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: GLSD1/M2/2018

Report

Presented to obtain the diploma of academic Master in

Computer Science

Option: **Software Engineering and Distributed Systems**

A Tool for Modeling and Simulating with Reconfigurable Object Nets

---

By:
Ben Terki Aboubaker Seddiq

Defended the 24/06/2018, in front of the jury composed of:

Bennaoui Hammadi	MCA	President
Kahloul Laid	MCA	Supervisor
Zernadji Tarek	MCB	Examiner

University Year: 2017/2018

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: **Dr. Zernadji Tarek** and **Dr. Bennaoui Hammadi** for reading and evaluating my dissertation.

We want to thank **Pr. Claudia Ermel** Technische Universität Berlin, Germany for her guidance and for her endless patience and precious advice.

I never forget to thank all my teachers at Computer Science Department who taught me the basic principles of computer science.

Finally, my gratitude is deeply paid to my mother and my father, and to all members of my family.

Abstract

In this report, we present a tool for modelling and simulation with reconfigurable Petri nets. Applying the idea of algebraic graph transformations to object nets we obtain reconfigurable object nets whose net structure can be changed dynamically. The rule-based change of the net structure enables the adequate modelling of complex and dynamic structures as for example reconfigurable manufacturing systems. The tool we developed uses reconfigurable object nets that are extended by various annotations. The transformation approach is based on the well-known algebraic transformation approach as **DPO** (Double PushOut).

Résumé

Dans ce rapport, nous présentons un outil de modélisation et de simulation avec des réseaux de Petri reconfigurables. L'application de l'idée des transformations de graphes algébriques sur les réseaux de Petri a donné lieu aux réseaux d'objets reconfigurables dont la structure peut être changée dynamiquement. La modification de la structure du réseau basée sur des règles permet une modélisation adéquate de structures complexes et dynamiques, par exemple des systèmes de fabrication reconfigurables. L'outil que nous avons développé utilise des réseaux de Petri reconfigurable étendus par diverses annotations. L'approche de transformation est basée sur l'approche de transformation algébrique bien connue en tant que **DPO** (Double PushOut).

Contents

List of Tables **ix**

List of Figures **xi**

Introduction **xiii**

I Background 1

1 Petri Nets and Graph Grammar 3

Introduction 3

1.1 Petri Nets (Definitions) 3

1.1.1 Basic Petri net 4

1.1.2 Reachability tree 4

1.1.3 Reachability graph 4

1.1.4 Incidence matrix 5

1.2 Behaviour and Properties of Petri Nets 5

1.2.1 Properties of Petri Nets 5

1.2.2 Behaviour of nets and examples 6

1.3 Graph Grammars and Graph Transformation 8

1.3.1 What is Graph Transformation ? 9

1.3.2 Overview of Different Approaches 9

1.3.3 The Main Ideas of the Algebraic Graph Transformation Approach . 11

1.3.4 Graphs and Graph Morphisms 11

1.3.5 Graph Productions 13

1.3.6 Graph Transformation 13

Conclusion 14

2 Reconfigurable Petri Nets **16**
Introduction . . . 16
2.1 Correspondence of Notions between Petri Nets and Graph Grammars . . . 16
2.2 Reconfigurable Petri Nets . . . 18
2.2.1 Basic Concepts . . . 18
2.2.2 Definition (P/T Morphism) . . . 18
2.2.3 Definition (Gluing Condition) . . . 19
2.2.4 Definition (P/T System Rule) . . . 19
2.2.5 Definition (Reconfigurable P/T Systems) . . . 20
2.2.6 Types of Reconfigurable Petri Nets . . . 21
2.2.7 Application of Reconfigurable Petri Nets . . . 21
2.2.8 Tools . . . 22
2.3 Reconfigurable Object Nets . . . 22
2.3.1 Definition RON : . . . 22
Conclusion . . . 24

II A Tool for Reconfigurable Petri Nets **25**

3 Analysis and Design **27**
3.1 Introduction . . . 27
Introduction . . . 27
3.2 Analysis . . . 27
3.3 Global Design . . . 28
3.4 Class Diagram . . . 29
Conclusion . . . 30

4 Implementation **33**
Introduction . . . 33
4.1 Development Tools and Languages . . . 33
4.1.1 Python programming language . . . 33
4.1.2 PyCharm Programming Editor . . . 34
4.1.3 Tool Kit Interface “Tkinter” Package . . . 34
4.1.4 Document Preparation System LaTeX . . . 34
4.1.5 Typesetting Editor (TeX MAKER) . . . 34
4.2 Implementation . . . 34
4.2.1 Application Home . . . 35

4.2.2 Application features 37
Conclusion 51

Conclusion **xv**

Bibliography **xvii**

List of Tables

2.1 Correspondence of Notions 18

4.1 Software/Hardware versions 35

List of Figures

1.1 A Petri net with corresponding reachability graph and reachability tree . . . 5
1.2 A state machine and an event graph . . . 6
1.3 Contact in Petri Nets . . . 6
1.4 Backwards Conflict . . . 7
1.5 Forwards Conflict . . . 7
1.6 Causality in Petri Nets . . . 8
1.7 Concurrency in Petri Nets . . . 8
1.8 Rule-based modification of graphs . . . 9
1.9 pointfview . . . 10
1.10 DPO-Graph Transformation . . . 12
1.11 Directed Labeled Graph G . . . 12
1.12 Graph Morphism . . . 12
1.13 Graph Transformation with Pushouts (1) and (2) . . . 13
1.14 A Sample Graph Transformation . . . 14

2.1 Transition Firing as Token Game . . . 16
2.2 Transition Firing as Double Pushout . . . 17
2.3 Cyclic net with rules . . . 20
2.4 Application of rule r_1 to N . . . 20
2.5 Example Transform Transition . . . 23
2.6 Example Fire transition . . . 24

3.1 Global architecture of the application . . . 28
3.2 Class Diagram (P/T nets) . . . 29
3.3 Class Diagram (P/T nets) . . . 30

4.1 Application Home . . . 35
4.2 New Menu 1. . . . 36
4.3 New Menu 2. . . . 36

4.4 Toolbar 37
4.5 Module Petri net 38
4.6 Edit Transition 38
4.7 Edit Place 39
4.8 Pnml File 39
4.9 Petri Net using our Application 40
4.10 Petri Net using TINA 40
4.11 Run Simulation 41
4.12 Before and After Firing Transition T0. 41
4.13 Create Rule 42
4.14 RULE FILE 43
4.15 Check Morphism 1 43
4.16 Check Morphism 2 44
4.17 Check Morphism 3 44
4.18 Check Morphism 4 45
4.19 A RON model 46
4.20 ADD NETS 47
4.21 File RON 47
4.22 ADD RULES 48
4.23 Apply fire Transition 1 49
4.24 Apply fire Transition 2 49
4.25 Apply Transform Transition 1 50
4.26 Apply Transform Transition 2 50
4.27 Apply Transform Transition 3 51

Introduction

Today the aim of every field in engineering science is to use the formal modelling language to describe, analyze or diagnose the systems, in order to allow the reliability of systems. Software engineering is one of these sciences which have a big family of formal modelling language; Petri nets, automata, pi-calculus, etc. In this family, Petri nets have been developed to model and analyze the systems. In Petri nets there is one category that can describe the reconfiguration of systems called reconfigurable Petri nets nets. These last ones are based on the algebraic approach **DPO**[1] , developped in graph grammar and graph transformation.

The characteristic feature of reconfigurable Petri nets and their types (high level or low level), consisting of a Petri net and a set of rules that can modify it, is the possibility to discriminate between different levels of change. They provide powerful and intuitive formalism to model dynamic software or hardware systems that are executed in dynamic infrastructures. These infrastructures are dynamic since they are subject to change as well and since they support various applications that may share some resources. Such dynamic software or hardware systems have become increasingly more common but are difficult to handle. Modelling and simulating dynamic systems require a tool, for modelling, simulating and analyzing easily the reconfigurable Petri nets.

The aim of our project is to provide a tool for Modeling and Simulating with the Reconfigurable object nets ($\mathbf{RON}_s$).

This rapport starts with an introduction that presents the problem and a proposed solution, then it is composed of two parts, the first part focuses on the theoretic level (background), the second part shows our contribution in this work.

Part I is divided into two chapters. Chapter 1 describes and gives the basic concepts of Petri nets, graph grammar and graph transformation. Chapter 2 gives the relationship between graph grammar and Petri nets and gives an overview of reconfigurable Petri nets. The last section in this chapter gives a theoretical framework for Reconfigurable object Nets.

[1] Double-pushout

Part II concentrates on the development of our solution, it contains two chapters. Chapter 3 illustrates the design of our application in the two levels (the global and the detailed design), and chapter 4 is introduced as manual to describe the use of our tool.

The thesis ends with a conclusion that evaluates the results and discusses some future works.

Part I

Background

Chapter 1

Petri Nets and Graph Grammar

Chapter 1

Petri Nets and Graph Grammar

Introduction

In this chapter, we review some basic definitions describing Petri nets, graph grammar and graph transformation. We shall begin by defining basic Petri nets and their properties. After that we give some information about graph grammar. These will form the basis of the net transformation to be presented in the following chapter.

1.1 Petri Nets (Definitions)

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical representations of discrete distributed systems. As a modeling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with annotations. As such, a Petri net has place nodes, transition nodes, and directed arcs connecting places with transitions.Petri nets were invented in 1962 by Carl Adam Petri in his Ph.D thesis.

A Petri net consists of places, transitions, and directed arcs. Arcs link between places and transitions - not between places and places or transitions and transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition.

Places may contain any number of tokens. A distribution of tokens over the places of a net is called a marking. Transitions act on input tokens by a process known as firing. A transition is enabled if it can fire, i.e., there are tokens in every input place. When a transition fires, it consumes the tokens from its input places, performs some processing task, and places a specified number of tokens into each of its output places.[Mur89]

1.1.1 Basic Petri net

A Petri net is a five-tuple,$(B, E, {}^\bullet(-), (-)^\bullet, M_0), where$:

- B is the set of conditions of the net,
- E is the set of events of the net, disjoint from B,
- ${}^\bullet(-)$ is the precondition map,${}^\bullet(-) : E \to Pow(B)$,
- $(-)^\bullet$ is the postcondition map,$(-)^\bullet : E \to Pow(B), and$
- M_0 is the initial marking of the net, $M_0 \subseteq B$.

A marking is a set of conditions, representing the global state of a net by indicating which conditions hold. Thus, an initial marking corresponds to the set of conditions that hold in the initial state.

The set of preconditions of an event e is written as ${}^\bullet e$ and its set of postconditions is $e^\bullet$.

Its neighbourhood is the union of these two sets,

$$ {}^\bullet e^\bullet = {}^\bullet e \cup e^\bullet. $$

1.1.2 Reachability tree

The reachability tree of a Petri net (G, M_0) is a tree with nodes in $\mathbb{N}^{|P|}$ which is obtained as follows:
the initial marking M_0 is the root node of this tree; for each E enabled in M_0, the marking $\overline{M}$ obtained by firing E is a new node of the reachability tree; arcs connect nodes which are reachable from one to another in one step; this process is applied recursively from each such $\overline{M}$.

1.1.3 Reachability graph

The reachability graph is obtained from the reachability tree by merging all nodes corresponding to the same marking into a single node. Take as an example the Petri net depicted in Figure 1.1. The initial marking is (1, 1, 1, 1). Both transitions are enabled. If q_1 fires first, the next marking is (1, 1, 0, 2).If q_2 fires instead, the marking becomes (1, 1, 2, 0). From (1, 1, 0, 2), only the initial marking can be reached immediately by firing q_2 ; starting from (1, 1, 2, 0), only q_1 can fire, which also leads to the initial marking. Thus it has been shown that there are three different markings in the reachability graph of (1, 1, 1, 1).

Note : A basic net is safe if there is no contact in any reachable marking.

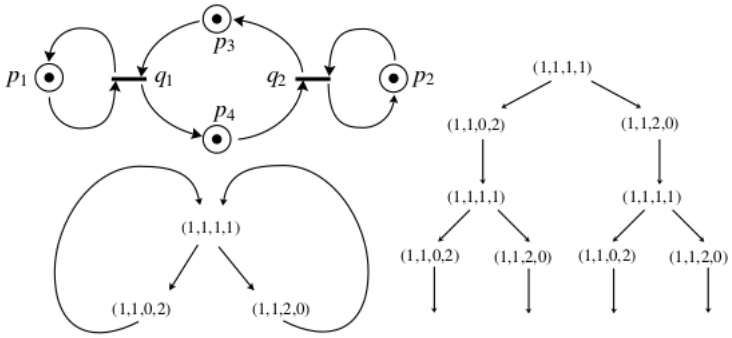


Figure 1.1 A Petri net with corresponding reachability graph and reachability tree

1.1.4 Incidence matrix

For a Petri net with n events and m conditions, the incidence matrix G = (G_{ij}) is an nm matrix of integers -1, 0 and +1. The entry G_{ij} is defined by

$$G_{ij} = G_{ij}^{out} - G_{ij}^{in}$$

where $G_{ij}^{out} = 1 resp.0$ if there is an (resp. no) arc from B_i to E_j and $G_{ij}^{in} = 1(resp.0)$ if there is an (resp. no) arc from B_j to E_i . Matrices G^{in} and G^{out} are defined as $G^{out} = (G_{ij}^{out})$ and $G^{in} = (G_{ij}^{in})$, respectively.

1.2 Behaviour and Properties of Petri Nets

In this section we introduce some Properties of Petri nets, for using later to analyse reconfigurable petri nets.

1.2.1 Properties of Petri Nets

Definition (Event graph and State machine) A Petri net is called an event graph if each place has exactly one upstream and one downstream transition.
A Petri net is called a state machine if each transition has exactly one upstream and one downstream place.

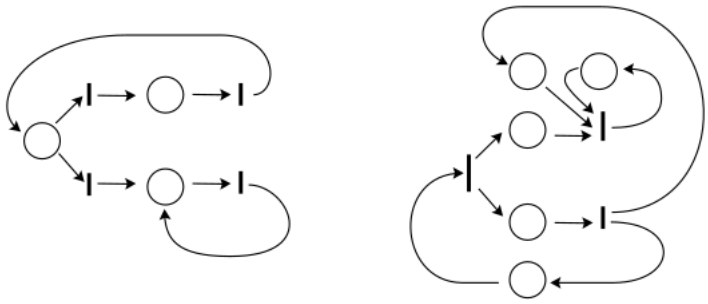


Figure 1.2 A state machine and an event graph

Bounded and Safe Nets A Petri net, with initial marking M_0, is said to be k-bounded if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0. Instead of 1-bounded Petri nets, one speaks of safe Petri nets.

Live net A Petri net is said to be live for the initial marking M_0 if for each marking $\overline{M}$ reachable from M_0 and for each transition $b \in B$, there exists a marking $\overline{M}_i$ which is reachable from $\overline{M}$ and such that b is enabled on $\overline{M}_i$. A Petri net which is not live is called deadlocked.

1.2.2 Behaviour of nets and examples

Through specifying events by their actions on local components of state, the behaviour of nets gives rise to several phenomena.

Contact In the following net, the event e_1 in the net N_1 cannot occur in the marking drawn due to contact: one of its postconditions still holds a token once its preconditions are unmarked. The event e_2 in the net N_2 does not have contact since the event can occur according to the definition.

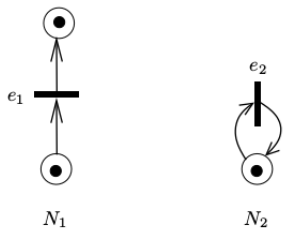


Figure 1.3 Contact in Petri Nets

Conflict Conflict describes how the occurrence of one event can inhibit the occurrence of another in a marking. There are two forms of conflict: forwards and backwards. Backwards conflict occurs where events compete to consume a condition. In the following net, only one of the events can occur from the marking drawn since the occurrence of one will consume the only token, causing the other not to have concession.

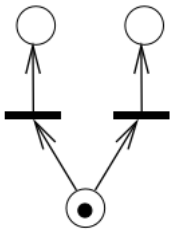


Figure 1.4 Backwards Conflict

Forwards conflict is where events compete to place a token in a condition - they compete on their post-conditions. In the following net, only one of the events can occur. The other will not be able to occur in the resulting marking due to contact.

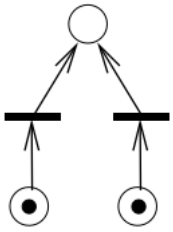


Figure 1.5 Forwards Conflict

Causality The causal relationships between the occurrences of events can also be extracted from nets. For example, in the net below the event e_2 can only occur once the event e_1 has occurred. This is due to the requirement that the condition b be marked for the occurrence of e_2, and b can only be marked through the occurrence of e_1.

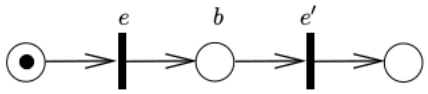


Figure 1.6 Causality in Petri Nets

Concurrency The final important concept in the behaviour of processes represented by nets is that independent events are allowed to occur concurrently. Independence of events is derived from their operation on non-overlapping sets of conditions. For events e and e' , we write eIe' defined as:

$$eIe' \Leftrightarrow {}^\bullet e^\bullet \bigcap {}^\bullet e'^\bullet = \emptyset.$$

For example, the events e and e' drawn below are independent:

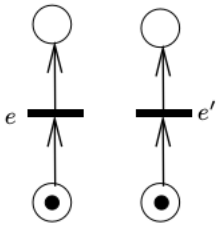


Figure 1.7 Concurrency in Petri Nets

1.3 Graph Grammars and Graph Transformation

Graph Grammar and Graph Transformation represent a discipline in computer science which dates back to the 1970s by Ehrig and the result of application has been study and applied in many fields of computer science. A detailed presentation of various graph grammar approaches and application areas of graph transformation is given in the three volumes of the Handbook of Graph Grammars and Computing by Graph Transformation[Roz97, EEHJG99, HHJUG99]. In this section we give the basic concepts of graph grammar and graph transformation and the algebraic approach of graph transformation double push out **DPO**; The latter will be the basis of Reconfigurable Petri Nets.

1.3.1 What is Graph Transformation?

There are many root to define graph transformation:

- from Chomsky grammars on strings to graph grammars;
- from term rewriting to graph rewriting;
- from textual description to visual modeling.

For giving a simple way to comprise the concept of graph grammars and graph rewriting, we use the notion of graph transformation. In any case, the main idea of graph transformation is the rule-based modification of graphs, as shown in Fig. 1.8.

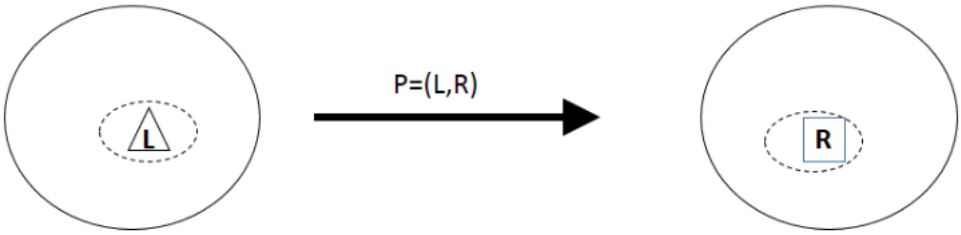


Figure 1.8 Rule-based modification of graphs

The core of a rule or production, p = (L, R) is a pair of graphs (L, R), called the left-hand side L and the right-hand side R. Applying the rule p = (L, R) means finding a match of L in the source graph and replacing L by R, leading to the target graph of the graph transformation. The main technical problems are how to delete L and how to connect R with the context in the target graph. In fact, there are several different solutions to how to handle these problems, leading to several different graph transformation approaches, which are summarized below.

1.3.2 Overview of Different Approaches

From an operational point of view, a graph transformation from G to H, written $G \Rightarrow H$, usually contains the following main steps, as shown in Fig.1.9:

1. Choose a production p : $L \Rightarrow R$ with a left-hand side L and a right-hand side R, and with an occurrence of L in G.
2. Check the application conditions of the production.

3. Remove from G that part of L which is not part of R. If edges dangle after deletion of L, either the production is not applied or the dangling edges are also deleted. The graph obtained is called D.

4. Glue the right-hand side R to the graph D at the part of L which still has an image in D. The part of R not coming from L is added disjointly to D. The resulting graph is E.

5. If the production p contains an additional embedding relation, then embed the right-hand side R further into the graph E according to this embedding relation. The end result is the graph H.

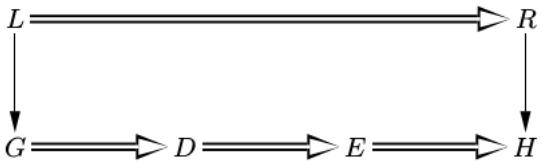


Figure 1.9 Graph transformation from an operational point of view

Graph transformation systems can show two kinds of nondeterminism:
first, several productions might be applicable and one of them is chosen arbitrarily; and second, given a certain production, several matches might be possible and one of them has to be chosen. There are techniques available to restrict both kinds of choice. Some kinds of control flow on rules can be defined for applying them in a certain order or by using explicit control constructs,priorities, layers, etc. Moreover, the choice of matches can be restricted by specifying partial matches using input parameters.

The main graph grammar and graph transformation approaches developed in the literature so far are presented in Volume 1 of the Handbook of Graph Grammars and Computing by Graph Transformation[Roz97]:

1. The node label replacement approach, developed mainly by Rozenberg,Engelfriet, and Janssens, allows a single node, as the left-hand side L,to be replaced by an arbitrary graph R. The connection of R with the context is determined by an embedding relation depending on node labels.For each removed dangling edge incident with the image of a node n in L, and each node n' in R, a new edge (with the same label) incident with n' is established provided that (n, n') belongs to the embedding relation.

2. The hyperedge replacement approach, developed mainly by Habel,Kre-owski, and Drewes, has as the left-hand side L a labeled hyperedge, which is replaced by an arbitrary hypergraph R with designated attachment nodes corresponding to the nodes

of L. The gluing of R to the context at the corresponding attachment nodes leads to the target graph without using an additional embedding relation.

3. The algebraic approaches are based on pushout and pullback constructions in the category of graphs, where pushouts are used to model the gluing of graphs. The double pushout approach, mainly developed by Ehrig, Schneider and the Berlin- and Pisa-groups.

4. The logical approach, developed mainly by Courcelle and Bouderon, allows graph transformation and graph properties to be expressed in monadic second-order logic.

5. The theory of 2-structures was initiated by Rozenberg and Ehrenfeucht, as a framework for the decomposition and transformation of graphs.

6. The programmed graph replacement approach of Schürr combines the glu- ing and embedding aspects of graph transformation. Furthermore, it uses programs in order to control the nondeterministic choice of rule applica tions.

1.3.3 The Main Ideas of the Algebraic Graph Transformation Approach

As mentioned above, the algebraic graph transformation approach is based on pushout constructions, where pushouts are used to model the gluing of graphs. In the algebraic approach, initiated by Ehrig, Pfender, and Schneider in [EMH73].The main idea is to model graph transformation by two gluing constructions for graphs and each gluing construction by a pushout. Roughly spoken, a production is given by p = (L, K, R), where L and R are the left and right hand side graphs and K is a common interface of L and R. Given a production p = (L, K, R) and a context graph D, which includes also the interface K, the source graph G of a graph transformation $G \Rightarrow H$ via p is given by the gluing of L and D via K, written G = L + $_kD$, and the target graph H by the gluing of R and D via K, written H = R + $_kD$. More precisely we will use graph morphisms $K \to L$, $K \to R$ and $K \to D$ to express how K is included in L, R, and D respectively. This allows to define the gluing constructions G = L +$_kD$ and H = R + $_kD$ as pushout constructions (1) and (2) leading to a double pushout in Figure 1.10.
Before we present the essentials of the Algebraic Approach of the DPO-approach

1.3.4 Graphs and Graph Morphisms

1. Let C = (C_A, C_N) be a pair of sets, called pair of color alphabets for arcs and nodes respectively,which will be fixed in the following.

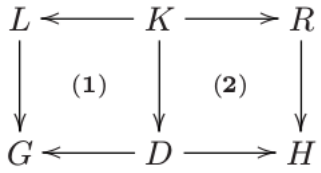


Figure 1.10 DPO-Graph Transformation

2. A (colored) graph G = (A , N , s , t , mA , mN) consists of sets A , N , called set of arcs and nodes respectively, and mappings s : A $\rightarrow$ N ,t : A $\rightarrow$ N,called source resp. target map, mA : A $\rightarrow C_A, mN : N \rightarrow C_N$, called arc resp. node coloring map. These data can be summarized in the diagram

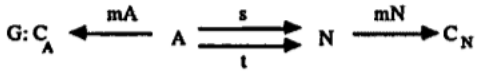


Figure 1.11 Directed Labeled Graph G

3. A graph G is called discrete if A_G is empty.

4. Graph G' is called subgraph of G if $A' \subseteq A, N' \subseteq N$ and all the mappings s', t', mA' and mN' are restrictions of the corresponding ones from G.

5. Given tow graphs G and G' a graph morphism $f : G \rightarrow G', forG \rightarrow G'$ for short is pair of maps $f = (f_A : A \rightarrow A', f_N : N \rightarrow N')$ such that $f_N \circ s = s' \circ f_A, f_N \circ t = t' \circ f_A, mA' \circ f_A = mAandmN' \circ f_N = mN$, i.e. the following diagram commutes for source and target mappings separately:

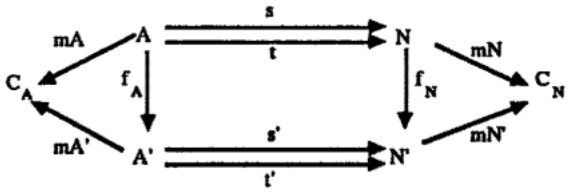


Figure 1.12 Graph Morphism

A graph morphism $f = (f_A, f_N)$is called injective resp. surjective if both f_A and f_N are injective resp. surjective mappings. If f : G –* G I $f : G \rightarrow G'$is injective and surjective it is called an isomorphism, and there is also an inverse isomorphism $f' : G' \rightarrow G$.

6. the composition $f' \circ f : G \rightarrow G$" of tow graph morphism $f = (f_A, f_N) : G \rightarrow G'$ and $f' = (f'_A, f'_N) : G' \rightarrow G$" is defined by $f' \circ f = (f'_A \circ f_A, f'_N \circ f_N)$.

7. Graphs and graph morphisms as above are defining a category in the sense of category theory, called the category of (colored) graphs.

1.3.5 Graph Productions

A (typed) graph production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ consists of (typed) graphs L, K, and R, called the left-hand side, gluing graph, and the right-hand side respectively, and two injective(typed) graph morphisms l and r.

Given a (typed) graph production p, the inverse production is defined by
$p^{-1} = (L \xleftarrow{l} K \xrightarrow{r} R)$

1.3.6 Graph Transformation

Given a (typed) graph production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a (typed) graph G with a (typed) graph morphism $m : L \to G$, called the match a direct (typed) transformation G $\overset{p,m}{\Rightarrow} H$ from G to a (typed) graph H is given by the following double-pushout(DPO)diagram , where (1) and (2) are pushouts in the category **Graph**.

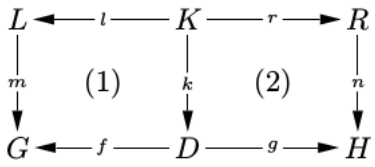


Figure 1.13 Graph Transformation with Pushouts (1) and (2)

A graph transformation as given in Figure 1.13 is denoted by $G \Rightarrow H$ via (p, m), where G is the source graph and H the target graph. In the next we will show that pushouts can be interpreted as gluing constructions. Given a production and a match $m : L \to G$ means that we require to be able to construct a context graph D such that G is the gluing of L and D along K in pushout (1) and H is the gluing of R and D along K in pushout (2) of Figure 1.13, written
$G = L +_K D$ and $H = R +_K D$.
The morphism $R \to H$in Figure 1.13 is called comatch of the graph transformation. A graph transformation sequence, also called derivation, is given by a finite sequence of graph transformations
$G_0 \Rightarrow$ via (p_0, m_0),$G_1 \Rightarrow$ via (p_1, m_1)....$G_n \Rightarrow$ via (p_n, m_n).
In general the construction of a graph transformation $G \Rightarrow H$ via (p,m) from a production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a match $m : L \to G$is given in two steps, where the first step requires that the gluing condition is satisfied:

STEP 1 (DELETE): Delete m(L -l(K)) from G leading to a context graph D (if the gluing condition is satisfied)[EMH73], s.t. G is the gluing of L and D along K, i.e. G = L + $_K D$ in (1) of Figure 1.13.

STEP 2 (ADD): Add R-r(K) to D leading to a graph H, s.t. H is the gluing of R and D along K, i.e. H = R + $_K D$ in (2) of Figure 1.13.

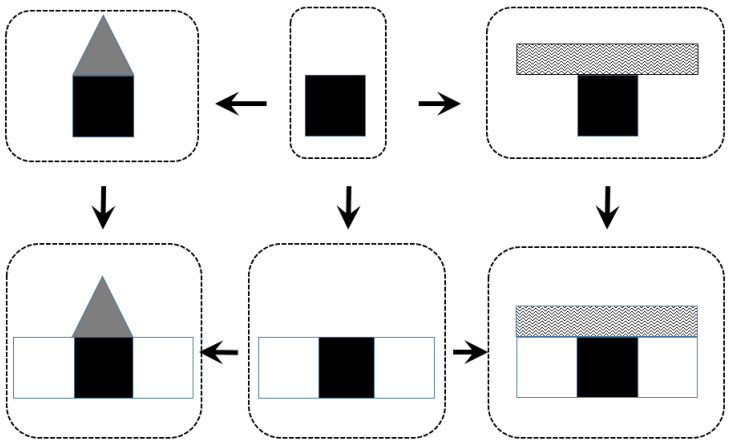


Figure 1.14 A Sample Graph Transformation

Conclusion

In this chapter, we have presented basic Petri net and we have given details about an algebraic approach in graph grammar and graph transformation called Double PushOut (DPO); these two disciplines in computer science Petri net and graph grammar using DPO as model of transformation, will be the foundation and aim of following chapter; "Reconfigurable Petri Nets".

Chapter 2

Reconfigurable Petri Nets

Chapter 2

Reconfigurable Petri Nets

Introduction

In the previous chapter, we gave the basic concepts of Petri nets, graph grammar, and graph transformation. Now we will give how to Petri net is suitable to use DPO from graph grammar and graph transformation. We begin by relationship between graph grammar and Petri nets. After that, we present an overview of reconfigurable Petri nets.

2.1 Correspondence of Notions between Petri Nets and Graph Grammars

The firing of a transition in a place-transition net can be modeled by a double pushout in the category of discrete graphs labeled over the places of the transitions. Let us consider the transition firing as token game in Figure 2.1.

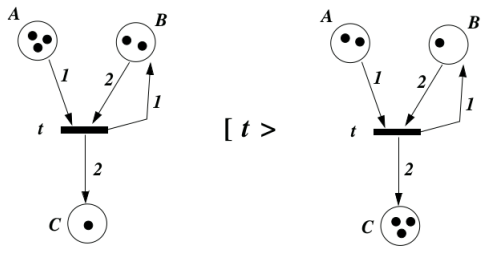


Figure 2.1 Transition Firing as Token Game

The transition t in Figure 2.1 requires in the pre-domain one token on place A and two tokens on place B and produces in the post-domain one token on B and two tokens

on place C. This corresponds to the production in the upper row of Figure 2.2, where the left hand side consists of three nodes labeled A, B and B and the right hand side of three nodes labeled B, C and C. The empty interface of the production means that no node is preserved by the production, which corresponds to the token game in place-transition nets. In fact, the transition t in Figure 2.1 consumes two tokens and produces one token on place B. Preservation of tokens in the framework of Petri nets can be modeled by contextual nets, and transition with context places can be modeled by productions with nonempty interface.

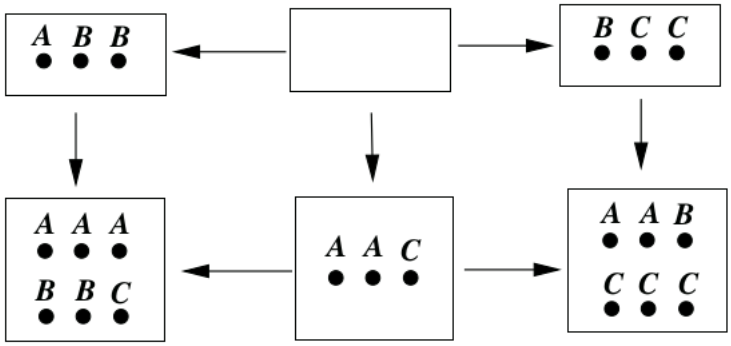


Figure 2.2 Transition Firing as Double Pushout

The marking of the left-hand side net in Figure 2.1 corresponds to the discrete graph to the left in the lower row of Figure 2.2, while the marking after firing of the transition in Figure 2.1 corresponds to the discrete graph to the right. The discrete graph in the middle of Figure 2.2 is the result of the deleting step of the double pushout and that on the right in the lower row is the result of the adding step. This shows that the firing step in Figure 2.1 corresponds exactly to a direct derivation in the double-pushout approach. This correspondence of notions between place/transition nets and graph grammars is shown in Table 1 in more detail. In fact, enabling of a transition at a marking corresponds to applicability of a production to a graph,concurrency of transitions corresponds to parallel independent productions applied with non-overlapping matches, conflicts correspond to parallel dependent direct derivations with overlapping matches, a parallel transition step of concurrent transitions corresponds to a parallel direct derivation, and finally a step sequence to a parallel derivation.

Petri Nets	Graph Grammars
tokens	nodes
places	node labels
marking	discrete, labeled graph
transition enabled at a marking	production applicable to a graph
firing	direct derivation
firing sequence concurrent transitions	derivation parallel independent productions
conflict	parallel dependence
step	parallel direct derivation
step	parallel direct derivation
step sequence	parallel derivation

Table 2.1 Correspondence of Notions

2.2 Reconfigurable Petri Nets

2.2.1 Basic Concepts

Definition 1 (Place/transition nets). place/transition nets is defined in [EP04] as:
A (marked place/transition) net is given by $N = (P, T, pre, post, cap, lab_p, lab_T, m)$ where P is a set of places, T is a set of transitions. pre :$T \to P^{\oplus}$ maps a transition to its pre-domain and post : $T \to P^{\oplus}$ maps it to its post-domain.Moreover cap :$P \to \mathbb{N}_{+}^{\omega}$ assigns to each place a capacity (either a natural number or infinityω), $lab_P : P \to A_P$is a label function mapping places to a name space, $lab_T : T \to A_T$ is a label function mapping transitions to a name space and $m \in P^{\oplus}$ is the marking denoted by a multiset of places.

2.2.2 Definition (P/T Morphism)

Given P/T systems $PS_i = (P_i, T_i, pre_i, post_i, label_i, M_i)$ for i = 1,2 a P/T morphism $f : PS_1 \to PS_2$ is given by $f = (f_P, f_T)$ with function $f_P : p_1 \to P_2$ and $f_T : T_1 \to T_2$ satisfying

1. $f_P^{\oplus} \circ pre_1 = pre_2 \circ f_T$ and $f_P^{\oplus} post_1 = post_2 \circ f_T$
2. $label_1 = label_2 \circ f_P$ and
3. $M_1(P) \leqslant M_2(f_P(P))$ for all $p \in P_1$ Moreover, the P/T morphism f is called strict if f_P and f_T are injective and $M_1(p) = M_2(f_P(P))$ for all $p \in P_1$

The category defined by P/T systems and P/T morphisms is denoted by **PTS** where the composition of P/T morphisms is defined component-wise for places and transitions. The

class of all strict P/T morphisms is denoted by $\mathscr{M}$.
Next we define the gluing condition which has to be satisfied in order to apply a rule at a given match. The characterization of specific points is a sufficient condition for the existence and uniqueness of the so-called pushout complement which is needed for the first step in a transformation.

2.2.3 Definition (Gluing Condition)

Given P/T systems $PS_i = (P_i, T_i, pre_i, post_i, label_i, M_i)$ $i \in \{L, K, 1\}$, and let $PS_L \xrightarrow{m} PS_1$ be a P/T morphism and $PS_K \xrightarrow{l} PS_L$ a strict morphism,then the gluing points GP, the dangling points DP and the identification points IP of PS L are defined by

GP = $l(P_K \cup T_K)$
DP = $\{P \in P_L | \exists t \in (T_1 \setminus m_T(T_L)) : m_P \in pre_1(t) \bigoplus post_1(t)\}$
IP = $\{P \in P_L | \exists p' \in P_L : P \neq p' \wedge m_P(P) = m_P(P')\}$
$\cup \{t \in T_L | \exists t' \in T_L : t \neq t' \wedge m_T(t) = m_T(t')\}$

The P/T morphisms m and l with l strict satisfy the gluing condition, if all dangling and identification points are gluing points, i.e DP $\cup$ IP $\subseteq$ GP, and m is strict on places to be deleted,i.e $\forall p \in P_L \setminus l(P_K) : M_L(P) = M_1(m(p))$. Next we present rule-based transformations of P/T systems following the double-pushout (DPO) approach of graph transformations in the sense of[Roz97, EEPT06].

2.2.4 Definition (P/T System Rule)

Given P/T systems $PS_i = (P_i, T_i, pre_i, post_i, label_i, M_i)$ $i \in \{L, K, R, 1\}$,then a rule $rule = (PS_L \xleftarrow{l} PS_K \xrightarrow{r} PS_R)$ consists of P/T systems PS_L, PSK ,andPS_R, called the left-hand side, interface, and right-hand side of rule, respectively, and two strict P/T morphisms $PS_K \xrightarrow{l} PS_LandPS_K \xrightarrow{r} PS_R$.

The rule rule is applicable at the match $PS_L \xrightarrow{m} PS_1$ if the gluing condition is satisfied for l and m.In this case, we obtain a P/T system PS_0 leading to a transformation step $PS_1 \xRightarrow{rule,m} PS_2$ consisting of the following pushout diagrams (1) and (2). The P/T morphism $n : PS_R \to PS_2$ is called comatch of the transformation step.

$$\begin{array}{ccccc} PS_L & \xleftarrow{l} & PS_K & \xrightarrow{r} & PS_R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle c} & (2) & \downarrow{\scriptstyle n} \\ PS_1 & \xleftarrow[l^*]{} & PS_0 & \xrightarrow[r^*]{} & PS_2 \end{array}$$

Now we are able to define reconfigurable P/T systems, which allow the modification of the net structure using rules and transformations of P/T systems.

2.2.5 Definition (Reconfigurable P/T Systems)

Given a P/T system **PS** and a set of rules **RULES**, a reconfigurable P/T system is defined by (**PS**,**RULES**).

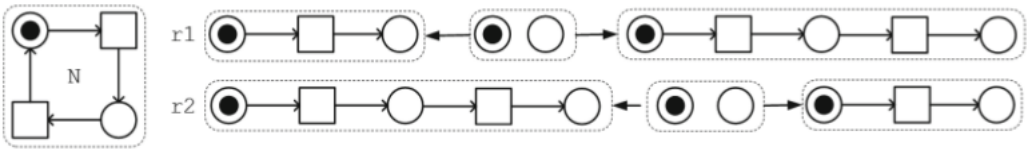


Figure 2.3 Cyclic net with rules

Example 1 (Modifying a cyclic process).[PK18] As an abstract example of a dynamic system we model a cyclic process that can either be executed or modified using the reconfigurable Petri net (N, r1, r2). Fig.2.3 depicts a simple place/transition net N and the rules r1 and r2. The net describes a cyclic process that executes one step and then returns to the start. The modifications in rule r1 change the process by inserting additional sequential steps. Rule r2 deletes an intermediate step. In Fig.2.4 the application of rule r1 to N is given. First a match of the left hand side of the rule is given by the occurrence morphism indicated by the light grey colour of the places and transitions in L and N. The gluing condition holds since the occurrence morphism preserves the token. In the first step the transition, which is coloured light grey, is deleted by the construction of the net D and in the second step the intermediate place and its adjacent transitions (coloured dark grey) are added.

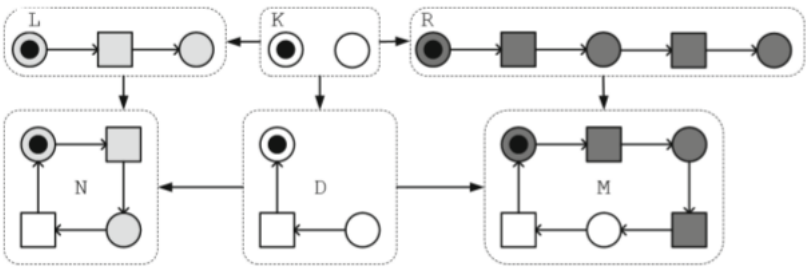


Figure 2.4 Application of rule r1 to N

2.2.6 Types of Reconfigurable Petri Nets

Reconfigurable petri net are divided in two family According to M-Adhesive Categories [PEHP08], It's summarized in the following.

- Low-Level
    1. P/T nets
    2. P/Ts with individual tokens
    3. decorated P/T nets
    4. P/T with inhibitor arcs and transition priorities
    5. timed P/T nets
- High-Level
    1. AHL[1] nets
    2. AHL nets with individual tokens
    3. AHO[2] nets
    4. AHO nets with individual tokens

2.2.7 Application of Reconfigurable Petri Nets

There are many application of reconfigurable petri net; we give some of these.

- emergency scenarios using mobile ad-hoc networks [EHP+07]
- reconfigurable manufacturing systems [KBD16]
- communication spaces (e.g. Skype, Apache Waves) [Gab14]
- ubiquitous computing (Living Space) [GNH12]
- hardware reconfiguration [PS16]
- Modeling Multicasting in Dynamic Communication-based Systems by Reconfigurable High-level Petri Nets[BEE+09]

[1] Algebraic high-level
[2] Algebraic higher-order

2.2.8 Tools

RON-Editor One of the tools concerned with reconfigurable Petri nets is RON-Editor [BEHM07]. The RON-editor is based on reconfigurable object nets [HEM05]. It is an open source and free tool [RON]. The RON-editor supports users to create, delete and edit parts of the model like object nets, net transformations rules and a top-level RON. The RON-editor makes several checks (e.g. for correct typing of tokens on RON places, to guarantee that mappings in rules satisfy net morphism properties) that help the user to obtain consistent RONs. Additionally, the editor comprises a simulator using the AGG engine to simulate the application of rules and thus firing of high-level transitions in the RONs created with the editors. The set of visual editors have been realized as Eclipse plug-ins using the Eclipse Modelling Framework (EMF) and Graphical Editor Framework (GEF) plug-ins.

ReConNet [PEOH12, ReC] is an open source project that has been initiated at the HAW Hamburg developing a tool for editing and simulating reconfigurable decorated nets. Itprovides an intuitive graphic-based user interface that allows the user to create, modify and simulate reconfigurable nets.

2.3 Reconfigurable Object Nets

Are considered reconfigurable petri; one of the formal modeling language using to describe the systems wich change their structure at the run-time, but this language is dedicated to modeling one system not plus. In fact there are reconfigurble distribute system. This requires a new model for description this systems. we introduce a new generation of reconfigurable petri nets. this generation is called reconfigurable object nets on used the paradigm of token as net introduce by valk [Val04]. In this section, we give the basic concepts of our contribution.

2.3.1 Definition RON:

Reconfigurable object nets (RON) represent an extension from reconfigurable Petri net, use to describe reconfigurable distributes systemes, multi-agent systems ad-hoc network. RONs use the paradigm of token as net from **valk** [Val04] to describe the distribution of the nets and tokens.The **DPO** approach from graph grammar and graph transformation (see chapter 1) allows to describe a set of productions rules and for changing nets structure.

RON represent high level nets with two types of places net-places , rule-places and two types of transition firing-transition and transform-transition, directed arcs connecting places

with transitions. Net-places is the buffer of basic reconfigurable Petri nets, Rule-places mean the set of production rules.

First off a High level places are typed either as NET or RULE and thus can either hold object nets or rules and do not have a capacity. Also the arcs have no weights in contrast to our regular Petri nets. There are two types of high-level transition:

Transform-Transition:

The pre-condition to be enabled. is one net-places and one of rule-places. The post-condition is a set net-places. The applicaiton of this transition makes changes in the net structure. Figure 2.5 shows that the application of transform transition required the presence of least one token object in precondition and at least one rule. After the transformation, the net from net places change its structure; but in rules places there is no changing in the marking or in the structure of the rules.

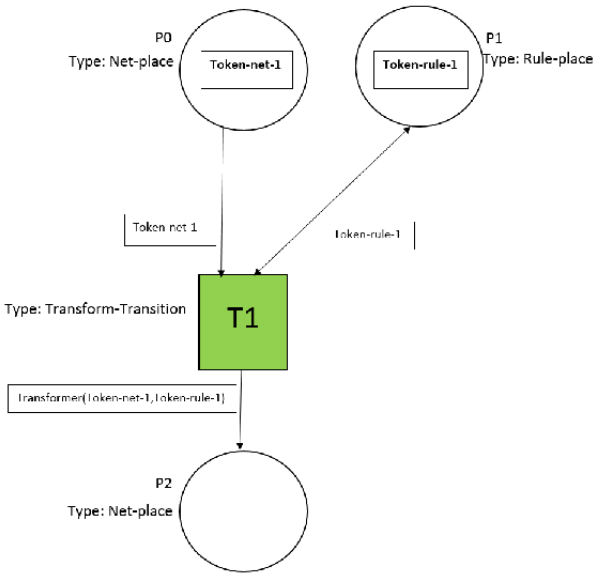


Figure 2.5 Example Transform Transition

Fire-Transition:

This transition is like a transition in basic petri nets with some different; it transition allows an object from net places to change their marking and shifting the object to another net-places, the figure 2.6 depicts the application of fire transition.

From figure 2.6, the fire transition have no rule place in the entry.

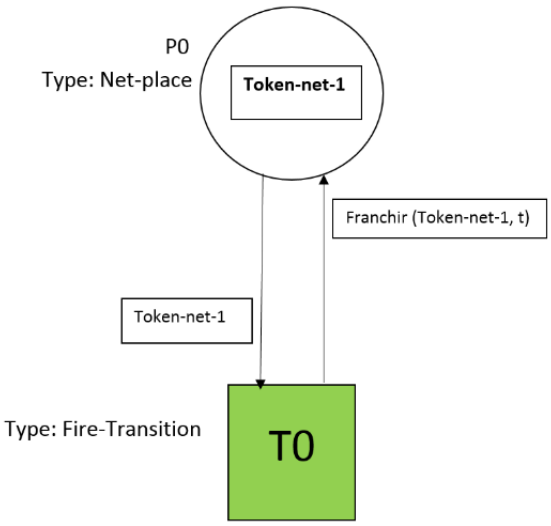


Figure 2.6 Example Fire transition

Conclusion

In this chapter we have presented a Petri net transformations as generalization of graph grammar and graph transformation and we presented this as reconfigurable Petri Nets. In the end of this and after giving the detail about reconfigurbale Petri Nets, we have introduced another extension for reconfigurable Petri nets called Reconfigurable Object Net. In the next chapter we give the realization a tool proposed for the editing and the simulation of *RONs*.

Part II

A Tool for Reconfigurable Petri Nets

Chapter 3

Analysis and Design

Chapter 3

Analysis and Design

3.1 Introduction

After having learned the necessary theoretical points, we pass to the application development. This chapter is composed of three sections; we begin by present the aim of our project. Next, we give an abstract model of the application, application. Finally, in last, we present the details of the general architecture, which was mentioned in the previous section; we use UML **UML**[1] class diagram to do that.

3.2 Analysis

Nowadays, formal modeling researchers look to introduce new model more flexible and easy than old ones.One of these models is called reconfigurable Petri nets. However modeling still difficult using these models and the tools dedicated to these kind of formalisms are not suitable neither sufficient. Our project aim is the implementation of new software solutions more easy than other tools and we take the aspect of tokens as nets [Val04], to introduce reconfigurable object nets (RON) but not like the RON developed in -berlin 2 [2][BEHM07].

[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.

[2]Technical University Berlin www.tfs.cs.tu-berlin.de/roneditor

3.3 Global Design

After having fixed the project aim and before entering in the process of development or the programming, it is necessary to present the results of analysis of our topic as an abstract architecture. Figure 3.1 represents our abstract model.

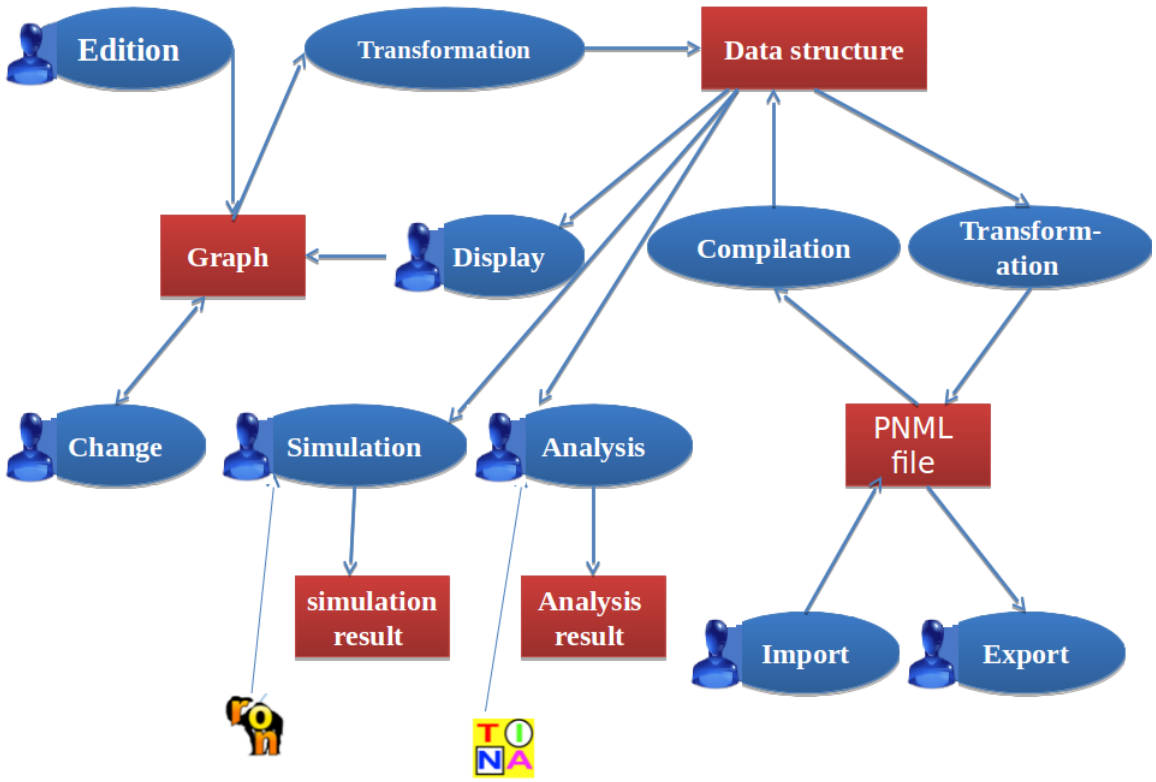


Figure 3.1 Global architecture of the application

According to our global architecture the user can create 3 types of models: petri net, rule and Ron[3] which are stored in files with the following extensions: pnml[4], Rule and Ron Respectively. These files are created After the user draws their model, he also can read these files with the using of our application in the simulation of petri net or RON or in checking the validity of the morphisme. The user maybe can use these files (PNML, RULE and RON) in external tools like Tina (see [Tin]) or ReConNet (see [ReC]) as analyzer to analyze their petri nets models.

[3]Reconfigurable Object Nets
[4]Petri Nets Markup Language

3.4 Class Diagram

The above model (figure 3.1) is the abstraction of our application but we need a model with some details to describe the tools in a way closer to the programming level. The best model is **UML** class diagram.

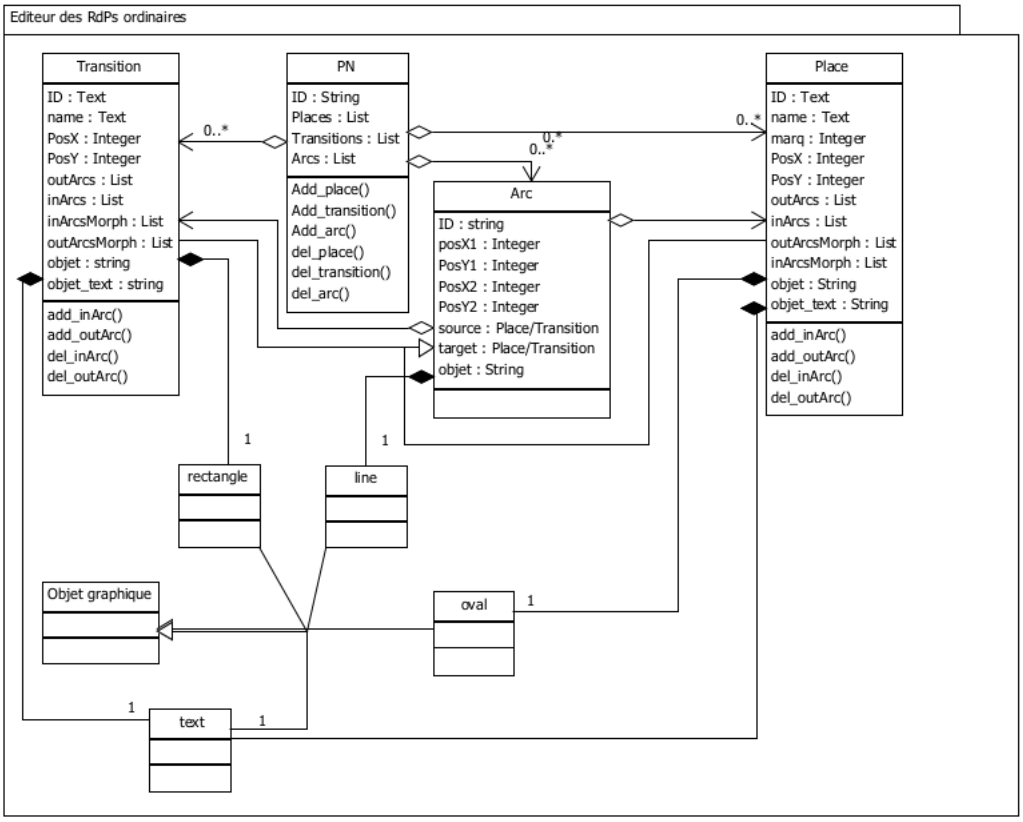


Figure 3.2 class diagram (P/T nets)

The (Figure 3.2) shows the main classes that are needed to make the addition of low-level Petri nets (PN) (see Chapter 1.1). A PN class represents a P/T network; instances of this class that we will use later as Object Nets in RONs. The PN class contains all the necessary attributes to define a P/T network: a list of Place objects, a list of Transition objects, a list of Arc objects, including the necessary methods for the addition and removal of these elements. The Transition class contains the graphical positions PosX and PosY which indicate where the graphics object is drawn, a reference on a rectangle object (a graphical object that represents a transition), a list of incoming arcs, a list of outgoing arcs and the necessary methods to add and delete. The same attributes for the Place class except that the place is associated with a graphical object of the oval class and that class contains another attribute that indicates the marking. The Arc class, contains a source object (Place or Transition), a target object (place

or transition), graphical positions of the line object (graphic object that represents an arc) and a reference to this graphics object (line).

To give a basic understanding of how the RON was implemented, consider the (partial) model that shown in Figure 3.3.

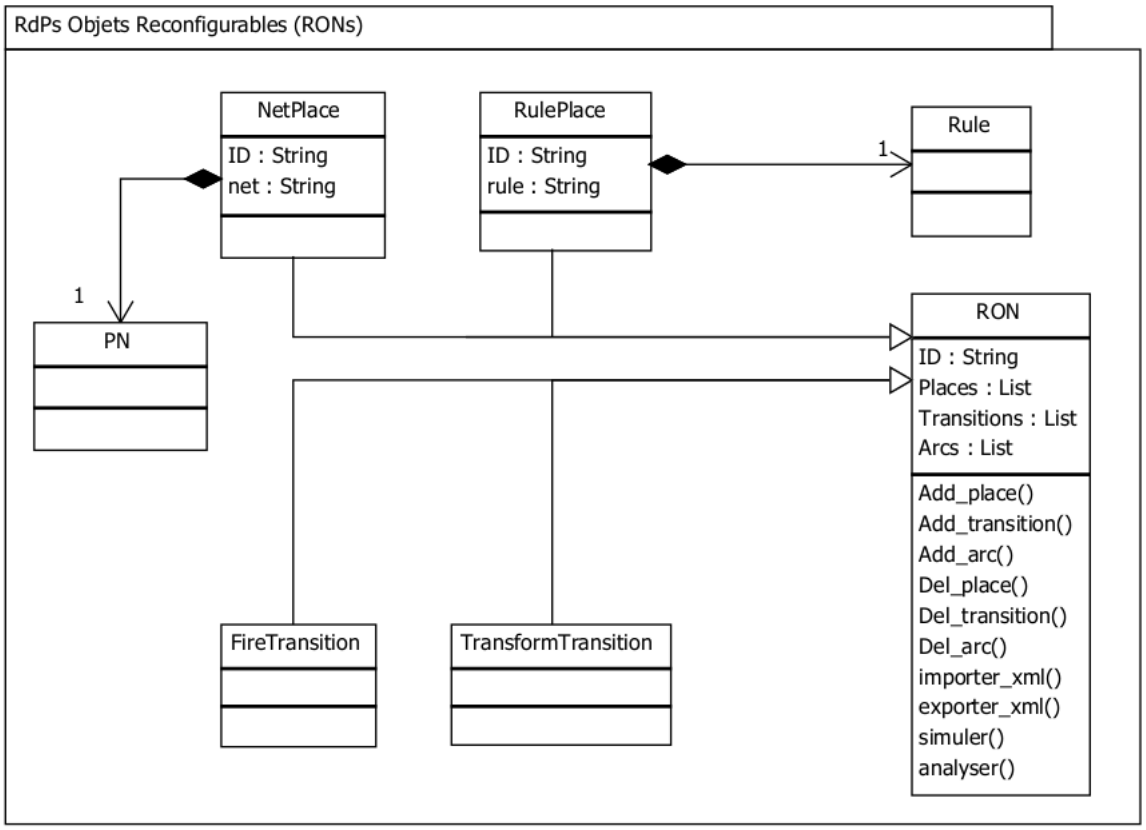


Figure 3.3 class diagram RON$_s$

Figure 3.3 shows the main classes needed for the creation of a RON. The RON class contains a list of places that can be objects of NetPlace or classes, a list of transitions that can be "Transform Transition" or "Fire Transition" classes, a list of arcs and necessary methods for adding, deleting, importing, exporting, simulation and analysis. The class "Netplace" contains a list of the objects of the class PN (Token Nets). class contains a list of objects in the class Rule (Token Rules).

Conclusion

In this chapter, we have shown analysis that has directed us to a design our tool. We move on to the next stage in which we implement this project. 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 chapter 4 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 tow modules. The first module introduces briefly the development tools and languages that we have learned and exploited in the realization of our project. The second section is 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 realization 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 is not 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*" [tk05], 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 Document Preparation System LaTeX

LaTeX [Lam17] 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 includes 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.

4.1.5 Typesetting Editor (TeX MAKER)

TeX MAKER [TMa17] 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 4.1), we used in our tool; oriented object programming (OOP) paradigm, and we used a set of software and hardware which are summarized in the following table. 4.1.

Software/Hardware	Version
OS	linux ubuntu , 64bits, version 16.04
CPU	Intel®Pentium(R) CPU P6200 @ 2.13GHz × 2
RAM	4.00Go
Python Interpreter	3.5.0
PyCharm	2016.3.1
Tkinter	8.6

Table 4.1 Software/Hardware versions

4.2.1 Application Home

Figure 4.1 depicts the whole interface of our application.

Figure 4.1 Application Home

After many steps of development, we implemented a graphical tool that allows to model and simulate;The Reconfigurable object nets based algebraic approach-DPO (see chapter 1,2). This tool is not only for RONs[1] but it also allows modeling and simulation of Low level Petri nets (net P / T). The tool has been designed to be easy to use with a simple interface (see Figure 4.1) which consists of a menu bar, a toolbar and a drawing area with scroll-bars, and a status bar. A menu bar contains the different editors that can work separately one of the other. Next some description about the components of our tools; menu bar, tool bar and drawing area, are presented.

[1] Reconfigurable Object Nets

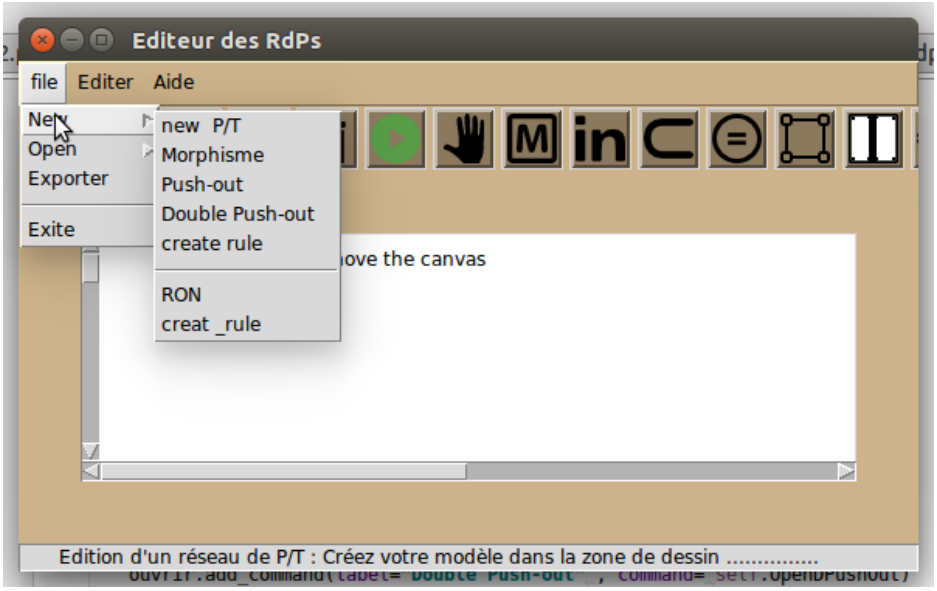


Figure 4.2 New Menu 1.

(a) File Menu. (b) Open Menu.

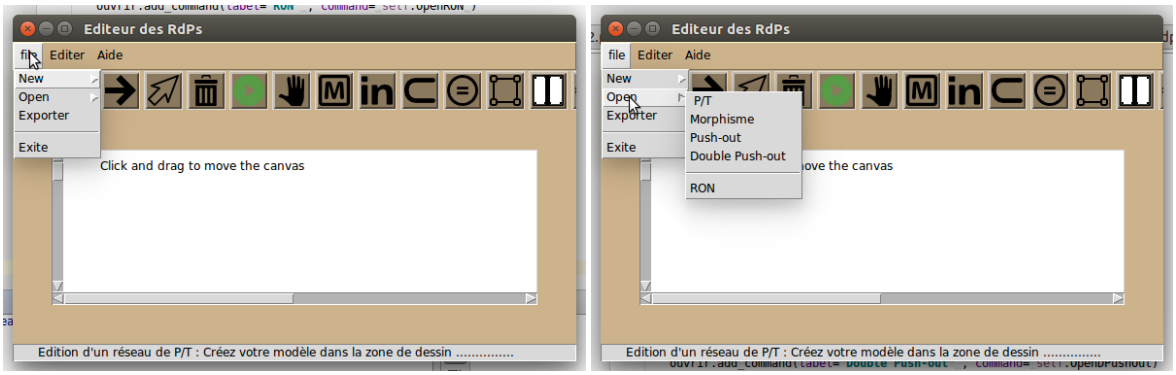


Figure 4.3 New Menu 2.

Menu Bar: 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".

File menu: File menu figure 4.3a contains 2 sub menus; new and open menu (See Figure 4.3b, 4.2 and 4.3a). The sub menus new; allows the user to open new drawing area for Petri net, reconfigurable object nets or to create rule. The sub menus open; allows user to recover his models for editing and simulation. If the user click at the exporter label in the master menu, it's allows to save their models, in their extensions (pnml, rule, ron).

Figure 4.4 Toolbar

Toolbar: toolbar consists of many buttons see (Figure 4.4).

In the toolbar above, 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 used for back to initial marking.

After learning about our tools in general, next we go to the features of our application.

4.2.2 Application features

Our tool consists of three modules: Module for P / T net editor, Module for creating a rule, the last is the module for RON editor, that has been made independently. Below the description of features of each editor.

Petri Net Editor

The first component we implemented was a graphical editor for P / T net (see Figure 4.5). This editor allows the creation and modification of a Petri net in the drawing area using the toolbar above (see 4.2.1.). The first component we implemented is a graphical editor for P / T net (see Figure 4.5). This editor allows to create and edit a Petri net using a toolbar above (see 4.2.1.) this format that allows opening their models in other tools like TINA (see 4.11). After drawing place or transition, the user can change the label and marking of the places (see figure 4.7, 4.8). The models that are drawn are saved in Pnml file (see figure 4.9 below).

Figure 4.5 Module Petri net

After drawing place or transition user can change the label and marking of the places (see figure below). The models are saved in pnml file see figure 4.8 below; this format allows opening their models in other tools like TINA (see 4.10). After drawing place or transition, the user can change the label and marking of the places (see figure 4.6, 4.7). The models that are drawn are saved in Pnml file (see figure 4.8 below).

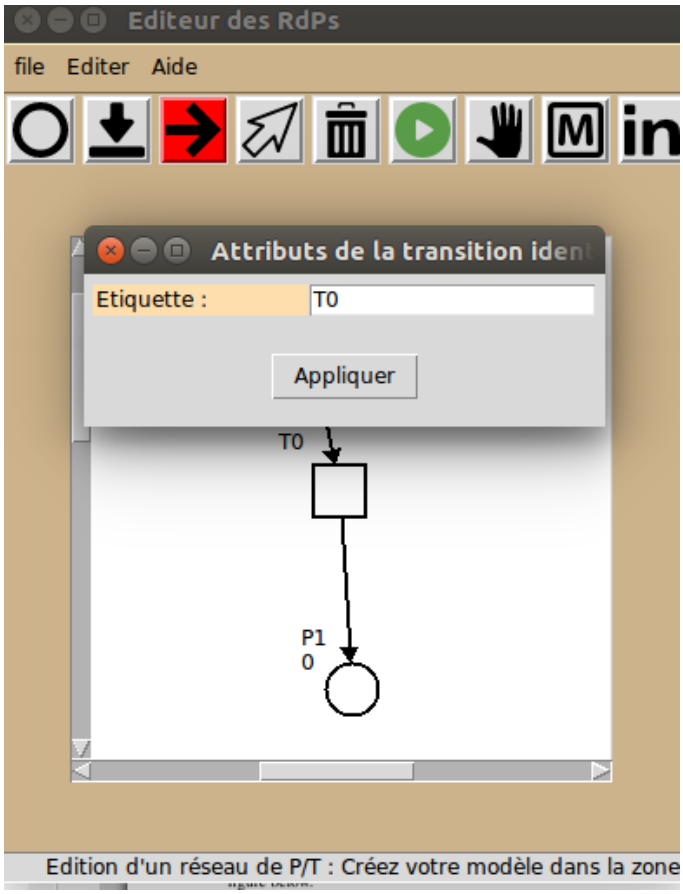


Figure 4.6 Edit Transition

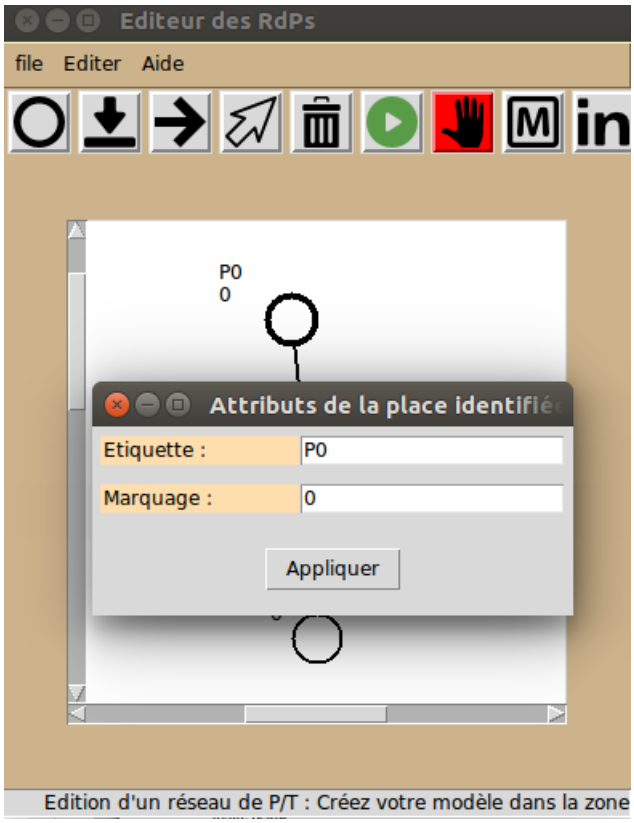


Figure 4.7 Edit Place

```
<?xml version="1.0" ?>
<pnml xmlns="http://www.pnml.org/version-2009/grammar/pnml">
    <net id="74cf1b2c-6a52-11e8-856e-642737d951af" type="http://www.laas.fr/tina/tpn">
        <name>
            <text>74cf1b2c-6a52-11e8-856e-642737d951af</text>
        </name>
        <page id="net_1">
            <place id="765dcede-6a52-11e8-856e-642737d951af">
                <name>
                    <text>P0</text>
                    <graphics>
                        <offset x="345" y="80"/>
                    </graphics>
                </name>
                <initialMarking>
                    <text>0</text>
                </initialMarking>
                <graphics>
                    <position x="365" y="100"/>
                </graphics>
            </place>
            <transition id="7887fd2e-6a52-11e8-856e-642737d951af">
                <name>
                    <text>T0</text>
                    <graphics>
                        <offset x="349" y="161"/>
                    </graphics>
                </name>
                <graphics>
                    <position x="369" y="181"/>
                </graphics>
            </transition>
            <arc id="7ae1e4f4-6a52-11e8-856e-642737d951af" source="765dcede-6a52-11e8-856e-642737d951af" target="
            7887fd2e-6a52-11e8-856e-642737d951af"/>
        </page>
    </net>
</pnml>
```

Figure 4.8 Pnml File

Figure 4.10 and 4.11 show how to use pnml file (Figure 4.8) in other tools like TINA to generate the same model. Moreover, our tool gives more than TINA, give to the user the enabled transition at the same time of drawing the model see the transition T0 (figure 4.10) with green colour, is enabled but on TINA no information about that.

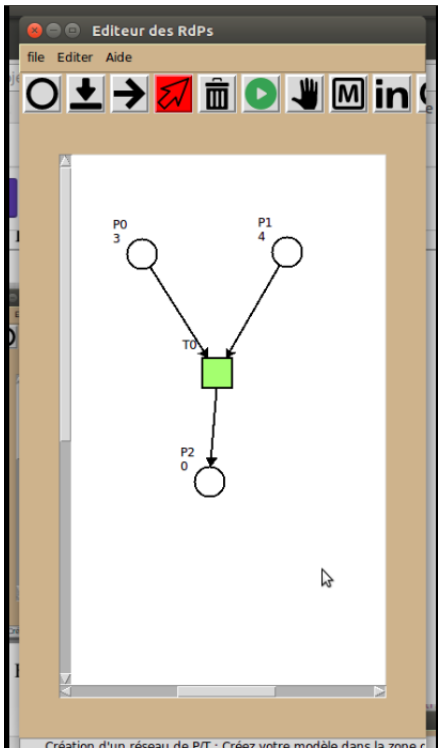


Figure 4.9 Petri Net using our Application

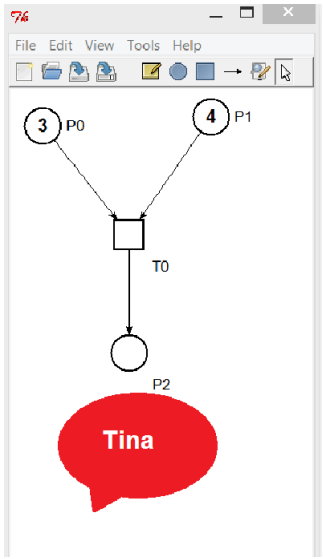


Figure 4.10 Petri Net using TINA

To simulate Petri net in our editor; we use the simulation button in the toolbar (Figure 4.5) after clicking this button the other drawing buttons are disabled.

Simulate Petri Net

To simulate Petri net in our editor, we use the simulation button in the toolbar (Figure 4.4), after clicking this button the other drawing buttons are disabled.

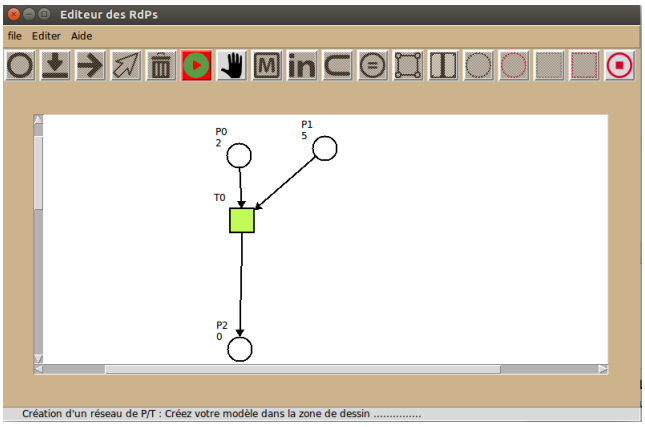
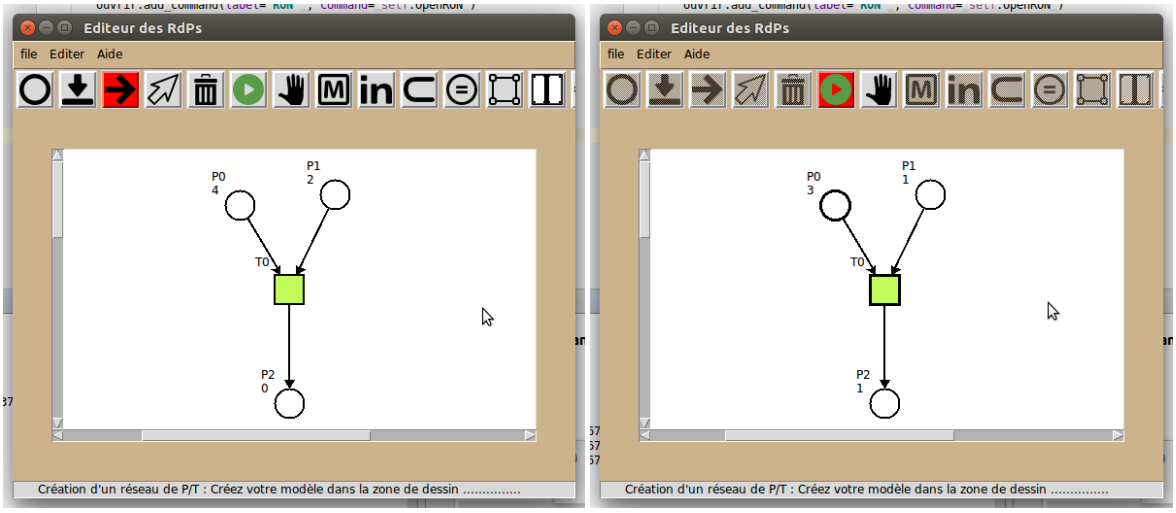


Figure 4.11 Run Simulation

In the previous chapters, we talk about a set of production rules. in our application to create a rule the user does not need to draw again the components of these rules but he needs, just, the pnml file from the Petri net editor.

Figure (4.12a, 4.12b) show that the dynamics (game token) see chapter 2.



(a) Before Firing Transition T0 (b) After Firing Transition T0

Figure 4.12 Before and After Firing Transition T0.

Create Rule:

In the previous chapters, we talk about a set of production rules. in our application to create a rule the user does not need to draw again the components of these rules but he needs, just, the pnml file from the Petri net editor.
The GUI of create rule (Figure 4.13) is composed of 3 parts left-hand, interface, right-hand the user imports the nets of the rules as pnml file using the button with net label see Figure4.13. Each part is shown in the drawing area to make user decide the file that import is correct or not.

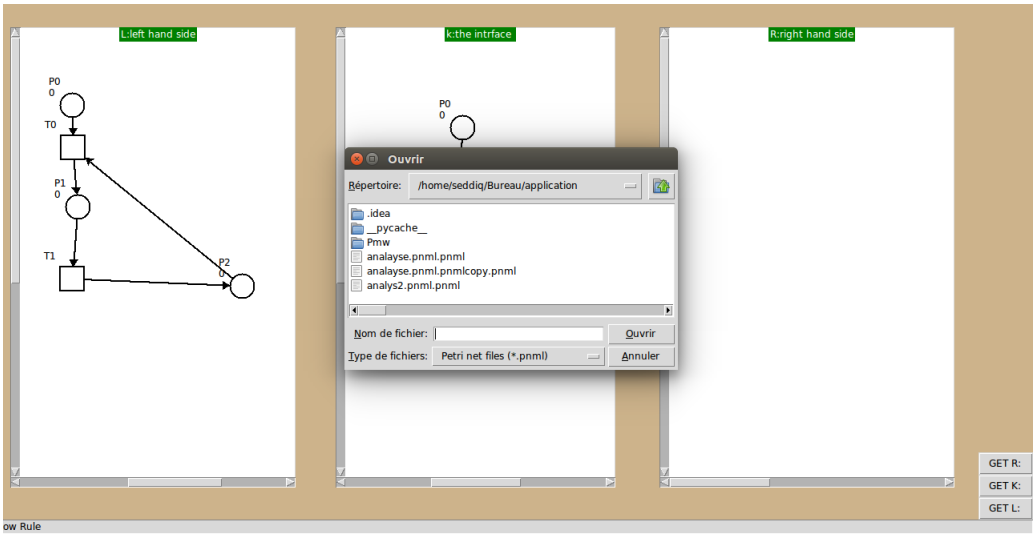


Figure 4.13 Create Rule

Check Rules and Save:

After the importation of components of the rule; the user can check and save the rule. To do that, the user opens the menu bar and clicks in export; file dialog is shown and ask the user about file name and the directory to save it, after clicking on button OK, the process of checker is run in background; this mode makes user check the morphism in different parts of the rule. If there is no problem in the rule, an information message is shown, "the rule is correct". After clicking on the button OK, the rule is saved as XML file with extension (**.rule**).
If not, an information message is shown, " rule not checked ". Moreover, the message indicates the part that does not check the conditions of morphism, in order to provide to the user to know about what's wrong in their rule.

The figures 4.15, 4.16, 4.17 and 4.18 show various morphism checker; this is helpful to the user in creating their rules. The first figure gives an information message which confirms that this rule is correct. After that, when the user clicks on OK button, the rule is saved in XML file with extension **rule**.

```
<?xml version="1.0" ?>
<rule>
    <Rule auteur="seddiq benterki">
        <Left_hand path="/home/seddiq/Bureau/application/left.pnml.pnml"/>
        <Interface path="/home/seddiq/Bureau/application/interface1.pnml.pnml"/>
        <Right_hand path="/home/seddiq/Bureau/application/right3.pnml.pnml"/>
    </Rule>
</rule>
```

Figure 4.14 RULE FILE

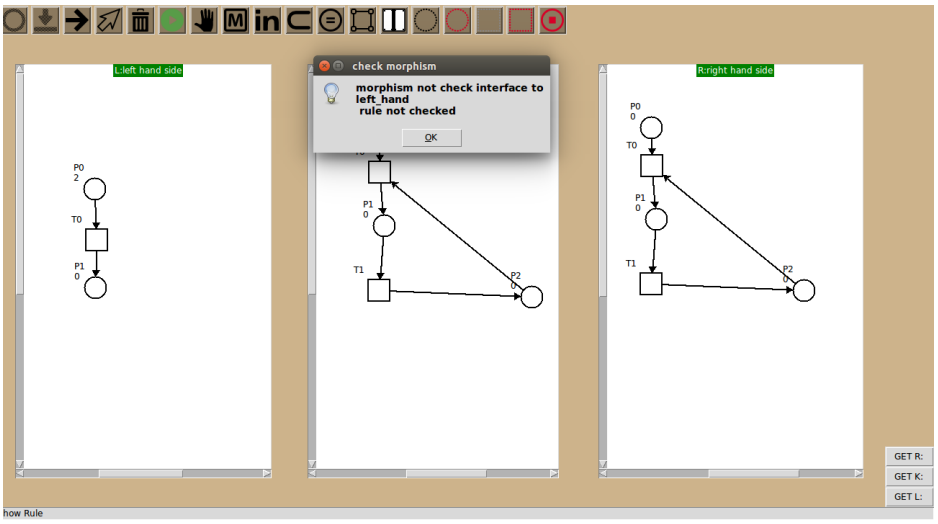


Figure 4.15 Check Morphism 1

The message in figure 4.15 provides that there is no morphism from the interface to the left hand. Figure 4.16 shows that rule is fully not correct.

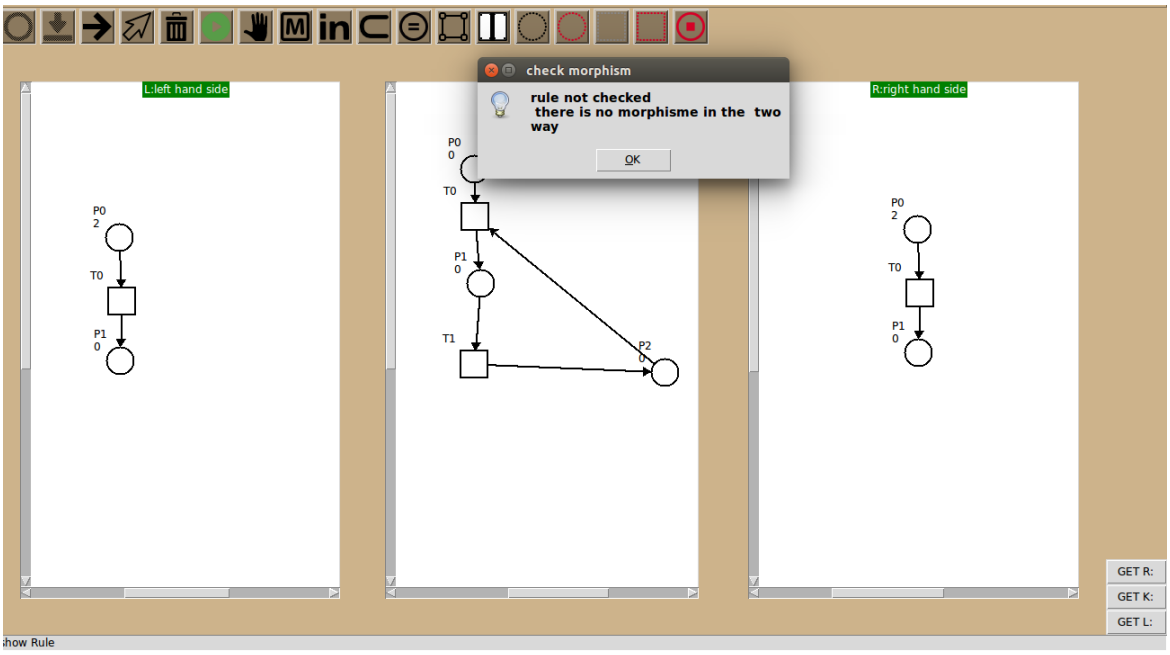


Figure 4.16 Check Morphism 2

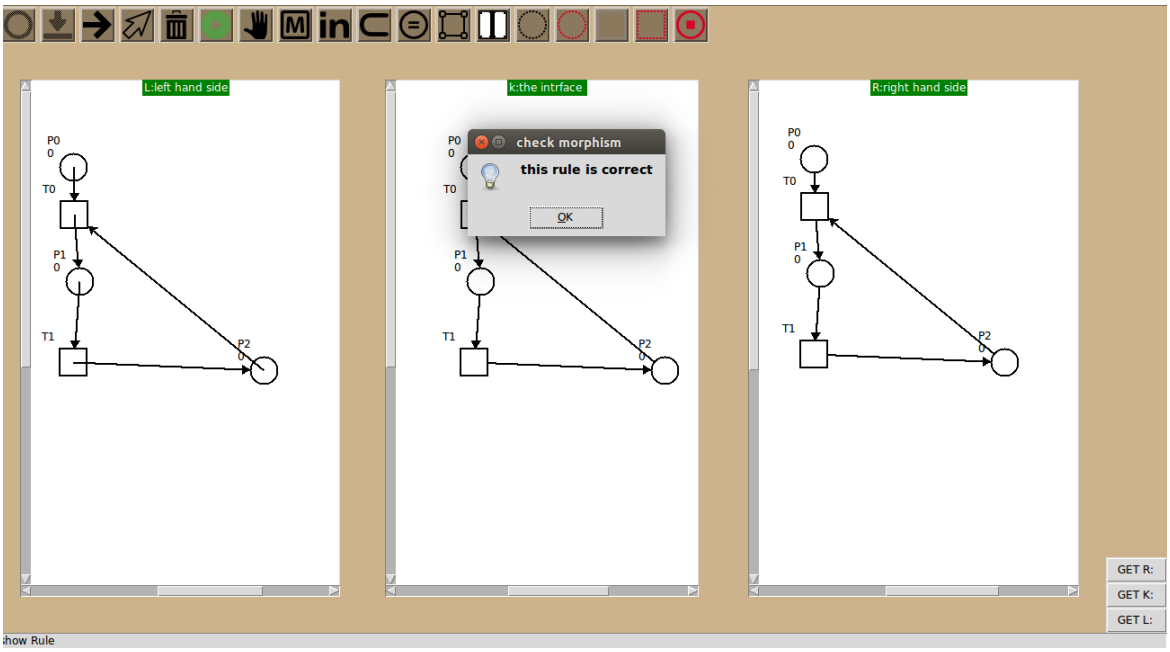


Figure 4.17 Check Morphism 3

The figure above shows that the case when the user can get rule file. This means that the user is creating a correct rule.

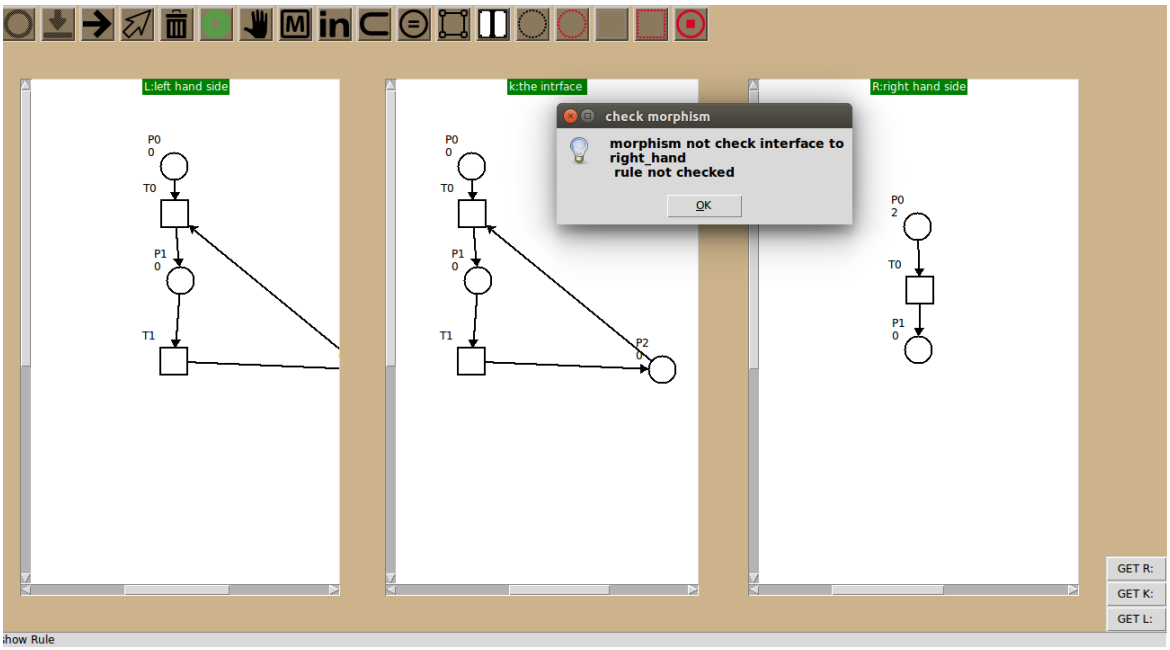


Figure 4.18 Check Morphism 4

Ron Editor:

Previously, we talked about Petri net editor drawing and simulation. In the production rule we talk about the creation and validate this rules, this latter will be the basis for the creation of the RON$_s$ model in our tool. Before entering in more details about how to use our tool in creation we back to the definition of RON$_s$. RON$_s$ represent a high-level Petri net which is composed of two parts of places (net-places and rule-places). Net-places are used to describe the models of different systems which can change their structure at the runtime if these systems satisfy the production rules. These rules represented as tokens in the set of rules places. Ron$_s$ are with two other types of components called transform transition and fire transition. These transitions will be used in the dynamic of the RON$_s$ model. Transform transition is used to change the net structure after the net satisfied the production rule (Figure 4.15). The fire transition provides the net to changes it's marking. The rest of this chapter is intended for explaining the instructions of using our tool (from editing the RON model to the simulation).

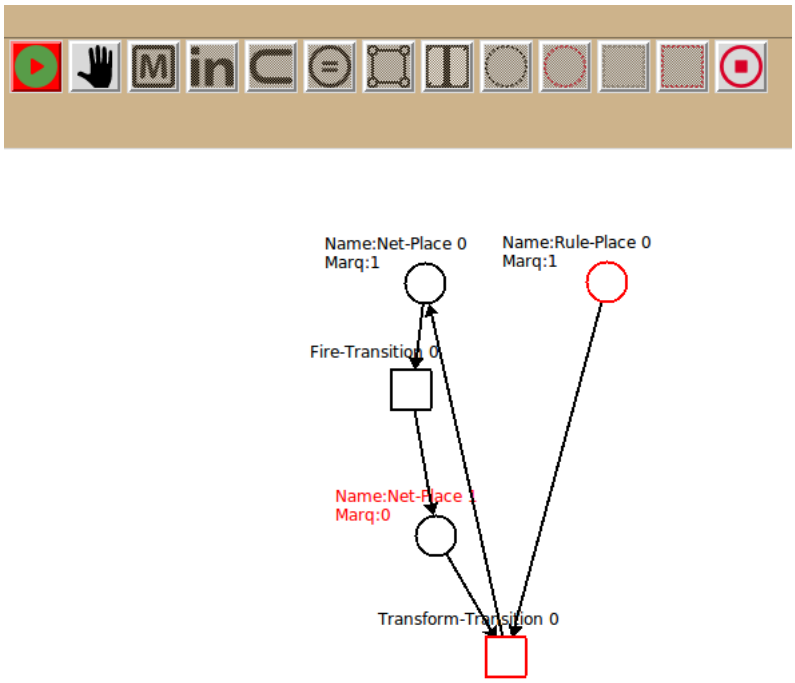
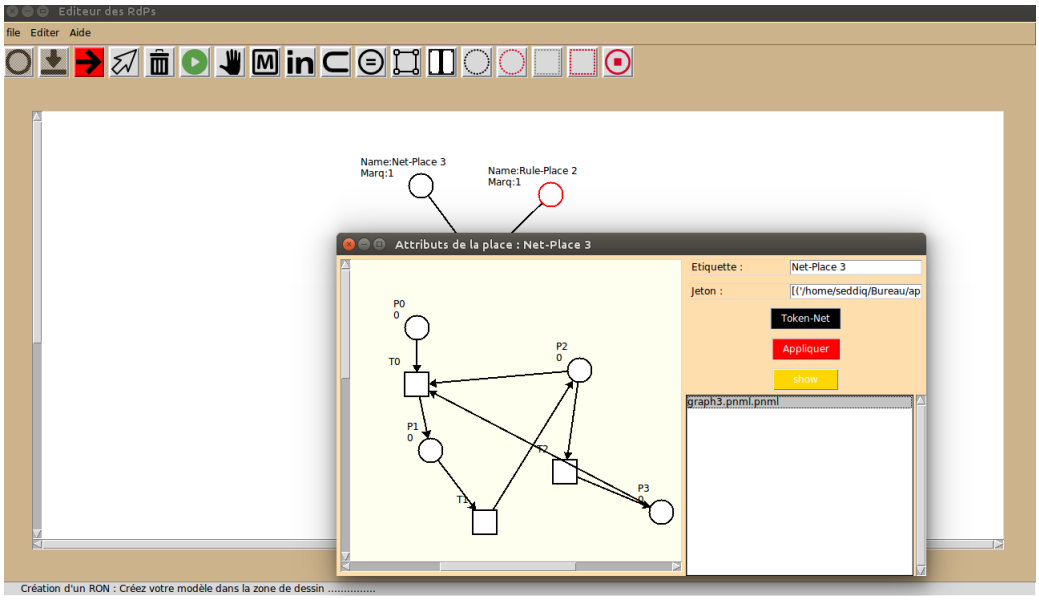


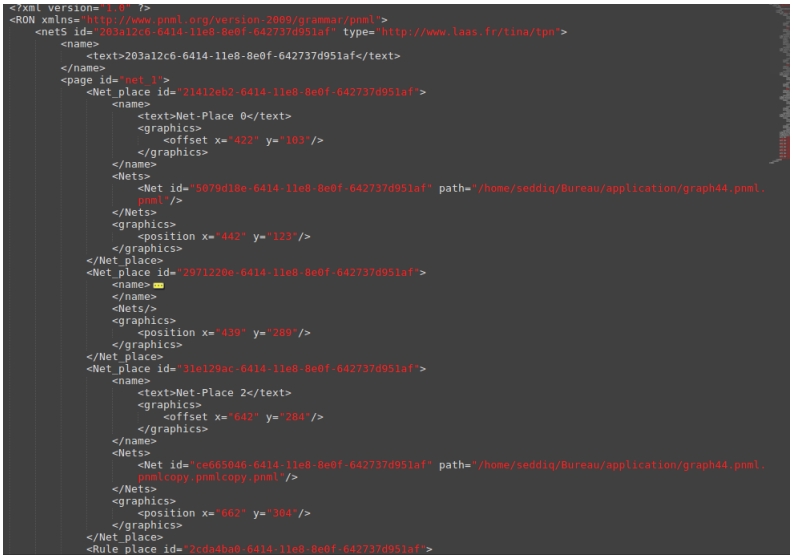
Figure 4.19 A RON model

The figure 4.19 above depicts our RON$_s$ model. In the next, we give the steps of creating this model using our tool. To create RON$_s$ model in the drawing area. First, open the new menu in the toolbar 4.2 and choice new RON. After that, you need the toolbar (Figure 4.4) to draw the different components of the structure of the RON$_s$ model.



.

Figure 4.20 ADD NETS



.

Figure 4.21 File RON

After drawing the structure of the RON$_s$, you must add the behaviour, nets and rules. We will use the results of the drawing of Petri net; it's represented pnml file (See figure 4.8) and the rules; it is represented by rule file (see figure 4.14). The saved of RON$_s$ model will be in the xml file with the extension **.ron** (see figure 4.21).

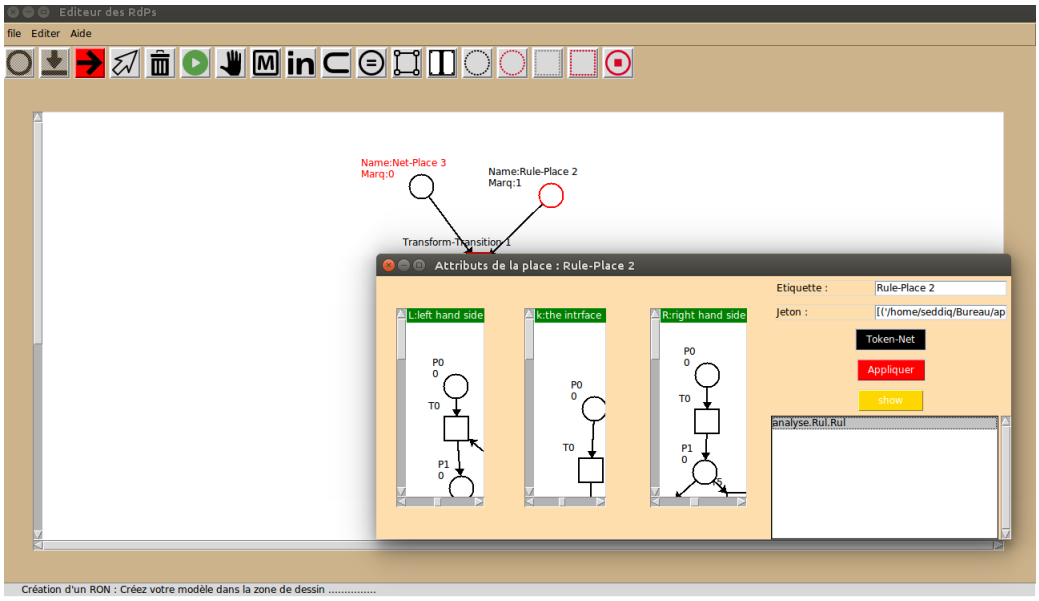


Figure 4.22 ADD RULES

The figure above 4.20 is the interface for adding nets. The user can add multiple nets and can change the name of places as well as he can show the nets in the drawing area.

The figure 4.22 shows how can user add set of production rule in the rule places. To do that we exploited the same GUI of adding nets but with one difference in the drawing area. When the user adds the set of rules, he can show the three parts of rules.

Simulate RON$_s$

After that, the user completes his model and wants to simulate it, the user clicks on the simulation button located on the toolbar. The simulation runs as the simulation of the basic nets. The other buttons are disabled and the enabled transitions are coloured. The rest of the simulation takes another way which differs from simulation basic Petri nets. The following is a description about simulation RON$_s$ step by step.

Apply Fire Transition The applying of fire translation provides the Ron model by change it's marking, more the net in the set of pre-conditions can change its marking. We do that by using an external interface displayed when clicking on the enabled transition.

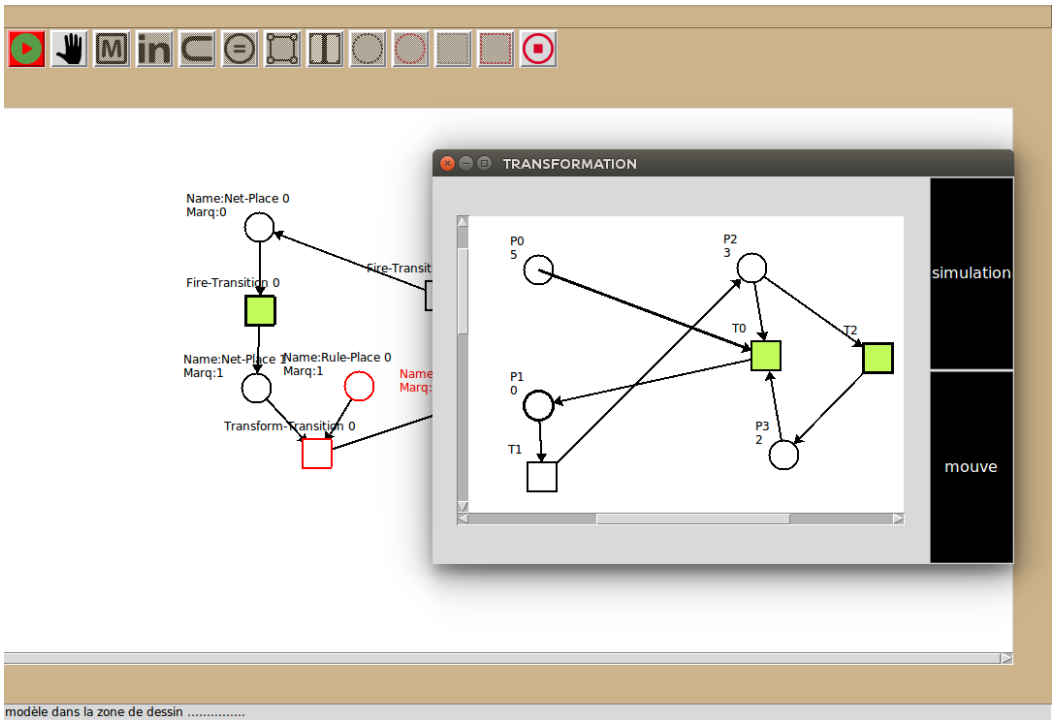


Figure 4.23 Apply fire Transition 1

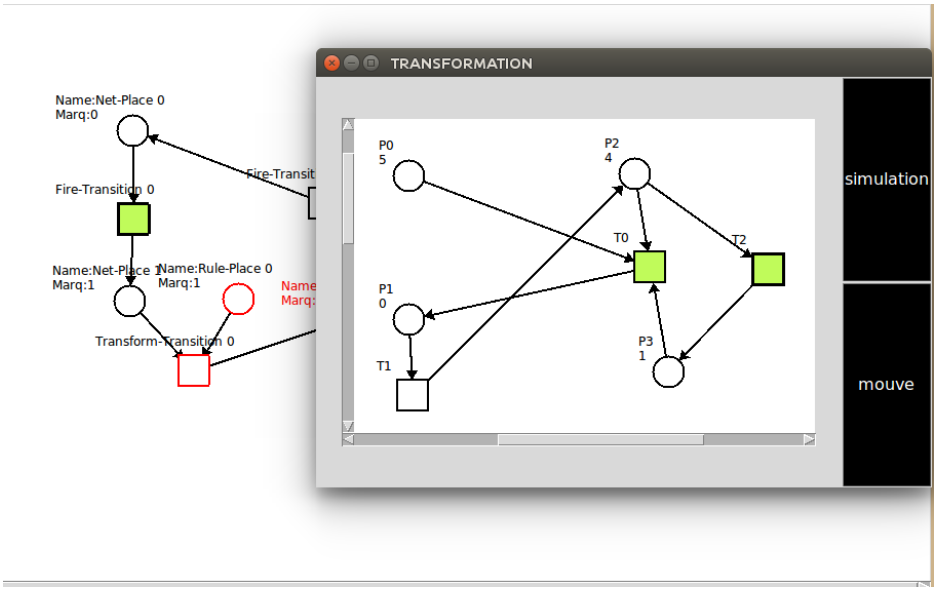


Figure 4.24 Apply fire Transition 2

The screen-shot above is about how apply the fire transition, with the simulation button in the external frame. The user can play the game tokens[2] like basic petri nets (see simulate petri net above figure 4.12a,4.12b).

[2]Game tokens is the firing the transition in basic petri nets

Apply Transform Transition The applying of transform transition is summarized as follows:
After the click on the enabled transition (see the condition of apply rules), a new frame is displayed the Petri nets after the transformation step. Moreover, there are 4 buttons for showing different parts of the rule and source nets (See figure 4.25).

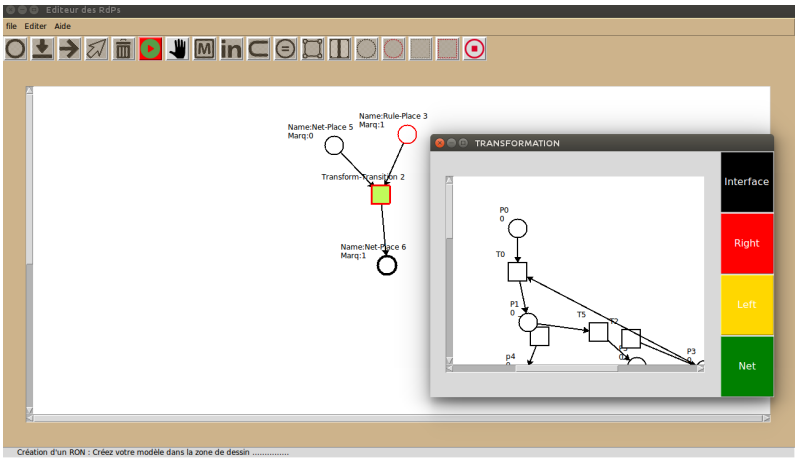


Figure 4.25 Apply Transform Transition 1

The user can see all the parts of rules and the source nets just with one click on the button which has the name of the needed part.
The following pictures explain how the user can see the nets transformation in the different vision.

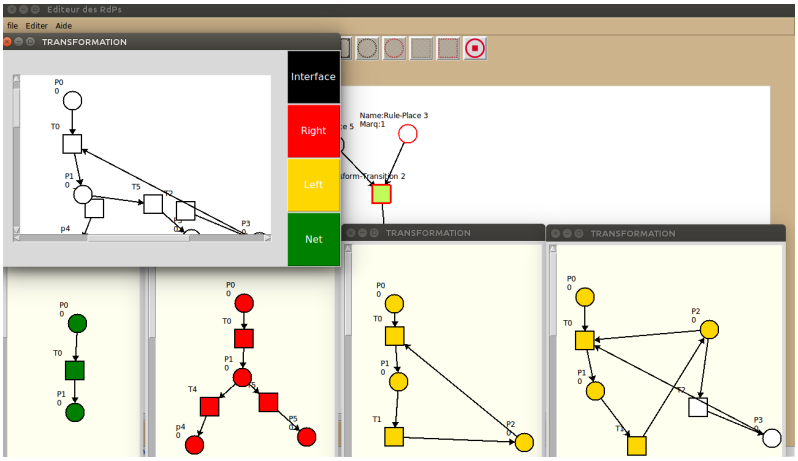


Figure 4.26 Apply Transform Transition 2

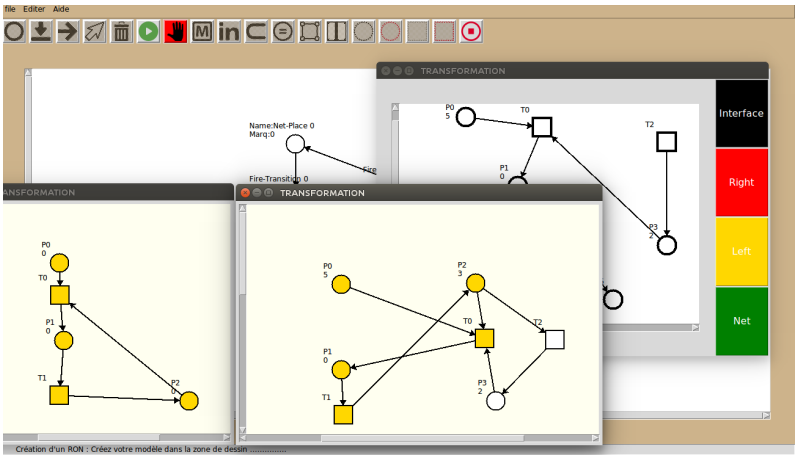


Figure 4.27 Apply Transform Transition 3

The last two figures show that the user can display any part using in the nets transformation; more than the figure 4.27 shows that the user can find the occurrence morphism on the net with the colouring of the deleting part.

The figure 4.26 shows that the user can display all the parts of the transformation at the same time with the colourization of each part, with special colour, for more flexibility in the simulation.

Conclusion

In this chapter, we have presented the tools that we have used to implement our application, then the results of our implementation as a set of graphical user interfaces (GUI).

Conclusion

Reconfigurable Petri nets are suitable to model, analyse and diagnose the systems which are parallel and reconfigurable, to allow the reliability in this type of systems. In fact, reconfigurable Petri nets provide powerful and intuitive formalism to model dynamic software or hardware systems that are executed in dynamic infrastructures, but they are difficult to handle. Modelling and simulating dynamic systems require an automatic tools. Up to now, there have been two tools that also implement some kind of reconfigurable Petri nets. However, these tools are not sufficient, which motivates us to develop a new tool for reconfigurable Petri nets. This tool is suitable for modelling and simulating the reconfigurable object nets. We have implemented the tool using Python programming language.

As future work, we plan to improve and complete the developer tool and to consider the analysis of RONs (Reconfigurbale Object Nets). On the theoretical level, look for a new invented a new method to provide more power and flexibility for reconfigurable Petri nets.

Bibliography

[BEE+09] Enrico Biermann, Hartmut Ehrig, Claudia Ermel, Kathrin Hoffmann, and Tony Modica. Modeling Multicasting in Dynamic Communication-based Systems by Reconfigurable High-level Petri Nets. In *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2009, Corvallis, OR, USA, 20-24 September 2009, Proceedings*, pages 47–50. IEEE, 2009.

[BEHM07] Enrico Biermann, Claudia Ermel, Frank Hermann, and Tony Modica. A Visual Editor for Reconfigurable Object Nets based on the ECLIPSE Graphical Editor Framework. In G. Juhas and J. Desel, editors, *Proc. 14th Workshop on Algorithms and Tools for Petri Nets (AWPN'07)*, Universität Koblenz-Landau, Germany, October 2007. GI Special Interest Group on Petri Nets and Related System Models. http://www.user.tu-berlin.de/lieske/tfs/publikationen/Papers07/BEHM07.pdf.

[EEHJG99] H. Ehrig, G. Engels, H.-J.Kreowski, and G.Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation, Concurrency, Parallelism and Distribution.*, volume 2. World Scientific, Singapore,, 1999.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Fundamentals of algebraic graph transformation. *EATCS Monographs.Springer*, 2006.

[EHP+07] Hartmut Ehrig, Kathrin Hoffmann, Julia Padberg, Ulrike Prange, and Claudia Ermel. Independence of net transformations and token firing in reconfigurable place/transition systems. In *Petri Nets and Other Models of Concurrency - ICATPN 2007, 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings*, pages 104–123, 2007.

[EMH73] H. Ehrig, M.Pfender, and H.J.Schneider. Graph grammars: an algebraic approach. in proceedings of focs 1973. *IEEE*, pages 167–180, 1973.

[EP04] Hartmut Ehrig and Julia Padberg. *Graph Grammars and Petri Net Transformations*, pages 496–536. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Gab14] Karsten Gabriel. *Interaction on human-centric communication platforms: modelling and analysis using algebraic high-level nets and processes*. PhD thesis, Berlin Institute of Technology, 2014.

[GNH12] Susann Gottmann, Nico Nachtigall, and Kathrin Hoffmann. On modelling communication in ubiquitous computing systems using algebraic higher order nets. *ECEASST*, 51, 2012.

[HEM05] Kathrin Hoffmann, Hartmut Ehrig, and Till Mossakowski. High-level nets with nets and rules as tokens. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets 2005*, pages 268–288, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[HHJUG99] H.Ehrig, H.-J.Kreowski, U.Montanari, and G.Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation, Concurrency, Parallelism and Distribution.*, volume 3. World Scientific, Singapore,, 1999.

[KBD16] Laïd Kahloul, Samir Bourekkache, and Karim Djouani. Designing reconfigurable manufacturing systems using reconfigurable object petri nets. *Int. J. Computer Integrated Manufacturing*, 29(8):889–906, 2016.

[Lam17] Leslie Lamport. LaTeX. https://en.wikipedia.org/wiki/LaTeX, 2017. accessed on May 18, 2017.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, 1989.

[PEHP08] Ulrike Prange, Hartmut Ehrig, Kathrin Hoffmann, and Julia Padberg. *Transformations in Reconfigurable Place/Transition Systems*, pages 96–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[PEOH12] Julia Padberg, Marvin Ede, Gerhard Oelker, and Kathrin Hoffmann. Reconnet: A tool for modeling and simulating with reconfigurable place/transition nets. *ECEASST*, 54, 2012.

[PK18] Julia Padberg and Laid Kahloul. *Overview of Reconfigurable Petri Nets*, pages 201–222. Springer International Publishing, Cham, 2018. https://doi.org/10.1007/978-3-319-75396-6_11.

[PS16] Julia Padberg and Alexander Schulz. Model checking reconfigurable petri nets with maude. In *Graph Transformation - 9th International Conference, ICGT 2016, in Memory of Hartmut Ehrig, Held as Part of STAF 2016, Vienna, Austria, July 5-6, 2016, Proceedings*, pages 54–70, 2016.

[ReC] *A Tool for Reconfigurable Petri Nets*. https://reconnetblog.wordpress.com/.

[RON] *An editor for Reconfigurable Object Nets*. http://www.user.tu-berlin.de/o.runge/tfs/projekte/roneditor/.

[Roz97] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation, Concurrency, Parallelism and Distribution.*, volume 1. World Scientific, Singapore,, 1997.

[Tin] *LAAS CNRS home page*. https://www.laas.fr/public/fr.

[tk05] An Introduction to Tkinter. http://effbot.org/tkinterbook/, 2005. accessed on May 18, 2017.

[TMa17] L'éditeur LaTeX universel. http://www.xm1math.net/texmaker/index_fr.html, 2017. accessed on May 18, 2017.

[Val04] *RudigerValk. Object Petri Nets Using the Nets-within-Nets Paradigm*. Springer, Berlin, Heidelberg, 2004.