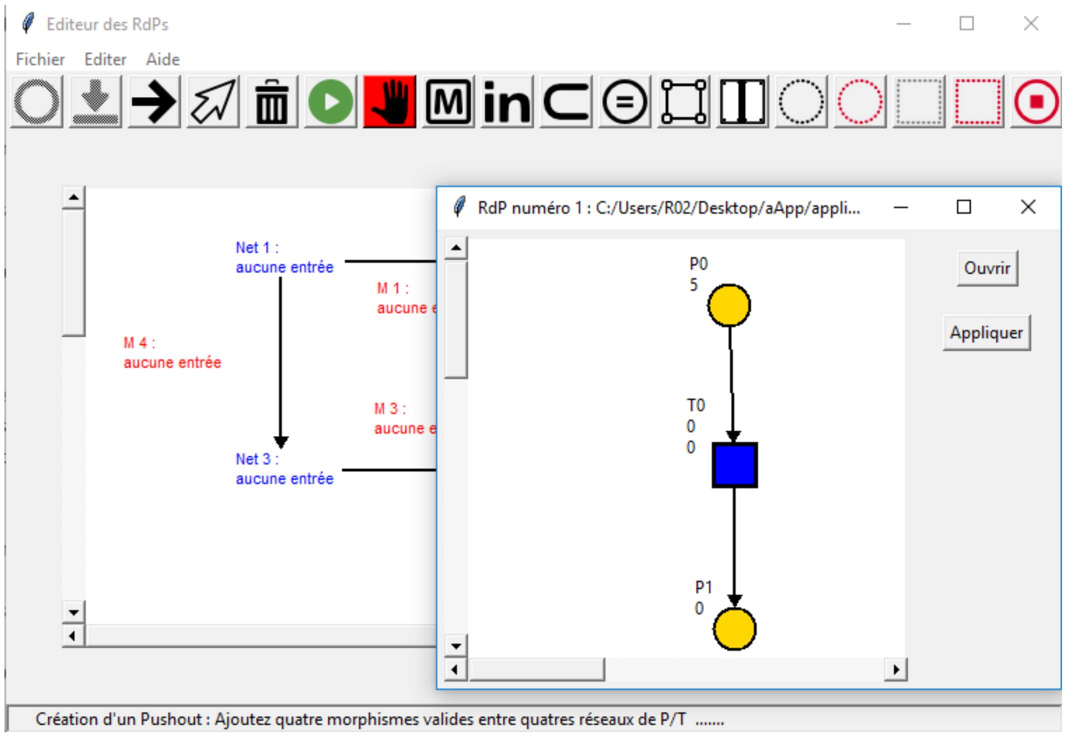
This tool allows the construction of morphisms to give a pushout then generate a POML code to export and re-import it in the tool.



Création d'un Pushout : Ajoutez quatre morphismes valides entre quatres réseaux de P/T ....

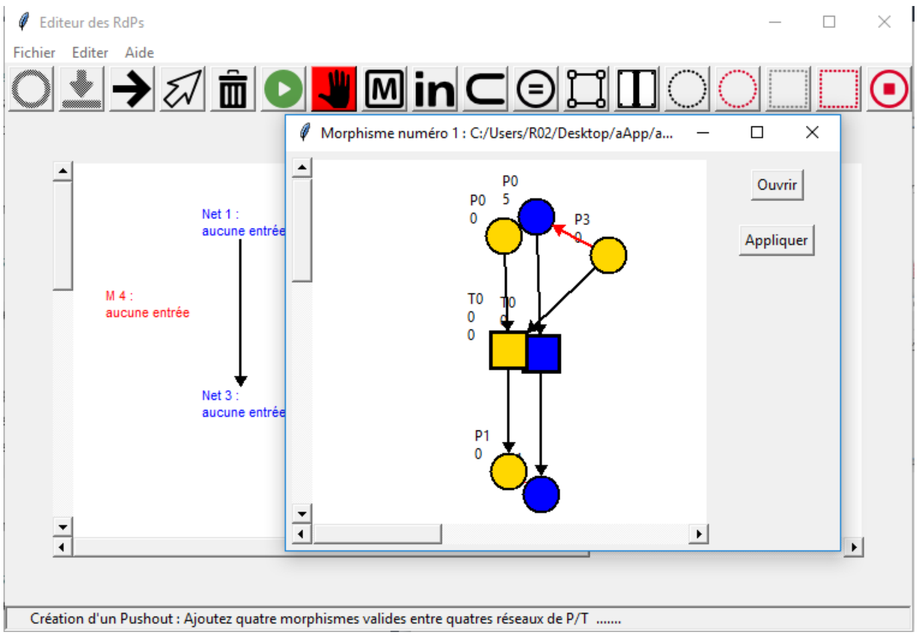
Figure 4.12: Pushout editor

Figure 4.12 represent the creation of a new pushout. The user have to enter four morphisms and P/T nets according to the schema by pressing in the blue text to add P/T net and in the red text to add a morphism and it will be shown in other window(see figures 4.13, 4.14)



Création d'un Pushout : Ajoutez quatre morphismes valides entre quatres réseaux de P/T ....

Figure 4.13: Pushout editor(enter the P/T nets)

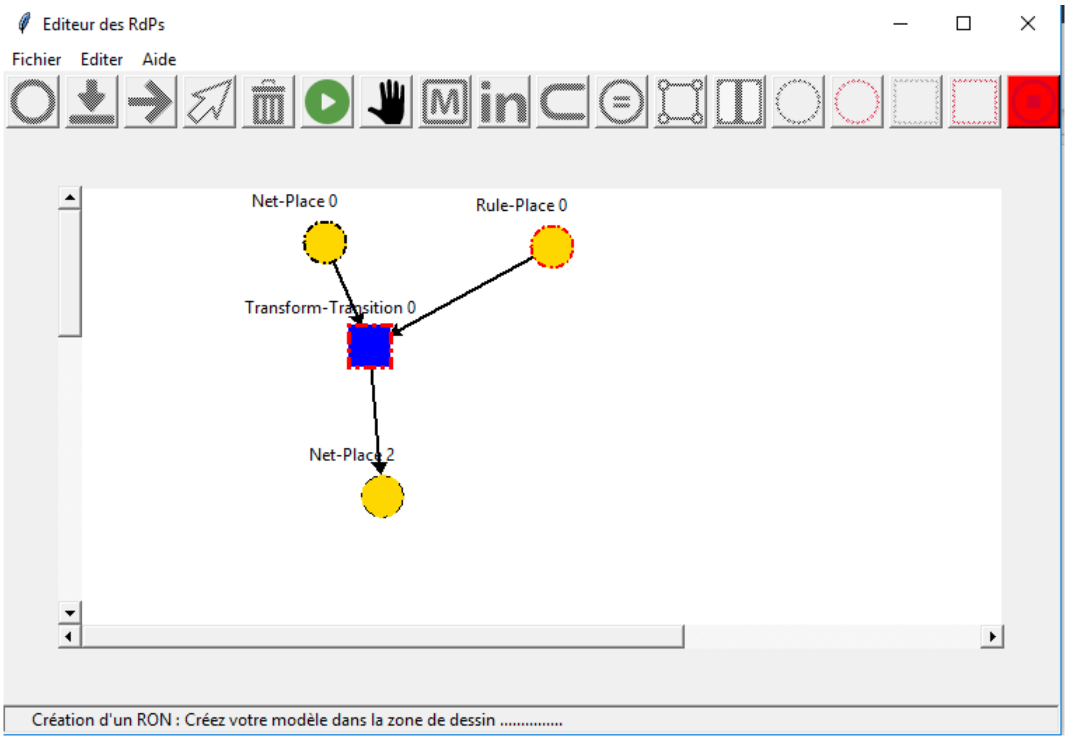


Création d'un Pushout : Ajoutez quatre morphismes valides entre quatres réseaux de P/T ....

Figure 4.14: Pushout editor(enter the morphisms)

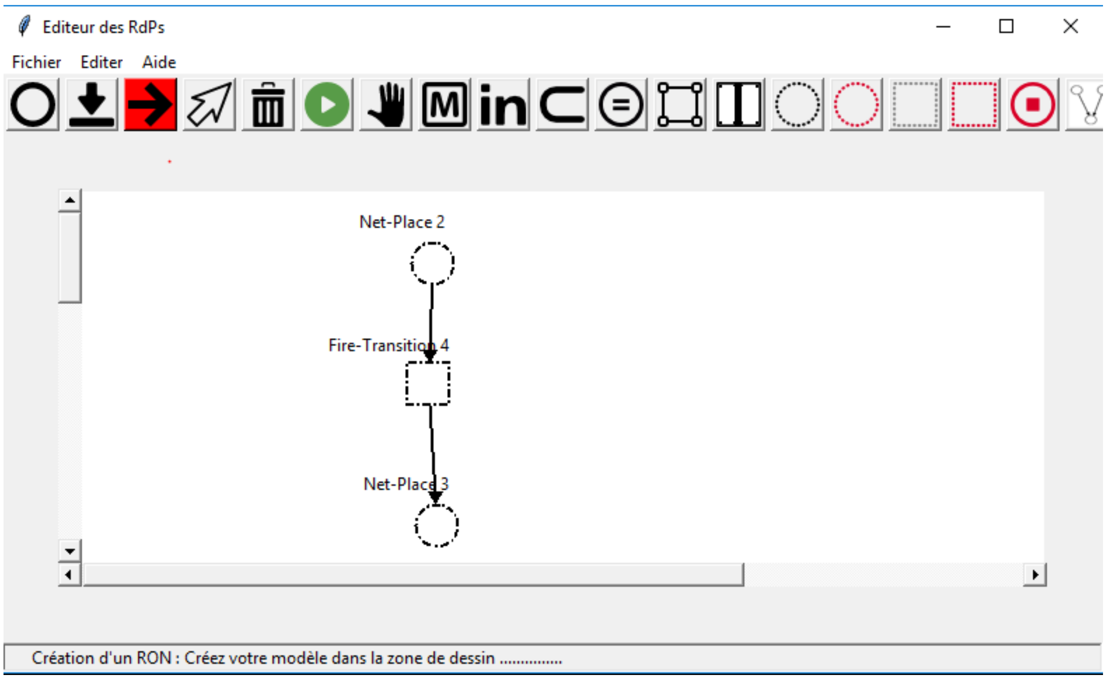
4.2.2.4 RON editor

The tool can also create a high level Petri net, it has two type of places: Net Place with black circle and Rule Place with red circle, and two types of transitions Fire transition with black rectangle and transform transition with red rectangle.



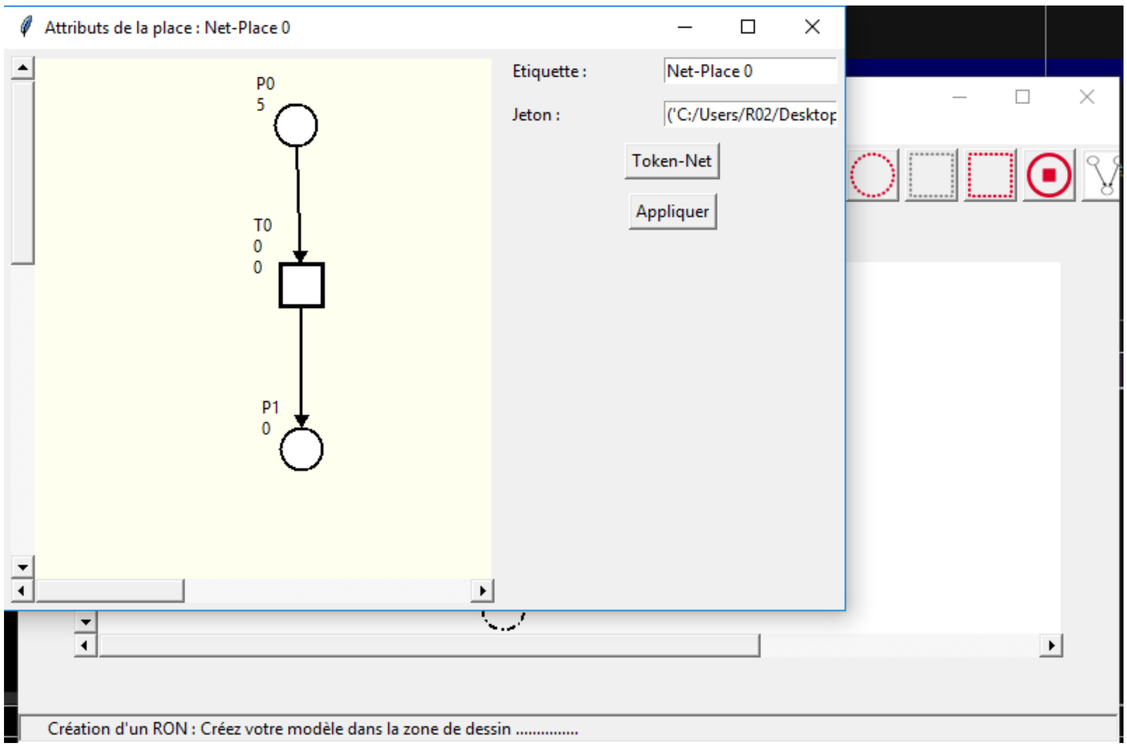
Création d'un RON : Créez votre modèle dans la zone de dessin ....

Figure 4.15: RON editor(transform transition)



Création d'un RON : Créez votre modèle dans la zone de dessin ....

Figure 4.16: RON editor(fire transition)

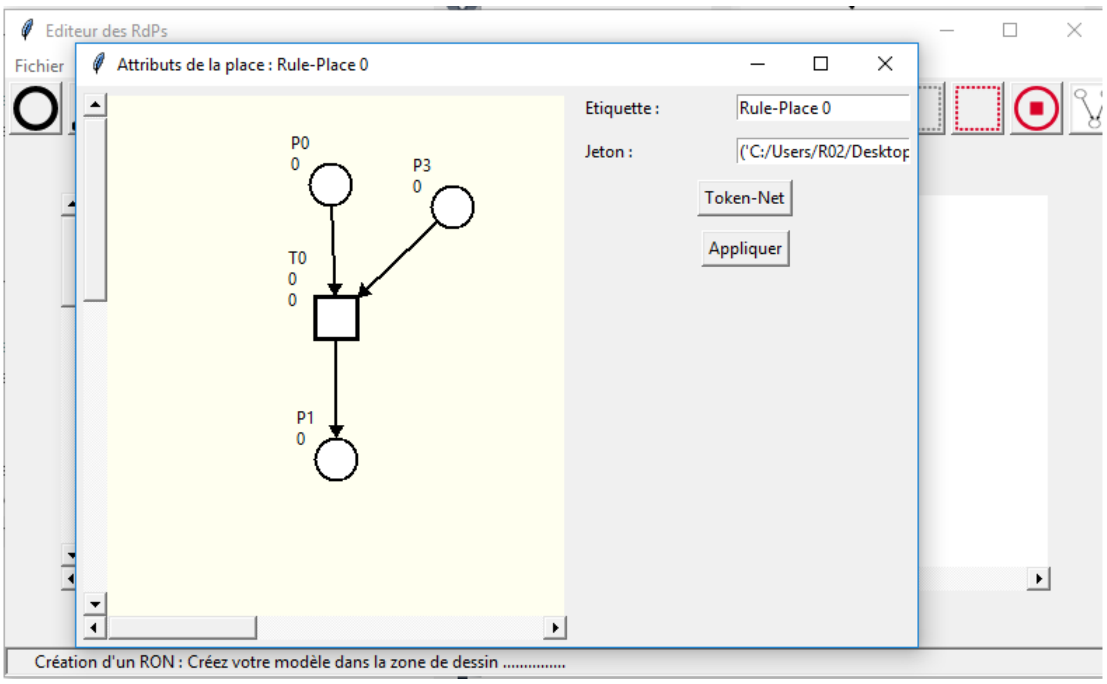


Etiquette : Net-Place 0

Jeton : ('C:/Users/R02/Desktop

Création d'un RON : Créez votre modèle dans la zone de dessin ....

Figure 4.17: RON editor(edit net place)



Etiquette : Rule-Place 0

Jeton : ('C:/Users/R02/Desktop

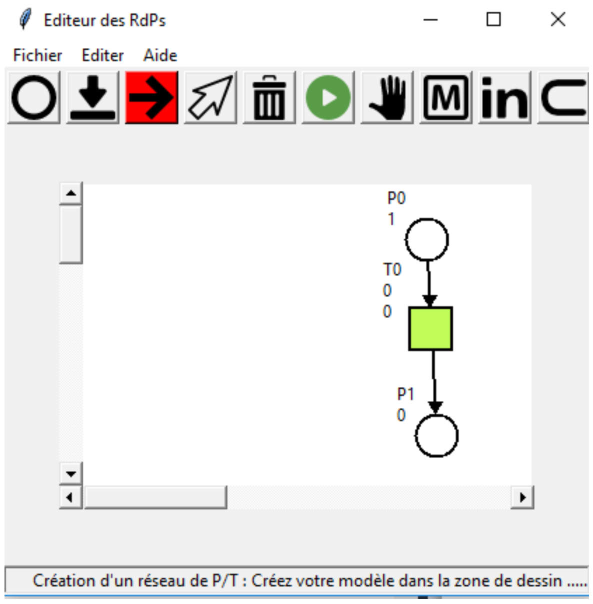
Création d'un RON : Créez votre modèle dans la zone de dessin ....

Figure 4.18: RON editor(edit rule place)

4.2.2.5 Reachability graph

The tool can also make the reachability graph of a Petri net where in contains of a set of nodes which represent the makring of each state and a set of arcs connect between graph nodes.

The reachability graph of the PN represented in Figure 4.19, 4.21 shown figures 4.20, 4.22respectively.



Création d'un réseau de P/T : Créez votre modèle dans la zone de dessin .....

Figure 4.19: PN editing with our tool

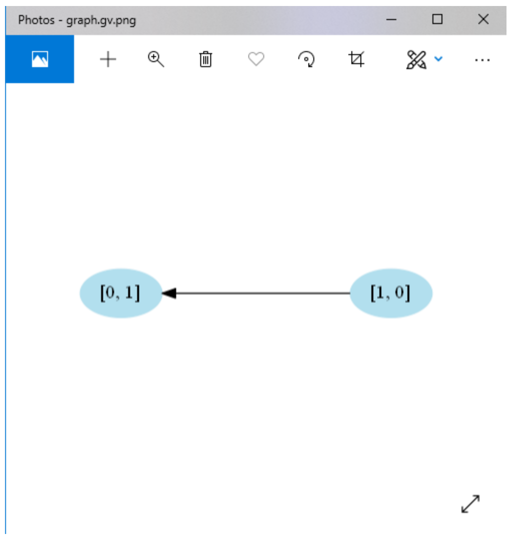
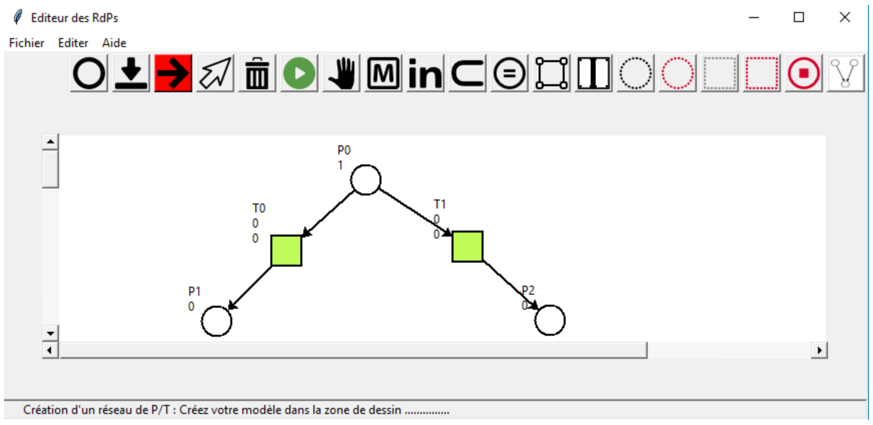


Figure 4.20: Reachability graph of PN in Figure 4.19



Création d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.21: PN editing with our tool

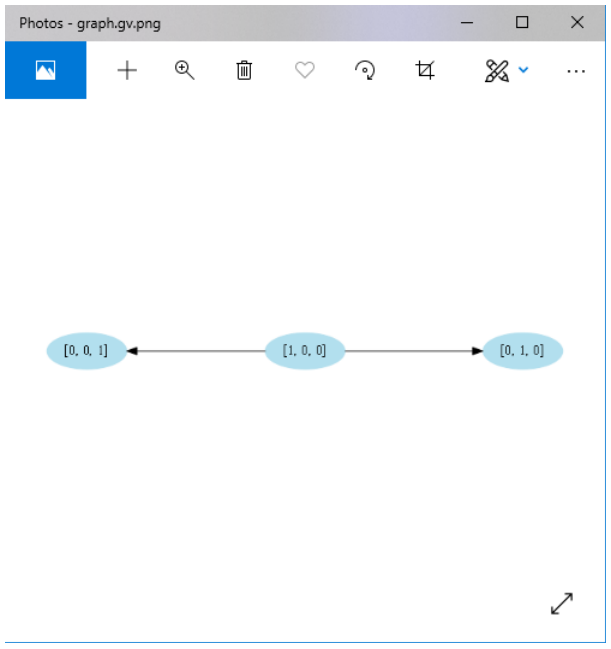


Figure 4.22: Reachability graph of PN represented in Figure 4.21

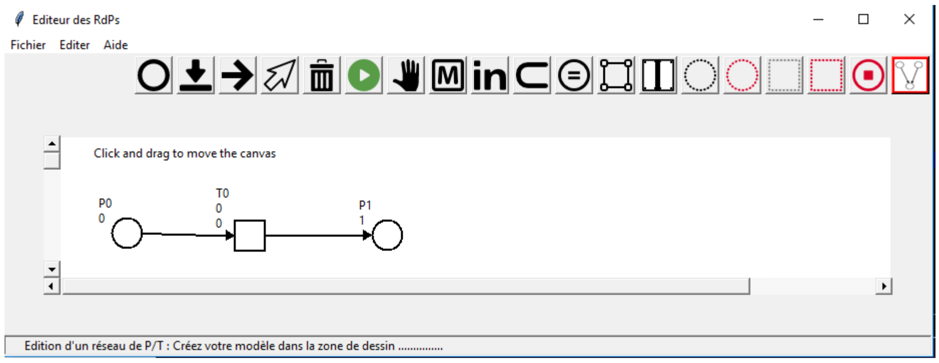

Editeur des RdPs
Fichier Editer Aide
Click and drag to move the canvas
P0
0
T0
0
0
P1
1
Edition d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.23: PN shown in Figure 4.19 after making reachability graph

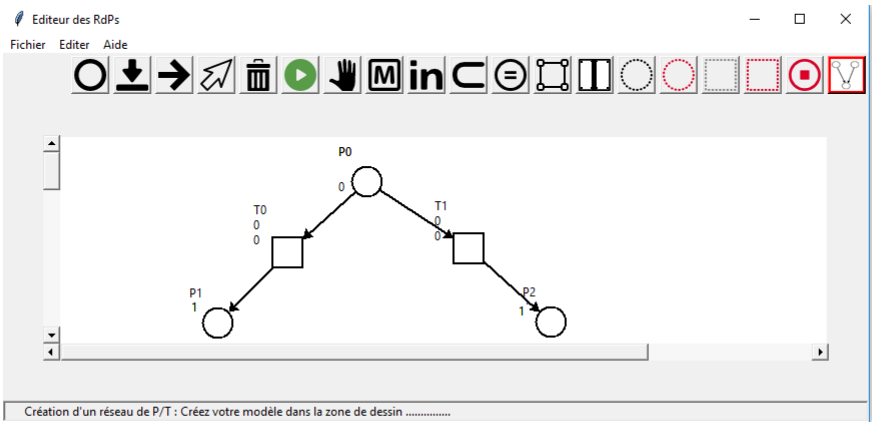

Editeur des RdPs
Fichier Editer Aide
P0
0
T0
0
0
T1
0
0
P1
1
P2
1
Création d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.24: PN shown in Figure4.21 after making reachability graph

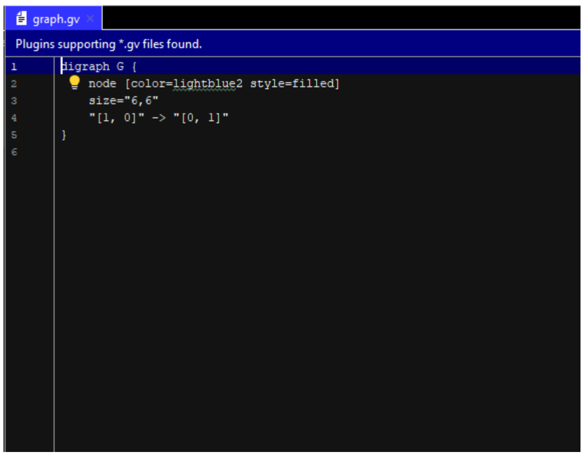
XML file of graph

After showing the reachability graph we can save in ih form xml.

```
 <?xml version="1.0" encoding="utf-8"?>
 <graph id="a751fda4-971e-11e9-9685-9457a5e6b569">
     <Node id="a751fda4-971e-11e9-9685-9457a5e6b569">
           <name>N0</name>
           <Marking state>[1, 0]</Marking state>
     </Node>
     <Node id="a751fda4-971e-11e9-9685-9457a5e6b569">
           <name>N1</name>
           <Marking state>[0, 1]</Marking state>
     </Node>
</graph>
```

Gv file of graph

After showing the reachability graph we can save it in gv form too.

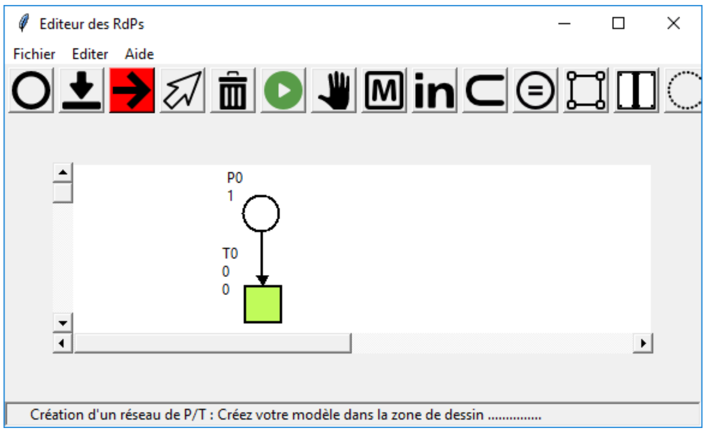


```
digraph G {
    node [color=lightblue2 style=filled]
    size="6,6"
    "[1, 0]" -> "[0, 1]"
}
```

Figure 4.25: gv file of reachability graph

4.2.2.6 Test

To test the analysis of P/T nets created with our tool, we start by modeling a P/T net (see figure 4.26).



Création d'un réseau de P/T : Créez votre modèle dans la zone de dessin ....

Figure 4.26: Petri net editing with our tool

The PN in previous figure 4.26 is exported as a file (.pnml) shown in the next xml files one with time and cost in transition and the other without.

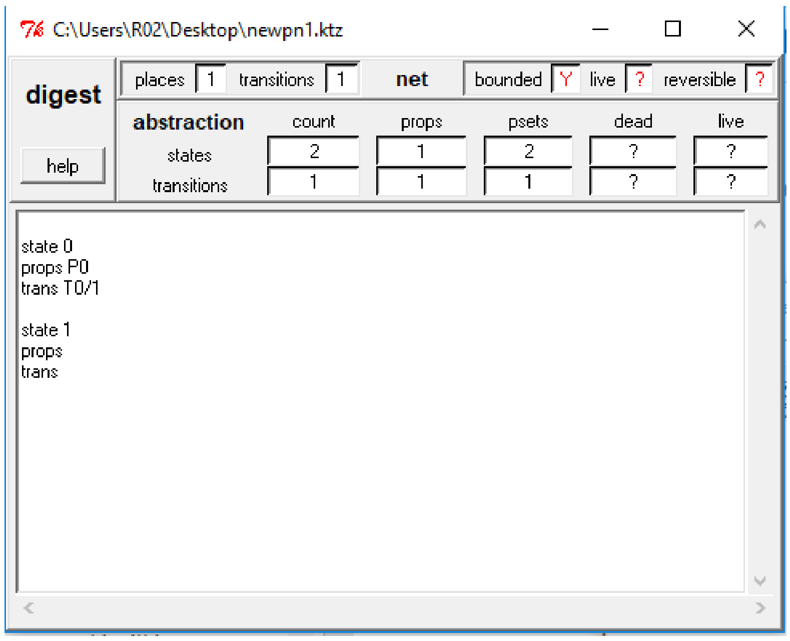
```
<?xml version="1.0" encoding="utf-8"?>
<pnml xmlns="http://www.pnml.org/version-2009/grammar/pnml">
   <net id="aa6fcade-9509-11e9-8e38-9457a5e6b569" type="http://
        pipe2.sourceforge.net/tpn">
        <name>
               <text>aa6fcade-9509-11e9-8e38-9457a5e6b569</text>
        </name>
        <page id="net_1">
            <place id="ad64eed8-9509-11e9-b8c0-9457a5e6b569">
                     <name>
                        <text>P0</text>
```

```
              <graphics>
                  <offset x="153" y="22"/>
              </graphics>
              </name>
              <initialMarking>
                   <text>1</text>
              </initialMarking>
              <graphics>
                <position x="173" y="42"/>
          </graphics>
        </place>
        <transition id="b1cb7870-9509-11e9-8bb9-9457a5e6b569">
                  <name>
                     <text>T0</text>
                   <graphics>
                     <offset x="155" y="98"/>
                   </graphics>
                   </name>
                   <graphics>
                         <position x="175" y="118"/>
                   </graphics>
                 </transition>
                 <arc id="b7744746-9509-11e9-aa90-9457a5e6b569"
                    source="ad64eed8-9509-11e9-b8c0-9457a5e6b569"
                    target="b1cb7870-9509-11e9-8bb9-9457a5e6b569"/>
     </page>
  </net>
</pnml>

<?xml version="1.0" ?>
<pnml xmlns="http://www.pnml.org/version-2009/grammar/pnml">
 <net id="aa6fcade-9509-11e9-8e38-9457a5e6b569" type="http://pipe2
     .sourceforge.net/tpn">
  <name>
    <text>aa6fcade-9509-11e9-8e38-9457a5e6b569</text>
  </name>
  <page id="net_1">
   <place id="ad64eed8-9509-11e9-b8c0-9457a5e6b569">
        <name>
          <text>P0</text>
          <graphics>
```

```
                <offset x="153" y="22"/>
          </graphics>
          </name>
          <initialMarking>
            <text>1</text>
          </initialMarking>
          <graphics>
                <position x="173" y="42"/>
          </graphics>
          </place>
          <transition id="b1cb7870-9509-11e9-8bb9-9457a5e6b569">
           <name>
                <text>T0</text>
                <graphics>
                   <offset x="155" y="98"/>
                </graphics>
           </name>
           <Time>
                 <text>0</text>
           </Time>
           <Cost>
                 <text>0</text>
           </Cost>
           <graphics>
                  <position x="175" y="118"/>
           </graphics>
           </transition>
           <arc id="b7744746-9509-11e9-aa90-9457a5e6b569" source="
              ad64eed8-9509-11e9-b8c0-9457a5e6b569" target="
              b1cb7870-9509-11e9-8bb9-9457a5e6b569"/>
        </page>
  </net>
</pnml>
```

The first (.pnml) file (without time and cost) is analyzed with Tina tool.


C:\Users\R02\Desktop\newpn1.ktz

digest

help

places 1 transitions 1 net bounded Y live ? reversible ?

abstraction	count	props	psets	dead	live
states	2	1	2	?	?
transitions	1	1	1	?	?

```
state 0
props P0
trans T0/1

state 1
props
trans
```

Figure 4.27: analyzing result

Conclusion

In this chapter we have presented we have presented the tools that we have used to release our application, then the results of our implementation as a set of graphical user interfaces (GUIs) and the realisation steps of a tool. This tool allow to create and edit graphically P/T nets, morphisms, pushout, double-pushout and Ron.Every module is created independently of the others.Each module has it's own structure to save it (pnml, poml, mml, dpml, roml). This work can be improved by adding properties analysis.

Conclusion

In the last few years there has been a growing interest in reconfigurable systems (RSs), and in order to be able to model this kind of systems, low level Petri nets have been developed and known many new extentions.High level Petri Nets supply the ability to design these systems and to analyse their properties with Reconfigurable Object Nets(RONs). RONs are Petri net with two kind of tokens:token net and token rule.

The global objective, of our work, is to build a formal approach that can be used to specify, simulate, and analyse reconfigurable systems. The approach uses the Reconfigurable Object Nets (RONs) formalisms [EHMTH05] as a formal model. For this purpose we have presented in this thesis a process of producing a tool for RONs.

During the project realisation, we have learned knowledges about:

1. High level Petri nets.
2. Exploitation of this knowledge to make a tool for RONs.
3. Python language programing.

As future work we intend to concentrate on addressing other questions which remain to resolve, some of which are:

- complete the tool by adding other functionalities like analysis of RONs.

Bibliography

[1] TEXMAKER (2019). *https://www.xm1math.net/texmaker/ on june 1 2019.*

[Bie08] Modica T. Biermann, E. " Independence analysis of firing and rule-based net transformations in reconfigurable object nets". *Electronic Communications of the EASST*, 10:1–13, 2008.

[CL09] C. Cassandras and S. Lafortune. " Introduction to Discrete Event Systems". *Springer Science Business Media*, 2009.

[EHMTH05] K. Ehrig H. Mossakowski T Hoffmann. High-level nets with nets and rules as tokens. *In International Conference on Application and Theory of Petri Nets. Springer, Berlin, Heidelberg*, pages 268–288, 2005.

[Kah16] Bourekkache S. Djouani K. Kahloul, L. " Designing reconfigurable manufacturing systems using reconfigurable object Petri nets". *International Journal of Computer Integrated Manufacturing,*, 29(8):889–906, 2016.

[KCD$^+$14] L Kahloul, A Chaoui, K Djouani, S Bourekkache, and O Kazar. " Using high level nets for the design of reconfigurable manufacturing systems". *in Z., Li and M., Khalgui, ed., 1st International Workshop on Petri Nets for Adaptive Discrete-Event Control Systems*, 1161:1–19, 2014.

[K.J00] Rozenberg K.Jensen, G. *Application and Theory of Petri Nets*. International series of monographs on physics. 2000.

[Kö18] Nolte D. Padberg J. Rensink A. König, B. "A Tutorial on Graph Transformation.In Graph Transformation, Specifications, and Nets ". *Springer, Cham.*, pages 83–104, 2018.

[Mar12] Hoffmann K. Gerhard O. Julia P Marvin, E. A tool for modeling and simulating with reconfigurable place/transition nets. *Hochschule fur Angewandte Wissenschaften Hamburg, Gemany*, 2012.

[Mor02] Yann Morère. " Cours de réseaux de Petri". 2002.

[Mur89] T Murata. " Petri nets: Properties, analysis and applications". *Proceedings of the IEEE*, 77(4):541–580, 1989.

[Pad08] Julia Padberg. " Petri Net Transformations". 2008.

[PBM06] K. Hoffmann P. Bottoni, F. De Rosa and M. Mecella. " Applying Algebraic Approaches for Modeling Workflows and their Transformations in Mobile Networks.Mobile Information Systems". 2(1):51—76, 2006.

[Rei85] W. Reisig. " Petri nets, An Introduction". *vol. 4 of EATCS: Monographs on Theoretical Computer Science. Springer-Verlag*, 1985.

[thpa] PIPE tool home page. http://pipe2.sourceforge.net. on 1 june 2019.

[thpb] TINA tool home page. http ://projects.laas.fr/tina. on 1 june 2019.

[VAL78] R. VALETTE. " Analysis of Petri Nets by Stepwise Refinements ". *Lnborntoire d'dutomatique et d'dnalyse des Systimes du Centre National de la Rechwche ,Scientifique,7, avenue du Colonel Roche, 31400 Toulouse, France*, Received August 22, 1977; revised May 26, 1978.

[Val04] R Valk. " Object Petri Nets Using the Nets-within-Nets Paradigm. In : Jrg Desel, Wolfgang Reisig, and G.R. (ed.) Advances in Petri Nets : Lectures on Concurrency and Petri Nets". *Springer-Verlag, Berlin, Hei- delberg, New York,USA*, 3098:819–848, 2004.

[Wol08] Petri Carl Adam; Reisig Wolfgang. "Petri net". *Scholarpedia*, 3(4):6477, 2008.