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 Verifying Reconfigurable Petri Nets

By:
Abderrahmane Chabira

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

University Year: 2017/2018

Acknowledgement:

I thank Allah for giving me the health and the strength to begin and end this project.

First of all, this work would not have been done without the help and guidance of Mr. Samir Tgane.

I thank him for the quality of his Exceptional work, for his patience, his rigor and his availability during my preparation for this graduation project.

I thank Mrs Zohra Hmidi and Mrs. Amina Zouiouche for agreeing to review this work. I also thank all our teachers for their generosity and the great patience they have shown despite their academic and professional responsibilities.

Dedication:

This work is specifically dedicated to my beloved parents.

I must express my very profound gratitude to my family and to Chayma Bouzenag for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of this work. This accomplishment would not have been possible without them.

Contents

List of Figures

Abstract

The objective of this work is using reconfigurable Petri nets to draw important conclusions about the system without going for time and cost ineffective trial and error prototyping. To do so, the first step is to model the system. Once a model is ready, the next task is to analyze the model to draw conclusions about the properties of the model and hence about the actual system. Then only one can answer questions like what the system behavior is supposed to be under specific operational conditions, what properties are inherent to the structure of the net, what to expect from the system during operation and whether there is any pitfall in the system design which must be avoided in operational phase.

General Introduction

In the last decades, formal verification has been suggested as an effective way of improving reliability. Unlike testing, formal verification examines all behaviours of the system and compares them to a formal specification, which distinguishes the good behaviours from the faulty ones. If the system is declared correct, formal verification has established a mathematically accurate proof that the system is free from all faults described by the specification. Otherwise, depending on the particular technique used. formal verification cannot guarantee absolute correctness, it only ensures the system to be as correct as the specification is. Two well known methods for proving correctness model checking and theorem proving. Model checking is a fully automatic technique. Here, a computer exhaustively explores all possible execution paths that the system could follow. Even very small systems contain a huge number of execution paths,which makes it very difficult to analyze. In our project we are interested in verifying reconfigurable systems that are modelled with reconfigurable Petri nets,this last are class of high level Petri nets that can dynamically change their own structures based on reconfiguration rules.
This rapport is organized as follows:

- chapter I: in this chapter we will see a general view of the Petri nets with some informal definitions, furthermore, we will see some of PN's formal definitions, some of its properties and methods of analyzing it. Finally, we will introduce Petri net as a modeling tool for the dynamic systems.

- chapter II: this chapter contains two important parts. In the first part we will have an overview of the reconfigurable systems and see some real examples. For the second part we will have an abstract view of the reconfigurable PN.

- chapter III: this chapter consists of two parts. First part, which is analysis stage, where we go deeper in reconfigurable Petri nets and see formal definition, marking graph algorithm and finally an example to make things clear. Second part, which is conception stage, where we explain how our tool is developed

- chapter IV: the final chapter, which contains some definitions of the tools used during this project, the programming language used and the implementation of our tool with some guide of use.

Chapter 1

Petri nets

1.1 Introduction

In the engineering discipline, system evolution has invariably been facing three major needs [2]:

- The need to develop increasingly complex systems.
- The need to assess the system's operational risks.
- The need to have a cost competitive solution to attain these requirements.

Due to time and money constraints, it is no longer feasible to follow the design cycle of trial and error prototyping. Instead, the industry is leaning more towards simulation, so that the design flaws can be worked out even before the prototype is built. It's here Petri Net comes in. Petri net is a net-based abstraction, which can be used as a modeling tool (graphical and mathematical), as a simulation tool and as an analysis tool. As a modeling tool, it helps in system design. The graphical nature aids system visualization, the mathematical nature captures system behaviour As a simulation tool, it enables one to identify design errors. Extensive simulations may detect errors, which are rare and elusive. As an analysis tool it reveals various properties of the model and hence of the actual physical system. Thus, one can draw important conclusions about the system without going for experimentation or performing lengthy calculation of conventional system modeling.
Petri nets are a formalism for modelling concurrent systems, first defined by Carl Adam Petri in 1962. They allow correctness of concurrent systems to be verified using well-defined, provable mathematical techniques and allow the behaviour of a system to be expressed both graphically and algebraically. They support the natural expression of such concepts as synchronisation and communication between processes. The ability they provide to visualise the structure of a system promotes greater intuitive understanding of what is being modelled. Petri nets have since been extended and augmented with additional behaviours. The building blocks of a Petri

net are places, transitions, arcs and tokens. Places model conditions or objects. Places may contain tokens, which represent the value of the condition or object. Transitions model activities, which change the value of conditions or objects. For example, firing a transition may destroy a token at one place and create a token at another place. The interconnectedness of places and transitions is represented using arcs. Each arc has one and only one source, and one and only one target. If the source is a place, the target must be a transition, and vice versa.

1.2 Informal definition

Any system consists of a number of activities and the system can be modeled by listing the states of the system, before and after those activities. An activity brings the system from one state to another i.e. activity causes state-transition. All such state-transitions, when graphically represented, are called state-transition diagram [2].

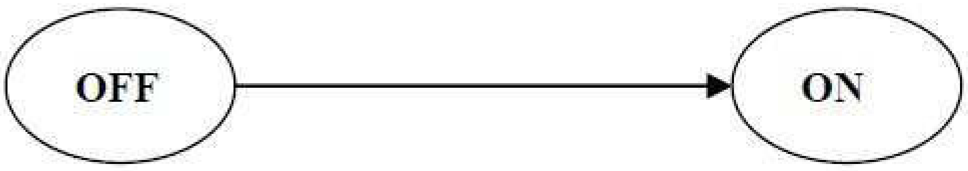


Figure 1.1: State-transition diagram for OFF-ON transition

The above state-transition diagram shows that the system undergoes a transition from OFF state to ON state. The activity, in this case, can be pressing of a switch. A closer look at Fig. 1.1 reveals that a state-transition diagram is a directed graph composed of two elements: nodes (representing state of the system) and arcs (representing the direction of state-transition). Pictorially nodes are represented by circles. Arcs are of two types: input arcs and output arcs. In Fig. 1.1, there are two nodes, representing OFF and ON state. The only arc in Fig. 1.1 is the output arc with respect to OFF node and input arc with respect to ON node. The state-transition representation, as shown above, has some serious limitations. As one can see in Fig. 1.1, there is no representation of the activity itself. Also, there is little or no scope to represent the condition (if any) for which the transition occurs (say, if the temperature is less than 40oC then the switch is pressed to make the system move from OFF to ON state). In addition to that, one has to define first the system's global state and then enumerate all the states and feasible events at each state. A possible consequence is the state-explosion problem for complex systems. A formalism that can overcome some of these limitations is Petri nets. It gives more modeling flexibility by introducing two kinds of nodes; one (called places) to represent the states and/or conditions and the other (called transitions) to represent the activities. It uses local states rather than global states, thereby avoiding the state enumeration problems in the modeling stage. It can explicitly represent precedence relations, conflicting situations, synchronization concepts, concurrent operations and mutually exclusive events.

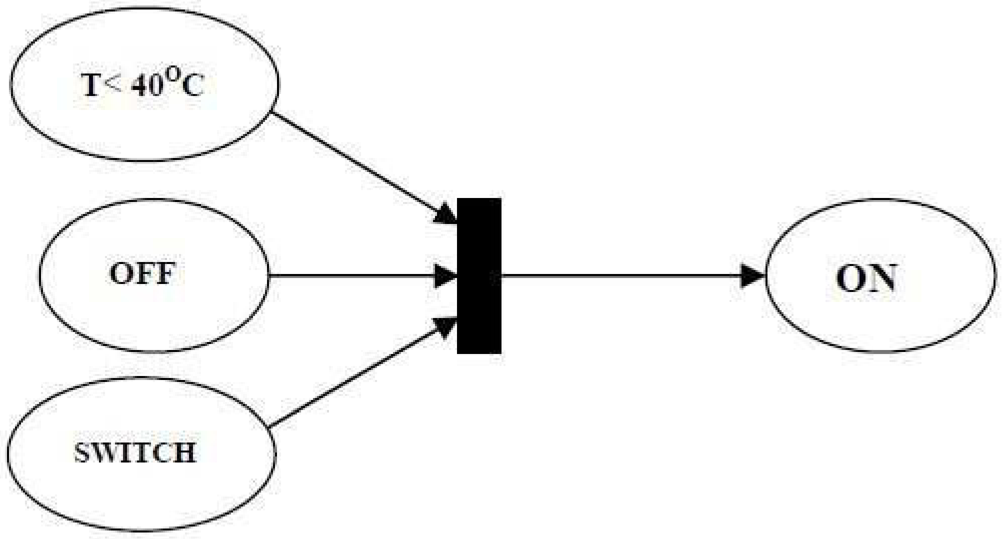


Figure 1.2: Petri net graph for OFF-ON transition
[2]

The Petri net graph of Fig. 1.2 shows that when the system is in OFF state and when temperature is less than 40^oC and when the switch is pressed then the system makes a transition from OFF state to ON state. A comparison of Fig. 1.1 and Fig. 1.2 shows that how Petri net graph provides more modeling power and flexibility over state-transition diagram. A closer look at Fig. 1.2 reveals that a Petri net is a directed graph composed of two elements: nodes (places and transitions) and arcs (input and output). Pictorially places are represented by circles and transitions by bars. Arcs are labeled with their weights (positive integers) – labels for unity weight are usually omitted. The nodes and arcs constitute the static structure of Petri net. The dynamic behavior of the net is given by 'token game', representing various states of the system. A particular state is a snapshot of the system's behavior. The state of a place is called its marking, represented by the presence (condition holds) or absence (condition does not hold) of black dots, called tokens, in the circle representing the place. Current state of the modeled system (marking of the system) is given by the number and type (if tokens are distinguishable) of tokens in each place. While places and arcs are passive components of the net, transitions are active components. When all input places and no output places of a transition contain tokens, then a transition fires. Firing of a transition removes tokens from all of its input places and puts tokens in its output places. Thus, token-flow occurs via the firing of transitions. The system achieves a new marking via the firing of a transition. Introduction of tokens into places and their flow through transitions enable one to describe the discrete-event dynamics of the PN and thereby of the modeled system [2].

1.2.1 Definition of ordinary Petri net

An Ordinary Petri net is one where all arcs are unity-weighted (and hence unlabeled),where $\omega{:}A \to N^*$ is the weight function on the arcs, which associates with each arc an integer 1. It can be mentioned that ordinary and non-ordinary Petri nets have same modeling power since one can always represent an arc of higher weight as a set of arcs, each of unit multiplicity, the cardinality of the set being the weight of the arc of non-ordinary Petri net. Therefore, it is always possible to convert a non-ordinary Petri net into an ordinary Petri net without sacrificing generality but sometimes non-ordinary Petri nets are preferred due to ease of modeling [2].

1.3 Formal defination

Definition 1. A PN is a quadruplet $Q = (P, T, A, \omega)$ where [3]:

- $P = \{p_1,p_2,\ldots, p_n\}$ a finite set of places.
- $T = \{t_1,t_2,\ldots,t_m\}$ a finite set of transitions where:
  $P \cap T = \varnothing$ and $P \cup T = \varnothing$.
- $A \subseteq (P \times T) \cup (T \times P)$ a set of arcs that connect a place p_i to a transition t_j or a transition t_j to a place p_i.
- $\omega{:}A \to N^*$ is the weight function on the arcs, which associates with each arc an integer differs from zero and greater then 1.

Definition 2. The matrices *Pre*, *Post* and C are defined as follows [3]:

- $Pre(p_i, t_j)$ is a matrix that represents the weight of the arc connecting p_i to t_j.
- $Post(p_i, t_j)$is a matrix that represents the weight of the arc connecting t_j to p_i .
- C is matrix of incidences and it is defined by:
  $C = Post(p_i, t_j) - Pre(p_i, t_j)$.

1.4 Dynamics

1.4.1 Fireable transition

A transition t is fireable if each place of entry of transition t contains at least a number of tokens equal to the weight of the directed p_i connection arc.

Definition 3. Formally, a transition t is fireable for M marking if:

- $\forall p_i \in {}^{\bullet}t$: $M(p_i) \geq Pre(p_i, t)$.
- ${}^{\bullet}t$ is the set of places of entry of transition t.
- We note by $M \xrightarrow{t}$ 't' should be fireable for M.

1.4.2 firing a transition

Firing a fireable transition t leads us from a current marking M to a new marking $M^{'}$, by updating the marking of the input /output places of t defined in M.

Definition 4. Firing a transition t from a marking M to $M^{'}$is formalized by:

- $M'(p) = M(p) - Pre(p, t) + Post(p, t)$.
- we write $M \xrightarrow{t} M'$.
- where $M^{'}$is accessible from M.

Figure 1.3 represents the firing of a transition in PN.

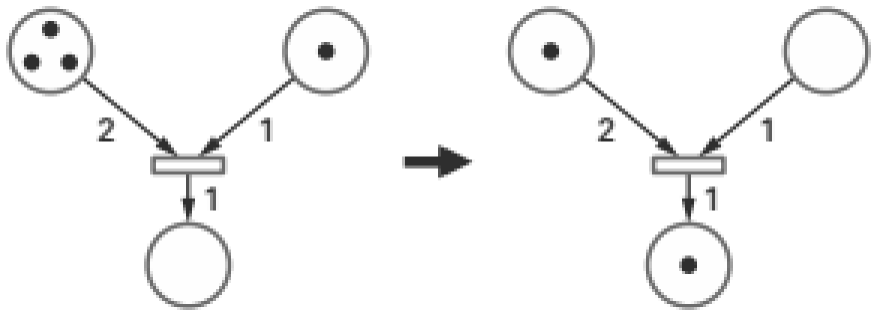


Figure 1.3: Firing a transition

1.5 Analysis

The analysis of a PN consists in verifying certain properties on the model. The behavioral analysis of many systems can be described in terms of system states and their modifications. In order to simulate the dynamic behavior of a system, a marking in the Petri net is changed, and a new state of the systems would have been created. This is called analysis with the dynamic method. However, the analysis can use a set of algebraic tools that do not give us the marking graph of a Petri net, but is based on the analysis of the equations of state. This is called the static method, it is independent of the initial marking.

1.5.1 PN properties

Two types of properties are generally distinguished: the behavioral properties, relate to an initially marked PN, and the structural properties relate to a PN structure independently of any initial marking [3].

1. Accessibility: The answer to the question is that there is a state M'accessible from another state M (possibly M_0)?

2. Boundedness: A place is called bounded if its marking never exceeds a bound k for all states. A PN is k-bounded if all places are bounded and the maximum number of tokens is k.

3. Reversibility: A PN is said to be reversible or reset for an initial marking M_0, if for any marking M accessible from M_0, there exists a transitions firing sequence S which leads to M_0.

4. Liveness: A Petri net is live if and only if all its transitions are live. A transition t of a Petri net having M_0 as initial marking is said live if for any marking M of the Petri net that is reachable from M_0 we can always find a sequence of fireable transitions. The vivacity of a Petri net depends on its initial marking M_0. this property makes it possible to deduce if a system does not involve blocking.

5. Deadlock: A state M is called a dead-end if it contains no fireable transition: $\nexists t \in T : M \xrightarrow{t}$.

1.5.2 Dynamic analysis method

1.5.2.1 Reachability

The reachability problem is stated as follows. Given a Petri net, given an initial marking m_0, given another marking m_r. The question is whether there exists a sequential firing of transitions which will bring the net from m_0 to m_r. If the answer is 'yes', then m_r is said to be Reachable from m_0. The set of all possible markings reachable from m_0, is called the *ReachabilitySet*, denoted by the symbol $R(m_0)$ for a given PN. Note that reachability set is defined for a given PN, for a given initial marking m_0. This dependency on initial marking clearly reveals that reachability is a behavioral property. It may happen that m_r is reached from m_0 by the firing of a single transition, in that case m_r is said to be immediately reachable from m_0. In the general case, m_r is reached via the sequential firing of r transitions, called the firing sequence, denoted by $\sigma_r = t_1, t_2, \ldots, t_r$ [4].

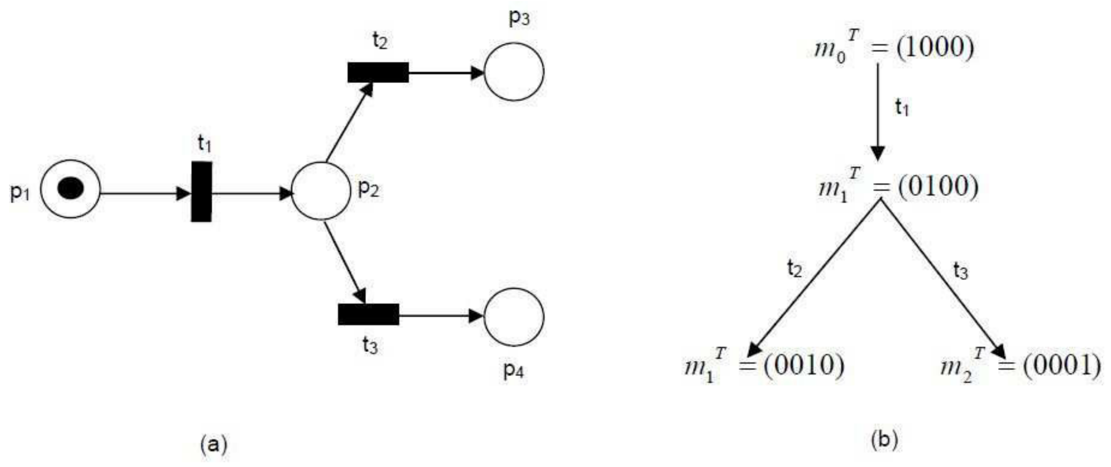


Figure 1.4: (a) Petri net graph, (b) reachability tree for (a)

Example. It is, however, very much possible that the Reachability Tree could grow indefinitely. To circumvent this problem, the concept of 'pseudo-infinity' is introduced, so that the tree size can be kept finite. 'Pseudo-infinity', as the name suggests, can be thought of as a finite representation of infinity. 'Pseudo-infinity' is denoted by the symbol ω which is subject to the following properties: $\omega > a$ where a is any integer. With the above concept of 'pseudo-infinity' (ω) one can always keep the tree finite. When the (ω) symbol is absent, then the term Reachability Tree is used. If, the (ω) symbol is present (truly infinite size tree, represented as finite size tree by introducing (ω) then the term Coverability tree is used. This means the term Coverability Tree is more general. When there is no (ω) symbol present then the terms Reachability Tree and Coverability Tree are synonymous. Otherwise the term Coverability Tree should be used. Now the question is, given a PN, how to construct the coverability tree. The following algorithm [3] is used for this purpose.

Algorithm 1.1 algorithm of the coverability tree

```
1:     Label the initial marking $m_0$ as the ’$root$’ and tag it ’$new$’;
2:     while’$new$’ markings exist, do the following:
3:         Select a new marking $m$;
4:         If no transitions are enabled at $m$:
5:             tag $m$ as’$dead - end$’;
6:         If $m$ is identical to a marking on the path from the ’$root$ to $m$:
7:             tag $m$ as ’$old$’ and go to another new marking;
8:         While there exist enabled transitions at $m$:
9:             for each enabled transition $t$ at $m$:
10:                Obtain the marking $m'$ by firing $t$ at $m$;
11:                On the path from $root$ to $m$, If there exists a marking
11.1:                  $m''$ such that: $m'(p_i) > m'' \forall i = 1, 2, \ldots, n$ and
11.2:                  $m' \neq m''$ i.e. $m'$ covers $m''$:
12:                    replace $m'(p_i)$ by$\omega$ wherever $m'(p_i) \geq m''(p_i)$:
13.1:              Introduce $m'$ as a node in the tree, draw an arc with
13.2                   label $t$ from $m$ to $m'$, and tag $m'$ as ’$new$’.
```

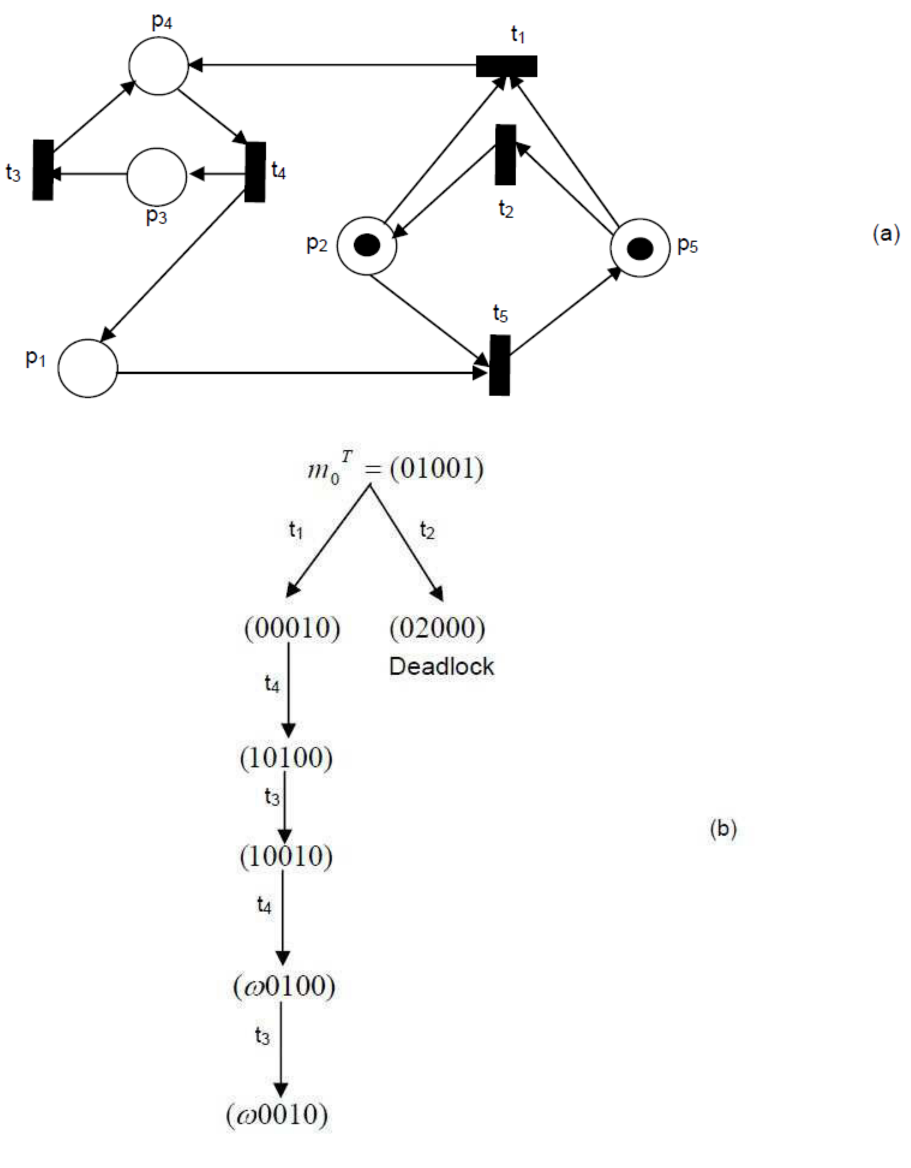


Figure 1.5: (a) Petri net graph, (b) coverability tree construction for (a)

1.5.3 Static analysis method

In systems and control theory, system description is given in terms of a set of differential or algebraic equations. Is it possible to give the Petri net system (static and dynamic) description in terms of some equations? That was the spirit that enabled the development of matrix equations for PN analysis.

Linear algebra analysis makes it possible to study the properties of a network (boundedness, liveness) independently of an initial marking, by calculating a set of vectors (invariant) that make it possible to deduce certain properties. As a result, we will talk about the structural properties of the network. For example, we can say that a network is structurally bounded if it is bounded for any initial marking. In the same way, if for any initial marking, the network is live, it will be said that the network is structurally live [3].

Definition 5. P-invariant: is a victor of places $\vec{Z_n}$ that satisfies the equation:

$$\vec{Z}_n^t \,.\, C_{n,m} = \vec{0_m}.$$

- If $\vec{Z_n}$ is a solution to positive values $\Longrightarrow$the PN is conservative $\Longrightarrow$ The PN is structurally bounded.
- If $\forall i = 1, \ldots, n: \quad Z_i = 1$ $\Longrightarrow$The number of tokens is always fixed in all markings.

Definition 6. T-invariant : is a vector of transitions $\vec{W_m}$ that satisfies the equation:

$$C_{n,m} \,.\vec{W_m} = \vec{0_n}.$$

.

- $\vec{W_m}$ is a T-invariant, if σ is a sequence of firing transitions that corresponds to it: $M \overset{\sigma}{\rightarrow} M$ if M allows to fire σ. It is said that this sequence is repeated.
- if $\exists\ i = 1, \ldots, m:\ W(i) = 0$ if we fire t_i from M, we will not get back to M again.
- if $\nexists \vec{W_m}$ $\Longrightarrow$PN has a dead lock.

1.5.4 Example static analysis

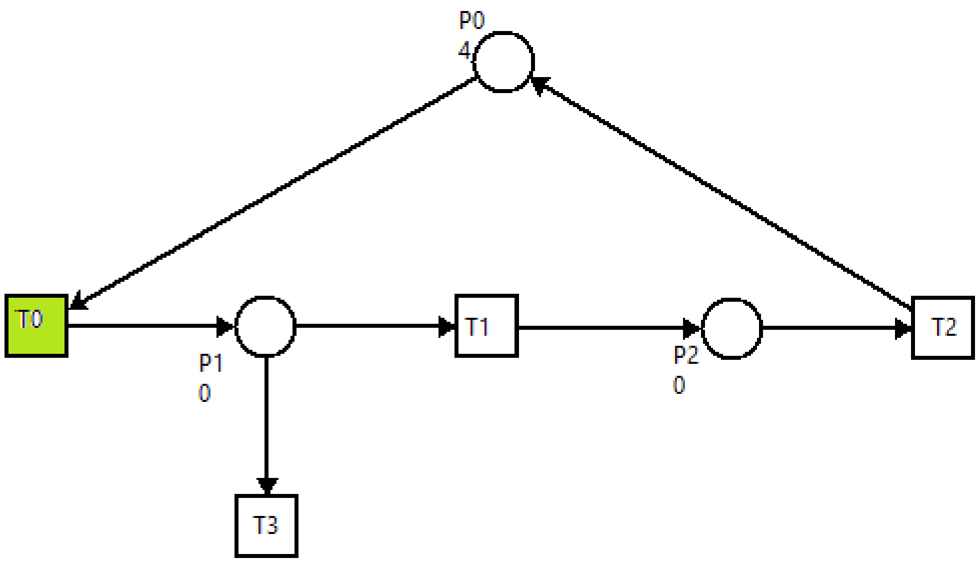


Figure 1.6: Example of Petri net.

The matrix of incidence corresponding to the network on Figure 1.4 is given as follows:

$$C \quad : \quad \left\{ \begin{array}{cccc} -1 & 0 & +1 & 0 \\ +1 & -1 & 0 & -1 \\ 0 & +1 & -1 & 0 \end{array} \right\}$$

Calculating the 2 victors $\vec{Z_n}$ and $\vec{W_m}$.

$$\vec{Z}_n^t \,.\, C_{n,m} = \vec{0_m} \Longrightarrow \left\{ \begin{array}{c} -x_1 + x_2 = 0 \\ -x_2 + x_3 = 0 \\ x_1 - x_3 = 0 \\ -x_2 = 0 \end{array} \right. \Longrightarrow \left\{ x_1 = x_2 = x_3 = 0 \right. \Longrightarrow noP{-}invariant\vec{Z}_n^t$$

$$C_{n,m}.\vec{W_m} = \vec{0_n} \Longrightarrow \left\{ \begin{array}{c} -x_1 + x_3 = 0 \\ x_1 - x_2 - x_4 = 0 \\ x_2 - x_3 = 0 \end{array} \right. \Longrightarrow \left\{ \begin{array}{c} x_1 = x_2 = x_3 \\ x_4 = 0 \end{array} \right. \Longrightarrow \vec{W_m} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

no P-invariant ⇒ the Petri net is non-conservative.

there exists a T-invariant $\vec{W}_m = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$ and $\sigma = t_0, t_1, t_2$ is a sequence of firing transitions that corresponds to it.

1.6 Petri nets as modeling formalism

Petri net is an efficient modeling tool for modeling hybrid system since it can capture properties like concurrency, asynchronous behavior, non-determinism are intrinsic to Petri nets. So here we will introduce Petri net as a modeling tool. In this section, some basic situations are taken which are encountered often during modeling a physical system. This section describes how Petri net handles these real life modeling situations, thus revealing the modeling power and ease of representation of Petri nets.

1.6.1 Sequential execution

Sequential execution poses a precedence constraint among the activities (transitions). In Fig 1.7 transition $t2$ can fire only after the firing of $t1$.

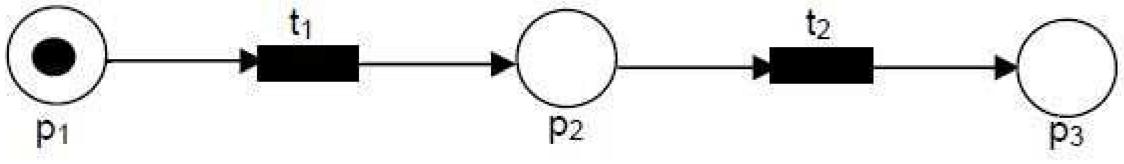


Figure 1.7: Transition $t1$ occurs first and then transition $t2$ occurs.

1.6.2 Synchronization

Petri nets can successfully capture the synchronization mechanism in the modeling phase. In Fig 1.8 transition $t1$ will fire only when the empty input place gets a token. Thus, the three input places of $t1$ are synchronized for the firing of transition $t1$.

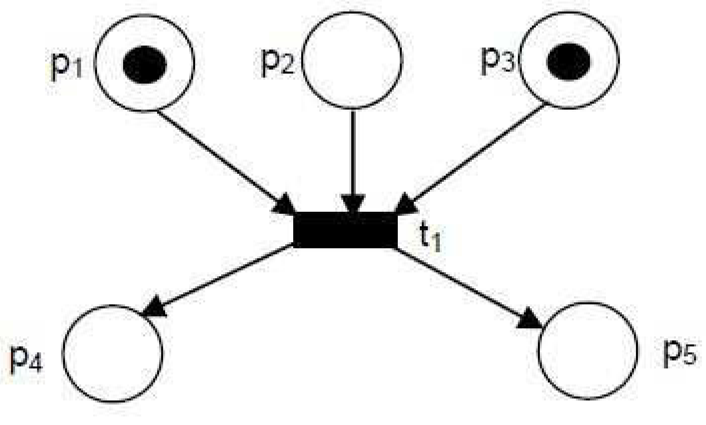


Figure 1.8: Transition t_1 fires when the place P_2 gets a token so that all the input places of transition t_1 have tokens.

1.6.3 Concurrency

In Fig. 1.9 transitions t_1, t_2 and t_3 are concurrent. Concurrency is characterized by the existence of a forking transition that deposits tokens simultaneously in two or more output places. In Fig 1.9 t_0 is the forking transition.

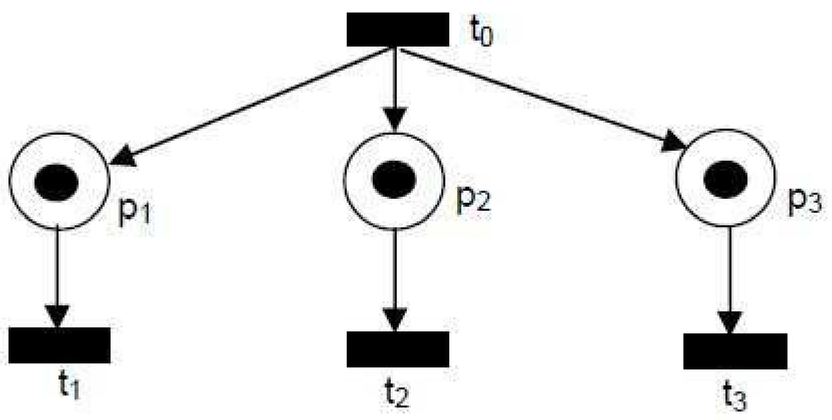


Figure 1.9: Transitions $t1$, $t2$ and $t3$ are concurrent.

1.6.4 Conflict

In Fig 1.10 transitions $t1$, $t2$ and $t3$ are in conflict. All three transitions are enabled but only one can fire at a time. Hence, choice has to be made regarding which transition will be fired. Firing one will lead to the disabling of other transitions.

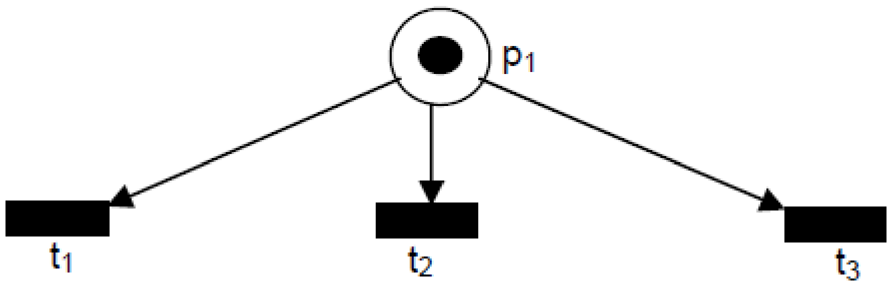


Figure 1.10: Transitions *t*1, *t*2 and *t*3 are in conflict.

1.6.5 Confusion

Confusion occurs when conflict and concurrency co-exist. In such a situation, it is not clear that whether a conflict is needed to be resolved or not, in going to the new state (marking). In Fig 1.11 transitions *t*1 and *t*3 are concurrent whereas transitions *t*1 and *t*2 are in conflict. Also *t*2 and *t*3 are in conflict.

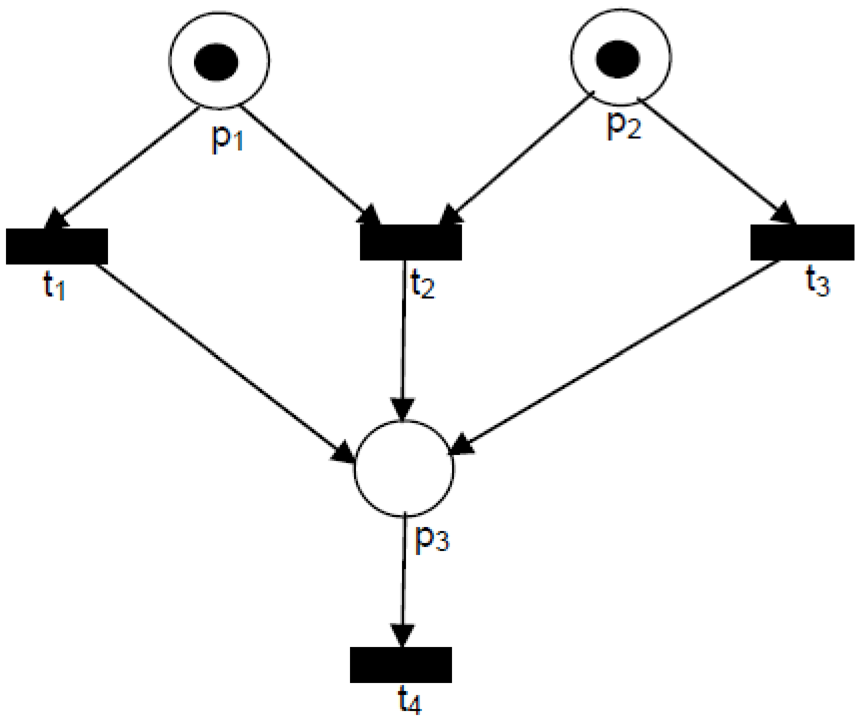


Figure 1.11: Transitions *t*1, *t*2 and *t*2, *t*3 are in conflict but *t*1, *t*3 are concurrent.

1.7 Primitives for Programming

This section describes basic programming constructs in Petri net formalism. This, in turn, will express the modeling power of Petri nets.

1.7.1 Selection (if ? else)

(a) If condition A then do activity X, else do activity Y.

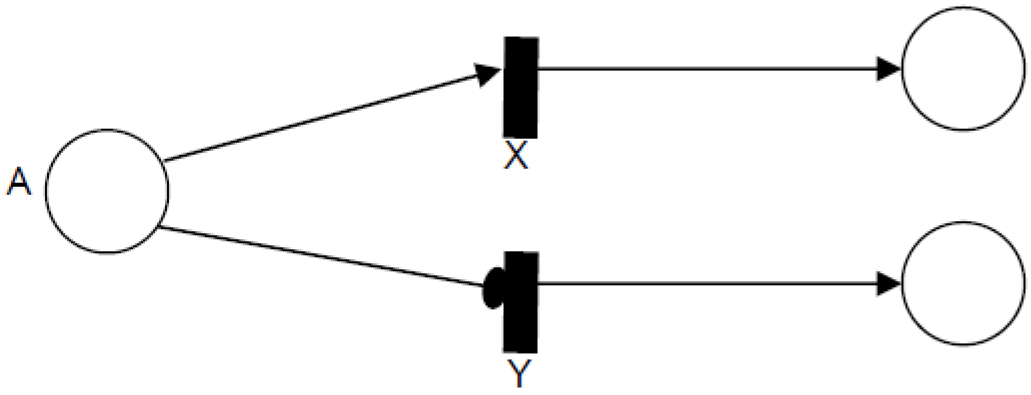


Figure 1.12: If - else condition.

(b) If condition A and condition B hold, then do activity X.

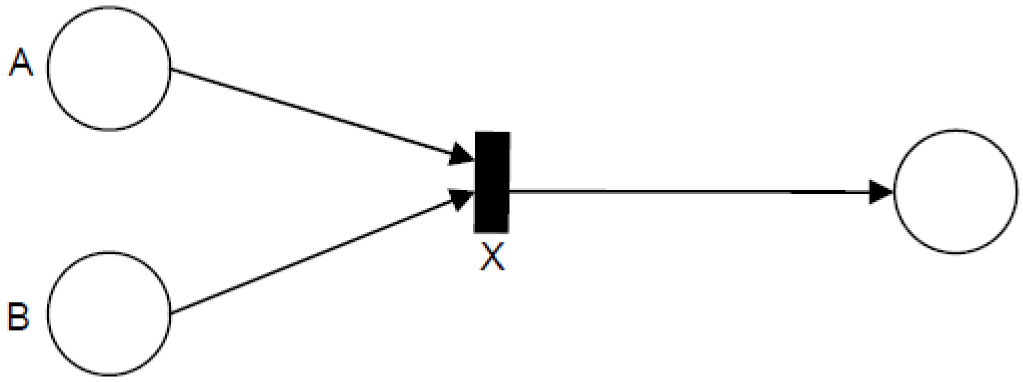


Figure 1.13: If - else with and operator.

1.7.2 Case (Switch) statement

If Case A do activity P, if Case B do activity Q, if Case C do activity R, if Case D do activity S. Switch statement

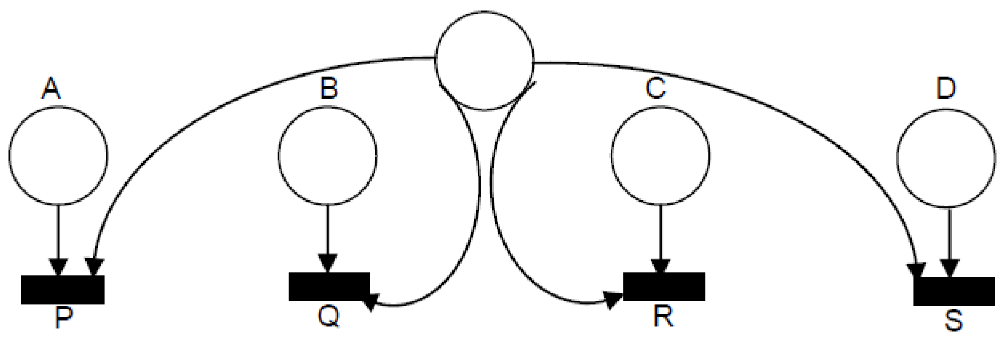


Figure 1.14: Switch statement.

1.7.3 While loop

While condition A holds, do activity X.

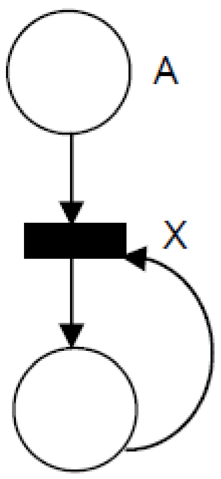


Figure 1.15: While loop.

1.7.4 Repeat (for) loop

For condition A, do activity X.

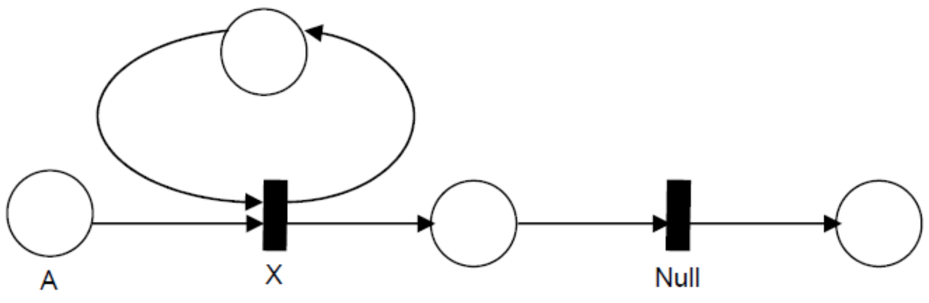


Figure 1.16: For loop.

1.7.5 Precedence

Activity X should precede activity Y.

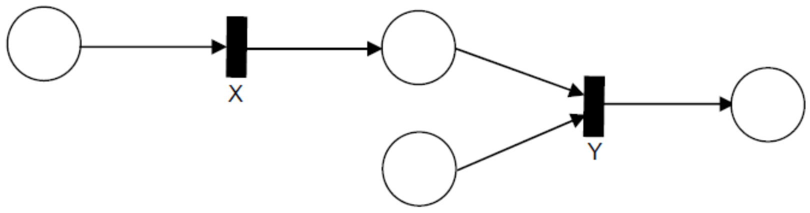


Figure 1.17: Precedence relation.

1.7.6 Timed occurrence

After k seconds do activity X.

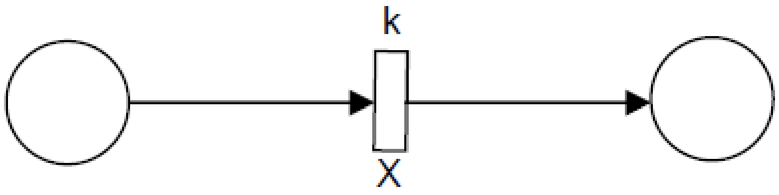


Figure 1.18: Timed transition.

1.8 Conclusion

in this chapter we have seen a general view of Petri net with some informal definitions, we also introduced some of PN's formal definitions, its properties and methods of analyzing it. Then, we have seen Petri net as a modeling tool for the dynamic systems. Finally we have presented some Primitives for Programming Constructs that can be modeled by Petri nets.

Chapter 2

Reconfigurability

2.1 Introduction

Global economic competition and rapid social and technological changes have forced manufacturing to face a new economic objective: manufacturing responsiveness, i.e. the ability of a production system to respond to disturbances which impact upon production goals, and consequently, its ability to adapt to changing market conditions. In the early 1990s the idea of agile production systems was pursued, enabling short changeover times between manufacturing different products. Since the end of the 1990s the trend is towards reconfigurable manufacturing systems (RMSs). i.e. systems that are capable of being quickly adapted to changing market requirements by providing the needed functionality and capacity at any time. Taking industry as a vantage point, industry 4.0 domain presents an example of the reconfigurable systems.

Despite the power of PN and the variety types of systems that can be modeled by the PNs (low or high level), but the PNs have a rigid structure and can not capture notions of reconfiguration, for example a PN model does not admit adding, removing and modifying nodes and arcs to model system with dynamic structures. Reconfigurable Petri nets have been applied in various application areas where complex coordination and structural adaptation at run-time are required (e.g. mobile ad-hoc networks [5], communication spaces [6], concurrent systems [7], flexible manufacturing systems [8], reconfigurable manufacturing systems [9], etc). They improve the expressiveness of Petri nets as they increase flexibility and change while allowing the transitions to fire. This greater expressiveness yields rich models with very large state spaces. The state space of the model is even more complex because states are not capturing only the change of the markings but also structure and connectivity changes [10].

2.2 Reconfigurable systems

Reconfigurable systems are those architectures that have the capability to adapt themselves to a given application, providing some kind of hardware specialization to it. Through this adaptation, they are expected to achieve great improvements, in terms of performance acceleration and energy savings, when compared to general purpose, fixed instruction set processors.

The fourth industrial revolution requires higher capabilities of changeability and reconfigurability, that's why we have chosen industry 4.0 as an example for our studies. We will also see another example in domain of simulation called Collaborative Virtual Environment (CVE).

2.2.1 Industry 4.0

The three past industrial revolutions were driven by technical innovations: the steam-powered machine for the first one, the division of labor at the beginning of the 20th century, and the beginning of automation in manufacturing in the latter 20th century. Experts see the upcoming industrial revolution triggered by the Internet, which permits to create communication between human but also with machines in Cyber Physical System (CPS). The demand for customized product is growing, forced companies to transform their organizational structure then also the production. This will be the Industry 4.0, in reference to a fourth industrial revolution, where the numeric or digital will be present everywhere as oxygen in our world.

The industry 4.0 focuses on the installation of intelligent product and production processes. In the future, manufacturing needs to be flexible and rapid as possible to answer to the needs of customers and here where reconfigurability takes a place. Some examples for Industry 4.0 are machines, which can predict failures and trigger maintenance processes autonomously, which react to unexpected changes in production.

The main features of industry 4.0 that concerns our studies are:

- Service Orientation.
- Modularity: flexible adaptation of smart factories to changing requirements by replacing or expanding individual modules.

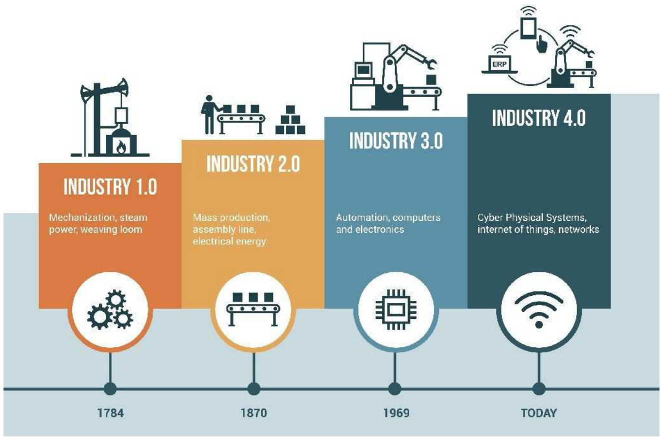

INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks
ERP
1784
1870
1969
TODAY

Figure 2.1: Industry 4.0

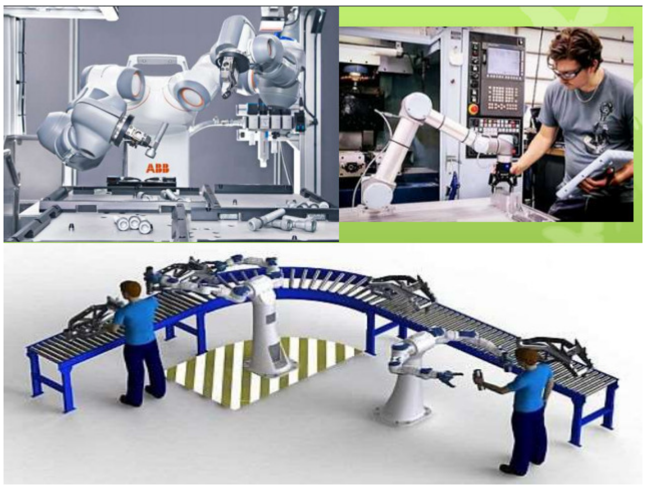


Figure 2.2: Industry 4.0

2.2.2 Example: DJINN

The DJINN multimedia-programming framework (Mitchell, et al, 1998) from the Distributed Systems Laboratory at the University of London is an example of reconfigurable system in the domain of multimedia communications. The aim is to guaranty quality of service (QoS) in communications through a distributed network of computers or mobile devices. In this network, nodes may be added or deleted dynamically triggering a reconfiguration. A configuration is expressed with paths between components. Reconfiguration of this system is done in two phases. A setup phase where the new components are created and an integration phase where they are started and connected to the system. Depending on the resources offered, the transition between the two configurations is more or less smooth (it depends on if the two configurations can live together in the system or not). The reconfiguration consists in the adding or removing of components and the modification of the connection between them. The decision of the reconfiguration and its control are centralized.

2.2.3 Collaborative Virtual Environment (CVE)

An example of a computer system using reconfiguration was given in [11]. This is a simulation environment battlefields for the US military using a computer system distributed on several computers. The simulation environment called CVE (Collaborative Virtual Environment) is built in a modular way, some of the modules are called VE (Virtual Environment), the others are utilities related to the simulation [11].

The reconfigurable system is the CVE, whose different modules can be loaded, unloaded or moved during system operation. The Bullpen module managing the reconfiguration is also part of the reconfigurable system.

Two scenarios requiring reconfiguration are proposed in [11]. The first occurs when a machine breaks down. It is then necessary that this failure has the least effect on the continuation of the simulation. A second use of the reconfiguration to ensure the balance of the load of the different machines during the evolution of the simulation.

2.2.4 Case of study

In order to illustrate the application of the proposed extension, we use a simple RMS example inspired from the one presented in [12]. This RMS is composed of a single reconfigurable machine M which has two possible configurations. In its first configuration C_1, it produces two types of products A and B. In its second configuration C_2, the machine produces a third product C besides the two former ones. The RMS uses a transport device TL to bring raw materials RawM to the machine, and to pick up the products. Figure 2.4.a (respectively Fig. 2.4.b) shows the RMS in its first configuration (respectively its second configuration). The interpretation of the nodes in the two graphs is illustrated in Table bellow.[1]

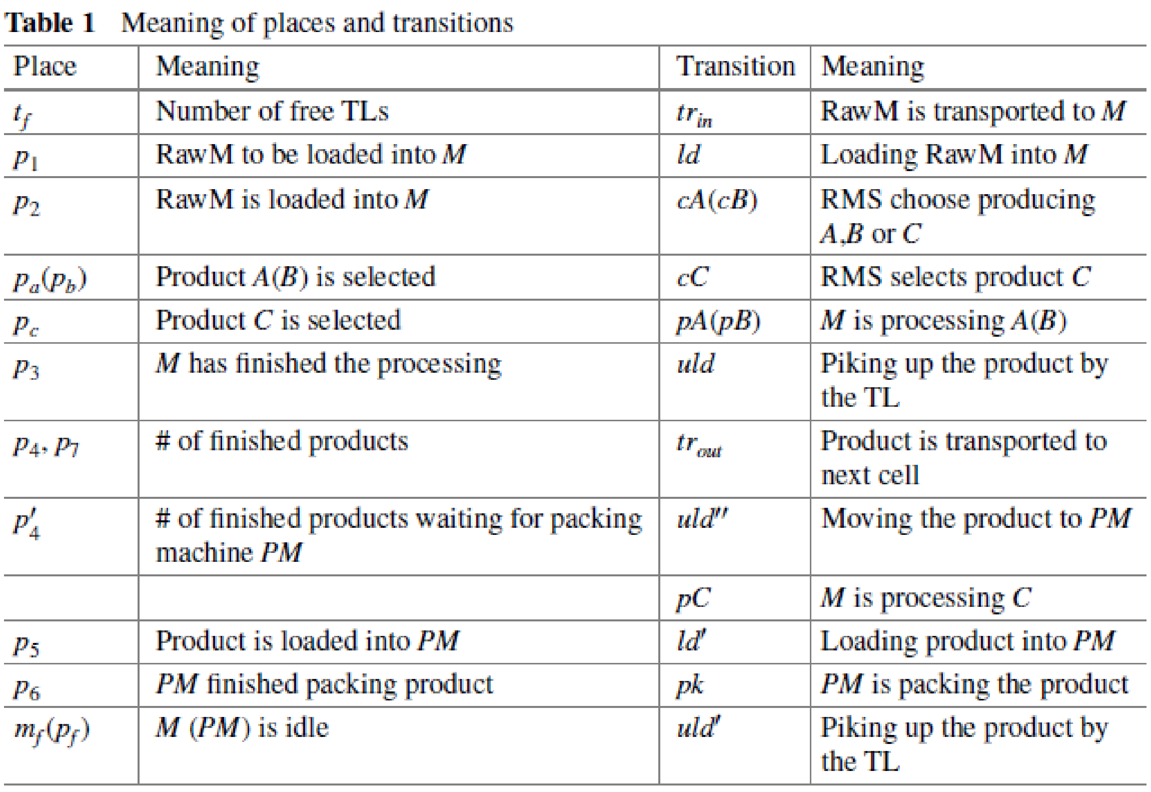
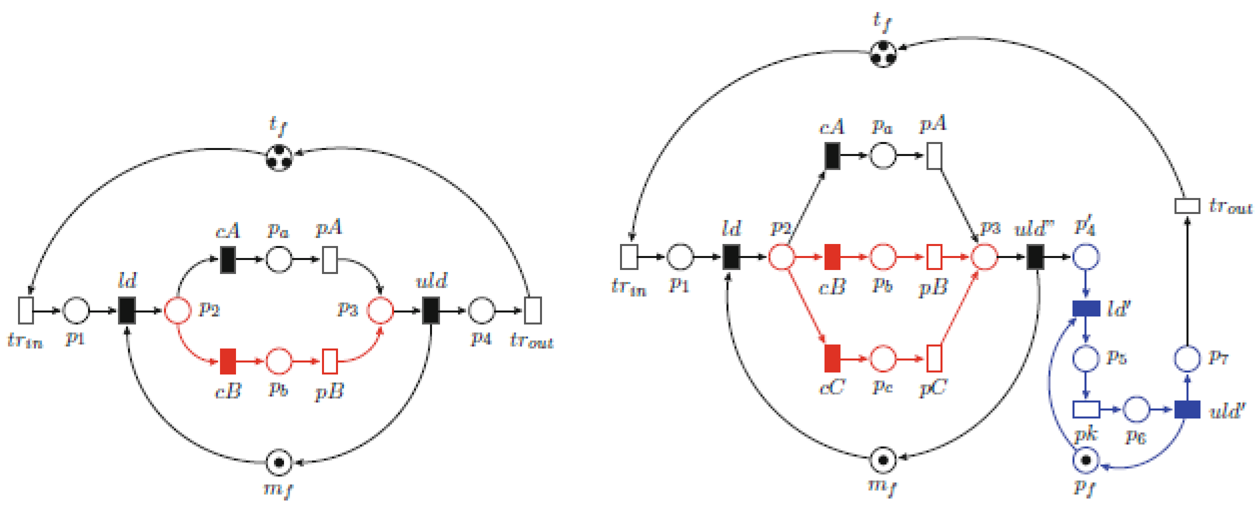


Table 1 Meaning of places and transitions

Place	Meaning	Transition	Meaning
t_f	Number of free TLs	tr_{in}	RawM is transported to M
p_1	RawM to be loaded into M	ld	Loading RawM into M
p_2	RawM is loaded into M	$cA(cB)$	RMS choose producing A,B or C
$p_a(p_b)$	Product $A(B)$ is selected	cC	RMS selects product C
p_c	Product C is selected	$pA(pB)$	M is processing $A(B)$
p_3	M has finished the processing	uld	Piking up the product by the TL
p_4, p_7	# of finished products	tr_{out}	Product is transported to next cell
p'_4	# of finished products waiting for packing machine PM	uld''	Moving the product to PM
		pC	M is processing C
p_5	Product is loaded into PM	ld'	Loading product into PM
p_6	PM finished packing product	pk	PM is packing the product
$m_f(p_f)$	M (PM) is idle	uld'	Piking up the product by the TL

Figure 2.3: Meaning of places and transitions[1].



(a) RMS model (b) New RMS model

Figure 2.4: Change in Configuration [1].

2.3 R-systems common points

2.3.1 Objectives

- Adaptation to demand: the system must be able to meet the needs for which it has been put in place. If these needs are not satisfied, we must find a new configuration to continue to meet the demand or to respond to a new request.
- Quality: It is not always enough to answer at the request, it is still necessary to answer it by following a set constraints.
- Optimize the use of system resources: This need is reflected in the project for reconfigurable robots. The goal is to make functions, so powerful with a minimum of resources. Reconfiguration makes it possible.
- Mass Customization: Like 3-D printing and advanced modularity concepts, where every user can intimately and radically personalize a system to meet a unique need or whim.
- Reduction of non-recurring engineering expenses: Sometimes it is easier to personalize a reconfigurable component than to build one from scratch.
- Economy through inventory collapse: Flexibility can dramatically reduce inventories, since a small number of reconfigurable components can be used to replace many customized components [13].

2.3.2 Architecture of R-systems

The architecture of a reconfigurable system is defined as the set of components of the system as well as the potential interactions between them.

2.3.3 Configuration of R-systems

The configuration of a reconfigurable system describes the use of architectural elements in order to meet the needs set in the system in the form of quality of service.

2.3.4 Reconfiguration of R-systems

The reconfiguration of a reconfigurable system consists in modifying the structure or the functionality of the system. The reconfiguration can be carried out in several steps if the current state of the system (product position and resource mode for a production system) is not acceptable by the new configuration.

2.3.5 Reconfiguration process

The process of reconfiguring a reconfigurable system starts when the target set for the current configuration is reached or when the current system configuration no longer meets these same goals. When the decision to launch the process of reconfiguration is taken, this process takes place in several phases, a new configuration is produced that meets the objectives set is sought. A path must be determined to achieve this configuration, it is necessary to place the system in an acceptable state for the new configuration. Then the reconfiguration is set to achieve the new operation that meets the objectives set in terms of of quality of service.

2.4 Reconfigurable PN

Reconfigurable Petri nets are class of high level Petri nets that can dynamically change their own structures by rewriting some of their components thus supporting dynamic changes within workflow systems. A reconfigurable net is a very natural extension of a Petri net.

The characteristic feature of reconfigurable Petri nets, 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 formalisms to model dynamic software that is executed in dynamic infrastructures.

dynamic changes in reconfigurable Petri nets:

- Topology changing.
- Appearance of new components.
- Disappearance of components.

2.5 conclusion

In this chapter, we have introduced a class of high level Petri nets called reconfigurable Petri nets, which they have the ability to change their structure dynamically based on reconfiguration rules. in the next chapter we will try to make a tool that can analyze this kind of Petri net's behavior.

Chapter 3

Analysis and conception

3.1 Introduction

We have seen the details that suits the needs for modeling a dynamic systems with changeable structures. Our goal is the realization of a tool for modeling and analyzing reconfigurable Petri nets. It will be necessary to follow a process of development consisting of several stages in order to obtain a reliable software, which corresponds to our needs in a limited duration. This chapter consists of three sections: (i) first, section of analysis needs, (ii) second, global design where we show an abstract architecture view and functionality of our tool that meets our needs, (iii) finally, third one that represents the details of our tool using the class diagram to show how this tool works.

3.2 RPN Formal definition

Definition 7. A reconfigurable Petri net is a structure N=$<G$,g_0,$R>$:

- R = { G_1,..., G_n} is a finite set of reconfigurations which each one of them is represented as a PN.
- g_0 is the initial configuration.
- $R = \{r_1,...,r_h\}$ is a finite set of rewriting rules.

A rewriting rule $r \epsilon R$ is a structure $r = (G_s,G_t,M)$ where:

- G_s is the current configuration.
- G_t is the target configuration.
- M is the required minimum marking so that the rule can be enabled.

The firing policy of transitions is identical to the Petri net obtained by discarding the non existing places .This Petri net is called a configuration of reconfigurable net. As long as no structure modifying rule take place, the reconfigurable net behaves exactly identical to this Petri net. Structure modifying rules produce a structure change in the net by removing existing places,transitions,and/or arcs, and creating new ones thus moving the system from one configuration to another one. Roughly speaking a reconfigurable net can be seen as bunch of Petri nets (its configurations) which correspond to the various *modes* of operation of the system. The structure modifying rules allow to switch from one mode of operation to another one. Thus a system modelled by a reconfigurable net has the ability of dynamically change its own structure when certain conditions are met. For instance if the content of some places becomes too large (there is large amount of work cases waiting for being processed) one can duplicate the output transitions of this place, technically we replace this place by a new one having twice as many output transitions playing the same role as the output transitions of the original place [14].

3.3 Analysis of a RPN

We can imagine a reconfigurable Petri net with configurations C_0 fig.3.1, C_1 fig.3.2 and a reconfiguration rule $r = (C_0, C_1, M_r)$ $M_r(P_1, P_2, P_3, P_4, P_5) =< 1, 0, 0, 0, 1 >$.

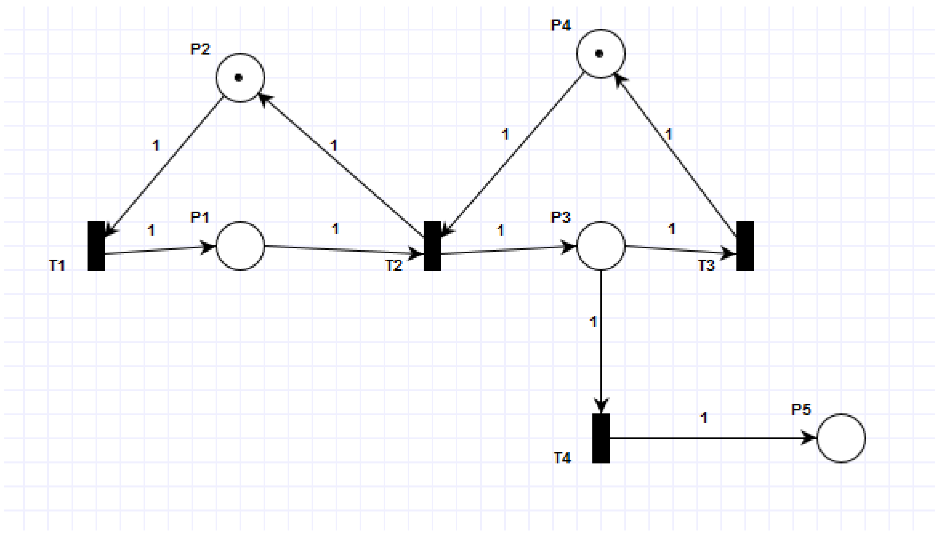


Figure 3.1: Initial configuration C_0.

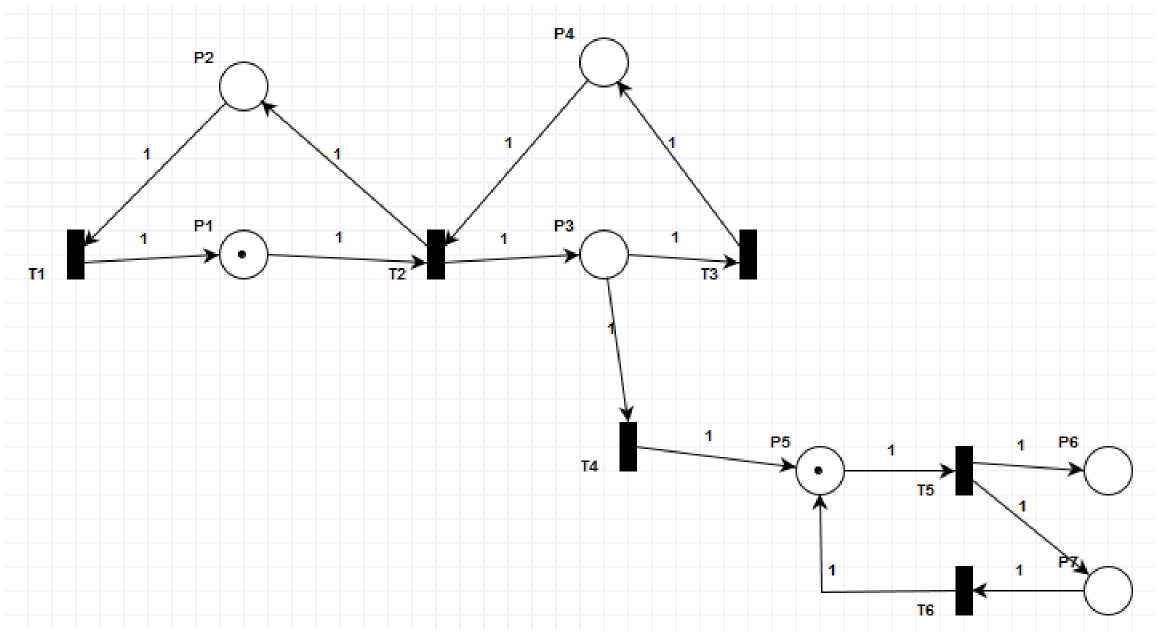


Figure 3.2: configuration C_1.

the coverability graph of this reconfigurable Petri net is shown bellow

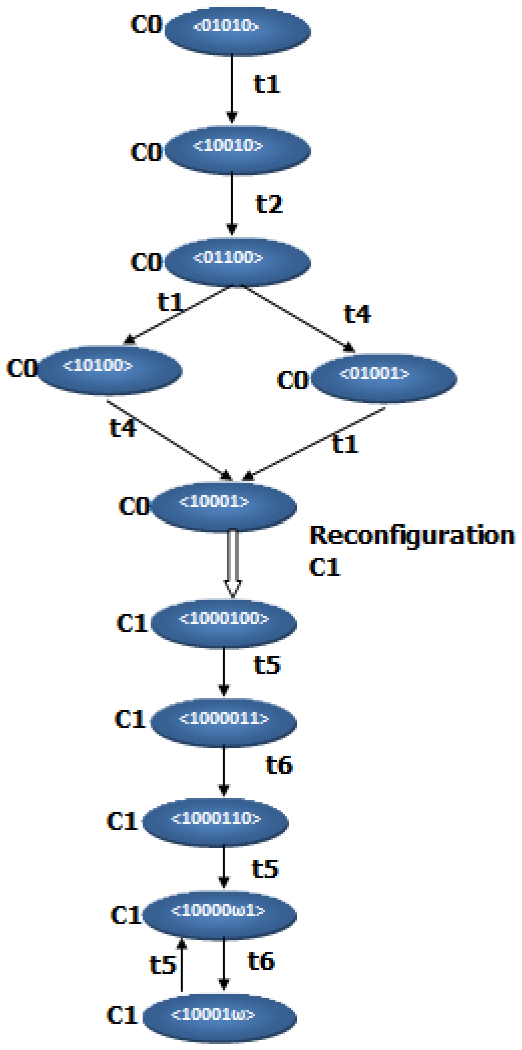


Figure 3.3: Coverability graph.

3.3.1 Marking graph

as inputs we need:

- finite set of configurations.
- initial configuration.
- finite set of rules.

Algorithm 3.1 algorithm marking graph for RPN

```
1:initial configuration and initial marking are set as:
1.1 a root of the graph.
2: do:
3:      based on the current configuration and its marking:
4:          get all the fireable transitions.
5:          fire each one of the fireable transitions to get a new making.
6:              for each obtained state:
7:                  if the state obtained does not already exist on the graph:
8:                      add the state to the graph.
9:                  a test is applied to see whether there is an applicable
10.1:                       rule for the current marking and configuration.
11:                 if a rule is enabled:
12:                     change in configuration to the target of the rule.
13:while there is non-visited state.
```

Remark 1. In this algorithm, we do not consider unbounded RPNs.

3.4 Conception

Inputs and outputs

In which we give an abstract solution that meets our needs and specifies the functionality of the tool that we seek to realize.

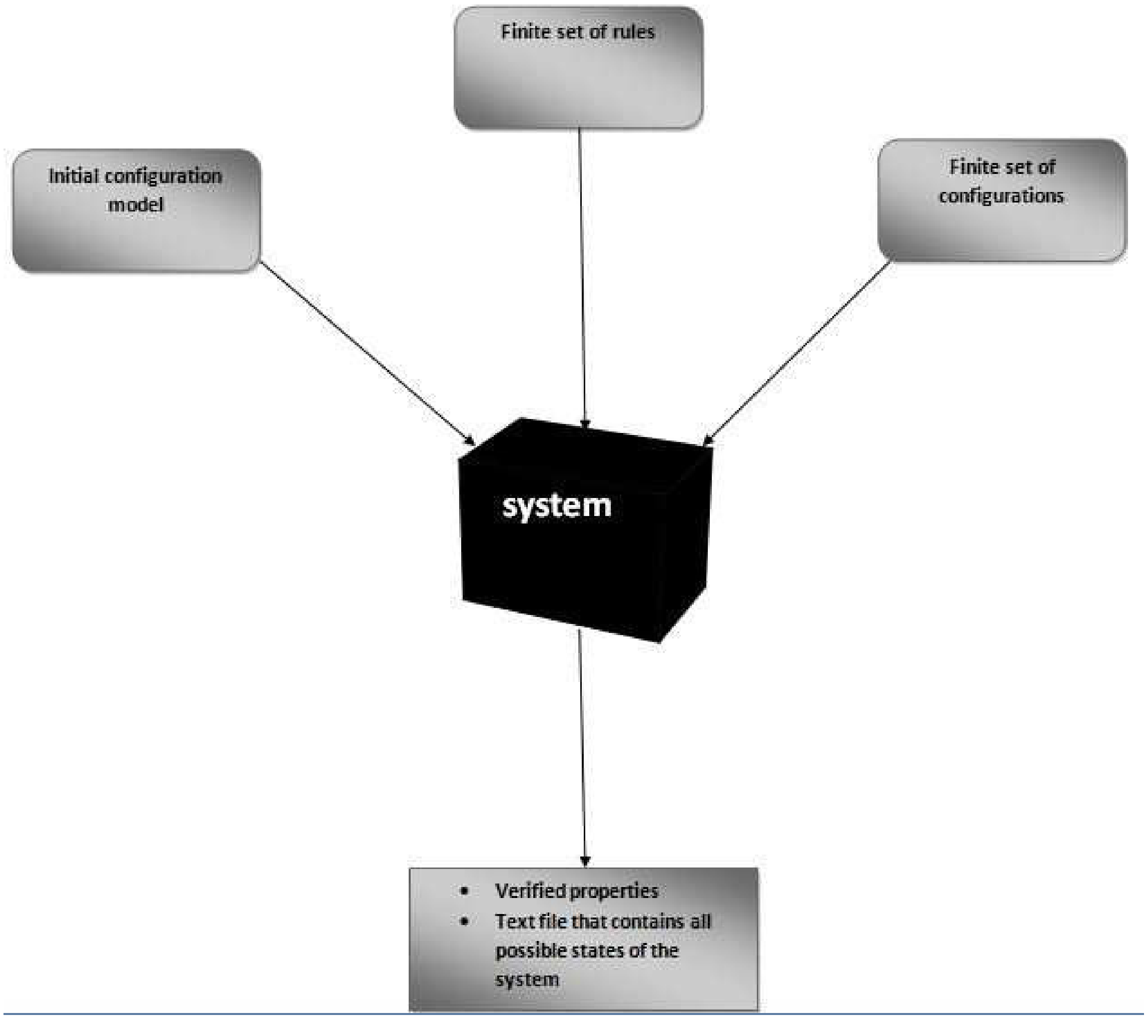

Finite set of rules
Initial configuration model
Finite set of configurations
system
Verified properties
Text file that contains all possible states of the system

Figure 3.4: Inputs and outputs of the tool.

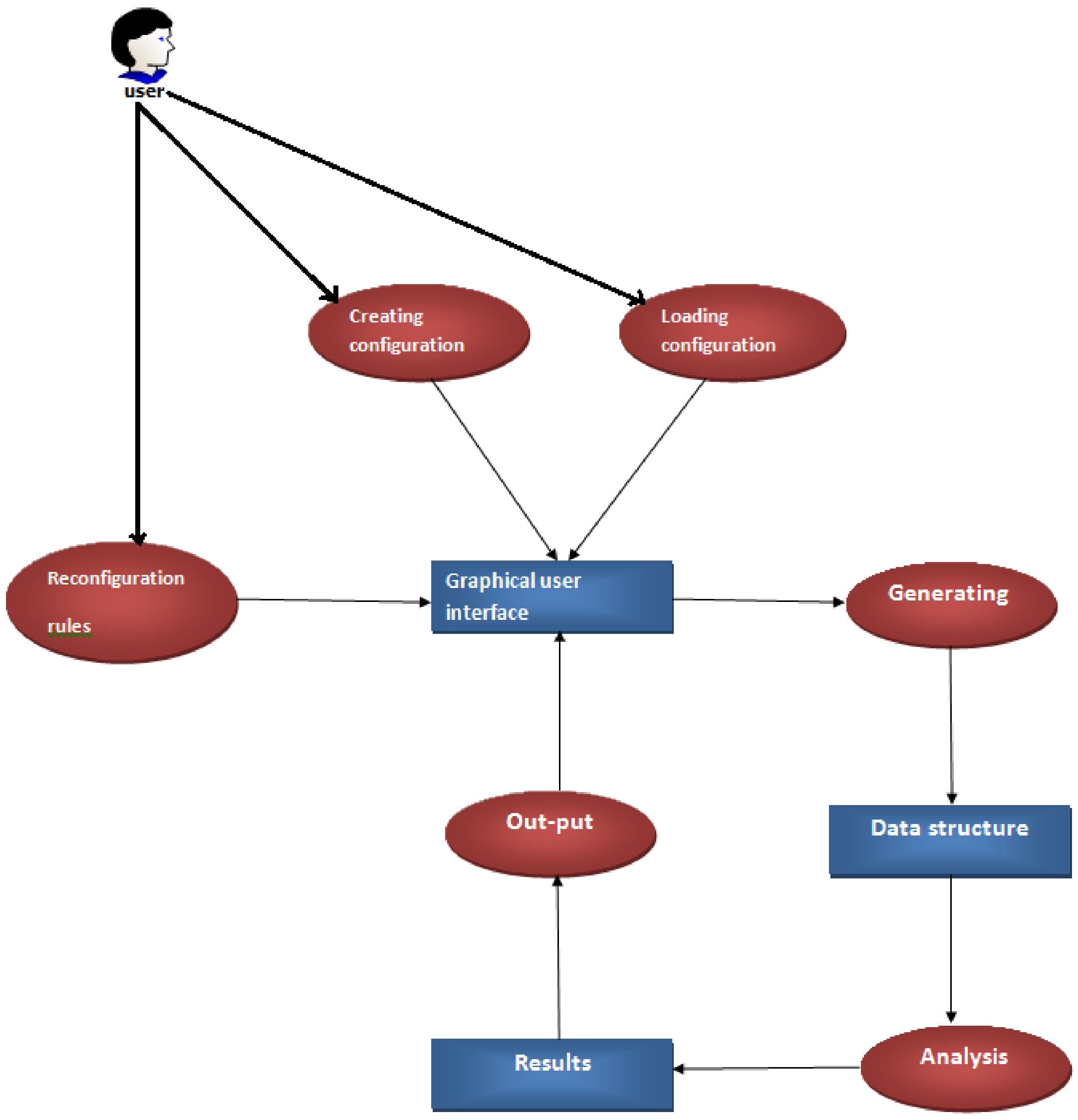


Figure 3.5: Global view of the conception.

- User needs to create Petri net models (configurations) using PIPE, which can be called from the the graphical user interface of the tool or loading configurations, that are saved as 'xml' files.
- User should define reconfiguration rules.

when the inputs are set, the tool is ready to start analyzing the reconfigurable Petri net and this happens after generating a data structure. The result will be shown to the user in the graphical user interface.

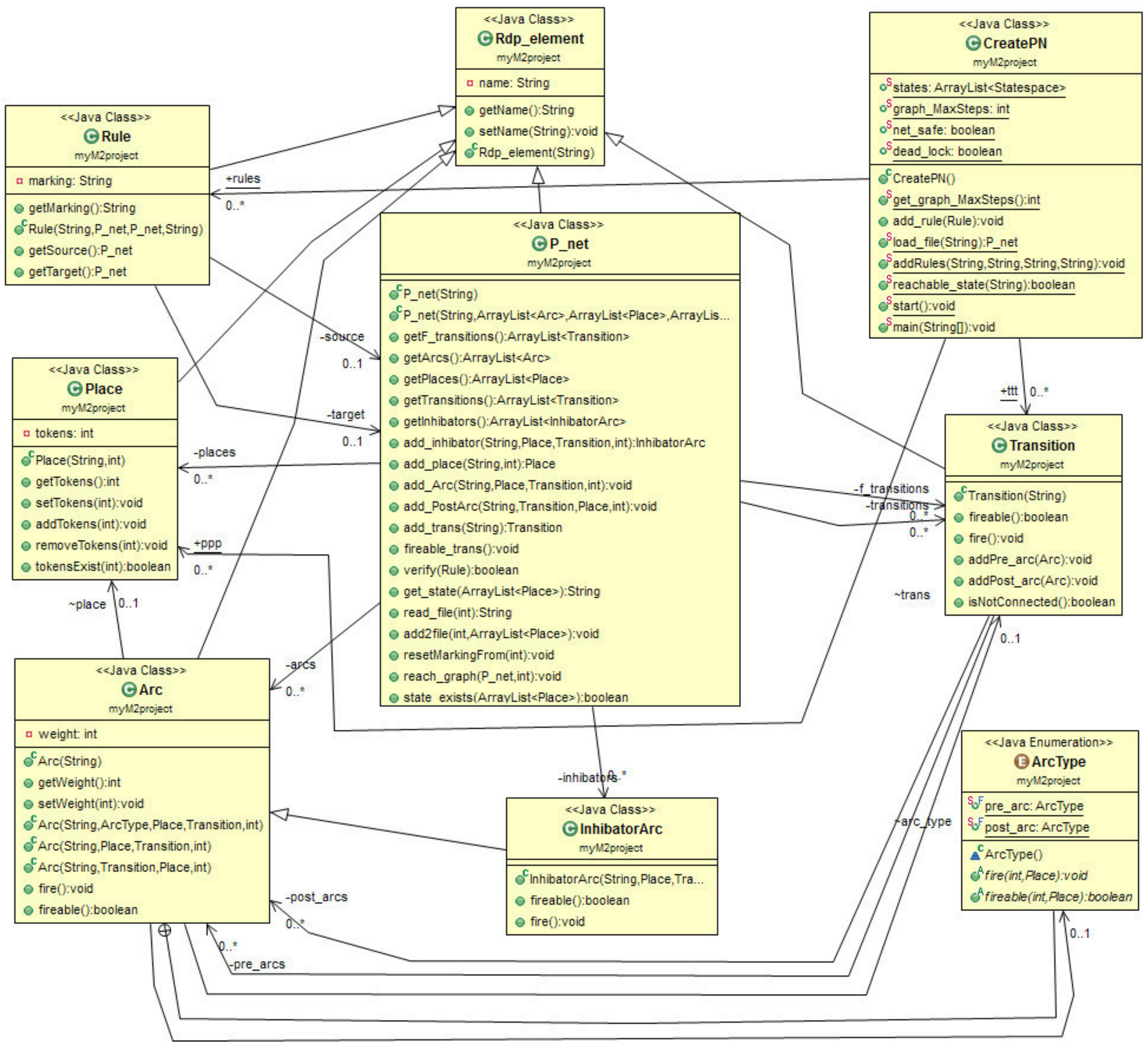


Figure 3.6: Class diagram model.

3.5 Conclusion

In this chapter, we have presented an analysis and conception that determine the technical solutions that satisfies our needs. We move on to the next chapter in which we will implement this project. The next chapter, we will explain the part of realization of our tool and finally represent the tool after the implementation.

Chapter 4

Realization

4.1 Introduction

After the analysis and design stages, we start the realization of our tool. This step allowed us to develop everything we studied in the previous chapter. This chapter consists of two parts: (i) first part which represents the various computer tools and development languages that we have chosen to begin this work, (ii) second part that explicitly represents the tool realized by showing its interface.

4.2 Used tools

During the implementation phase of this project, it was necessary to resort to certain areas of programming and we had the opportunity to familiarize ourselves with various technical and software development tools that will be presented below.

4.2.1 PIPE

Is an open source project implemented in Java initially launched in 2002 / 2003 as a project of the MSc team Project Group at the Department of Computer Science at the Imperial College London turns on UNIX-like platforms, Windows, MacOS X, it is in its evolved version 4.3.0. PIPE offers a full suite of analytical tools for verification of structural properties and dynamics of these nets and produces the statistics of performance and classification.

4.2.2 XML

Extensible Markup Language (XML) is a markup language that has been designed to store and transport data. The most important thing about this language is that it has designed to be both human readable and machine readable. We used this language to store and load our files from PIPE for our tool.

4.2.3 NetBeans IDE 8.1

NetBeans is an Open Source IDE by Sun in June 2000 under CDDL (Common Development and Distribution License) and GPLv2. In addition to Java, NetBeans provides native support for various languages such as C, C ++, JavaScript, XML, Groovy, PHP and HTML, or others (including Python and Ruby) by adding plugins. It offers all the facilities of a Modern IDE (color editor, multi-language projects, refactoring, graphic editor interfaces and web pages). Compiled in Java, NetBeans is available under Windows, Linux, Solaris (on x86 and SPARC), Mac OS X or an independent version of operating systems (requiring a Java virtual machine). A Java environment JDK is required for Java developments. NetBeans is a platform that allows the development of specific applications (Swing library (Java)).

4.2.4 JDOM

Is an open source library for manipulating XML files in Java. It integrates DOM and SAX, and supports XPath and XSLT. It uses analyses external syntax for building documents.

4.2.5 Java FX

With the appearance of Java 8 in March 2014, Java FX becomes the creation tool of the GUI toolkit for the Java language, the development of its predecessor Swing being abandoned (except for bug fixes). Java FX is an evolution of Java, it allows to create RIA (Rich Internet Applications), that is, applications containing videos, music, very interesting graphic effects, etc. Java FX allows you to create web applications, applications for your desktop and for your mobile phone.
Java FX is now a pure Java API (the scripting language specific that has been a time associated with Java FX is now abandoned). Java FX contains a variety of tools, including audio and video media, graphics 2D and 3D, Web programming, multi-threaded programming etc. The Java FX SDK now is integrated into the standard JDK Java SE, there is no need to realize specific installation for Java FX.

4.2.6 LyX

LyX is an open source document publisher based on the latex system. In contrary to most word processing tools that follow the paradigm WYSIWYG (what you see is what you get), LyX has an approach WYSIWYM (what you see is what you mean) or what appears on the screen is only an approximation of what will appear on the page. We chose LyX for the report and presentation of our project. The power and flexibility of LaTeX that it offers and facility for writing and document processing quickly. It should also be noted that knowledge of LaTeX tag language is not needed for basic use.

4.3 Implementation

After following several stages of development, we implement a tool with a graphical user interface that allows us to model and analyze the reconfigurable Petri nets.

4.3.1 The graphical user interface of the tool

The figure 4.1 represents the interface of our tool. The user can utilize PIPE (external program to create and simulate configurations), import source and target configurations, add reconfigurable rules, choose which properties to verify and finally verify them.

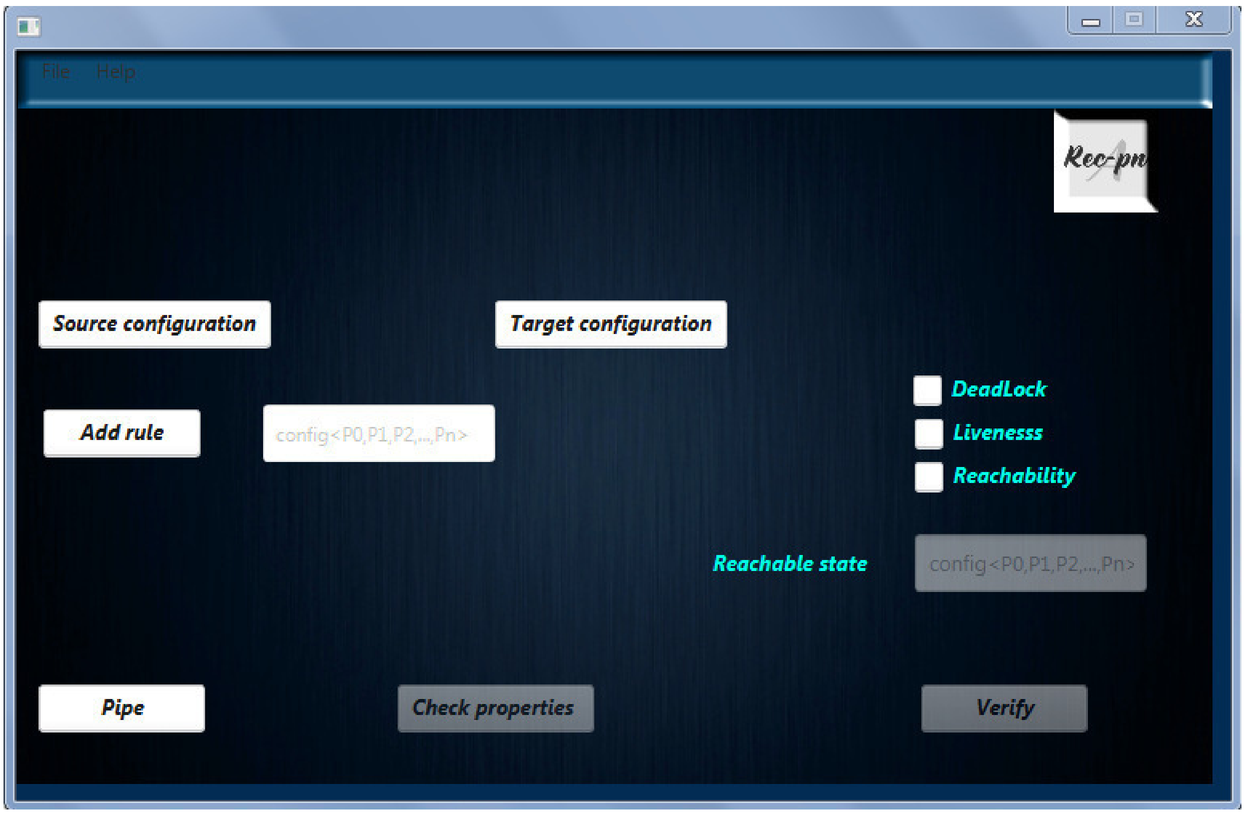


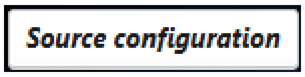
Figure 4.1: The main interface of the tool.

4.3.1.1 The initial components:

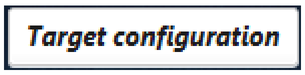
- Enabled components.
- Disabled components.
- Invisible components.

4.3.1.2 Enabled components:

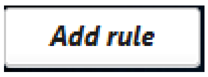
- Source configuration button.



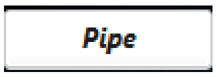
- Target configuration button.



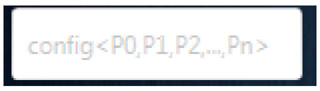
- Add rule button



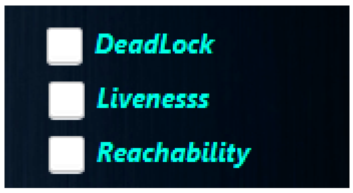
- Pipe button.



- Rule text field.



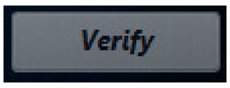
- property checkBox.



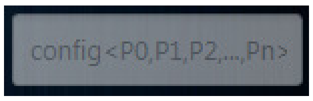
4.3.1.3 Disabled components:

Disabled components are set to control the use of this tool.

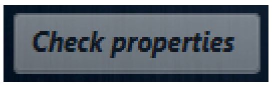
- Verify button



- State text field.



- Check properties button.



4.3.1.4 Unvisible components:

Figure 4.2: to indicate that the input is set right.

Figure 4.3: to indicate that the input is set wrong.

Figure 4.4: to indicate that the state is reachable .

4.3.2 Functionality

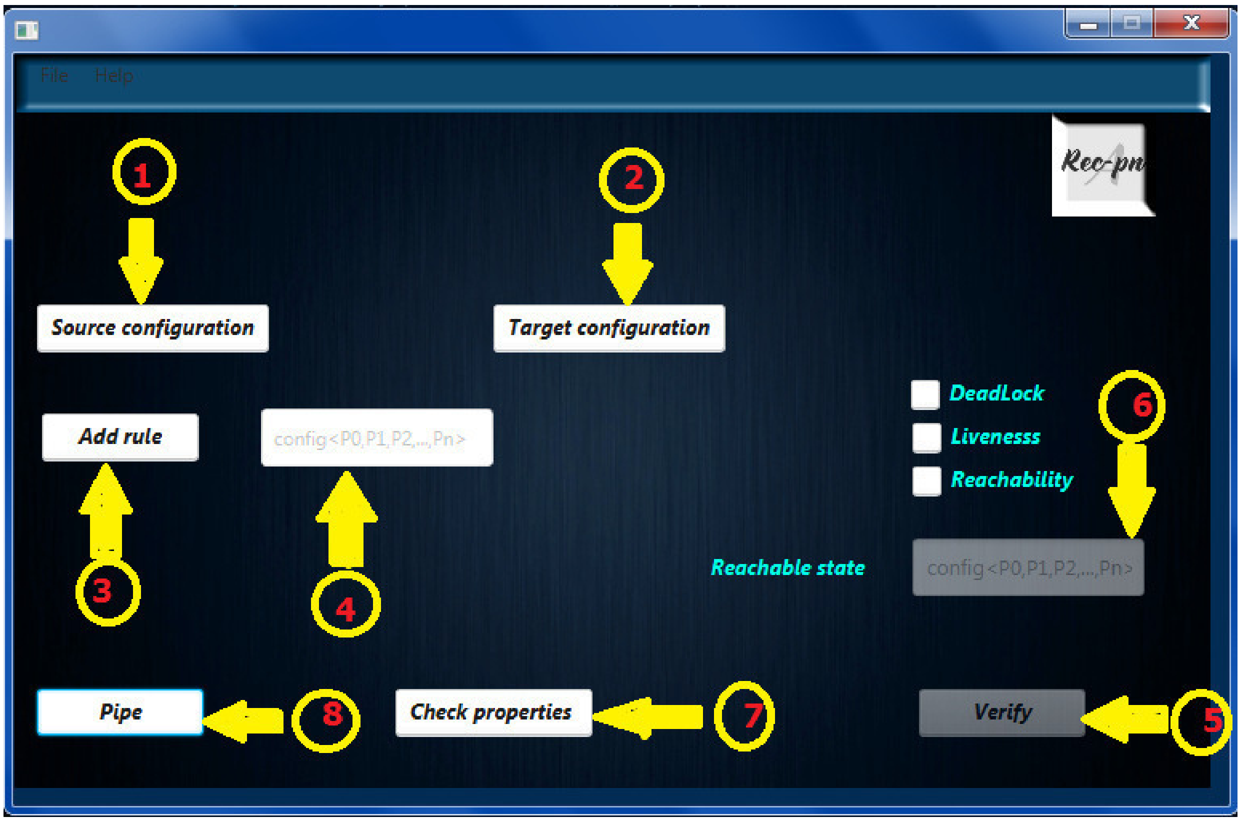


Figure 4.5: functionalities of the tool.

Here we will explain some of our GUI components functionality:

1. Source configuration button: to import source configuration file '.xml'. It will open a file chooser window as depicted in the figure below.

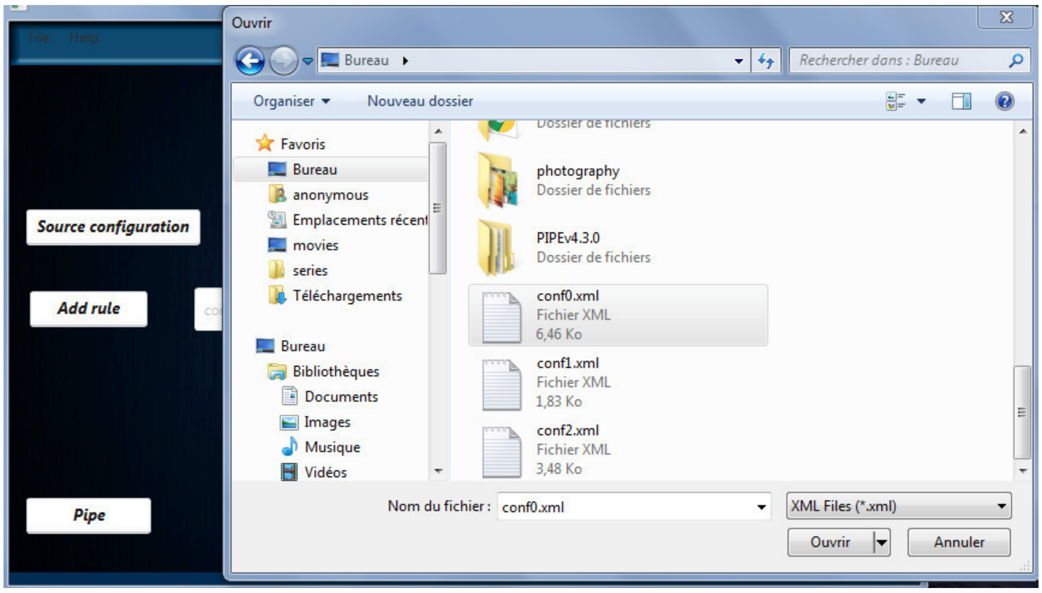
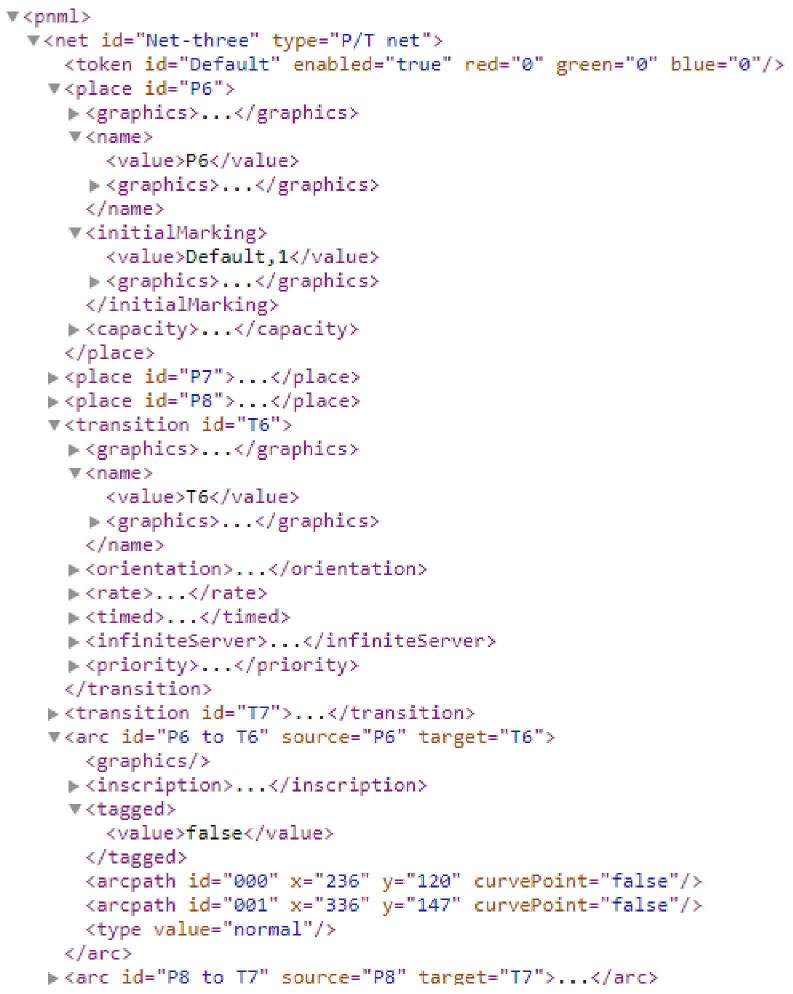


Figure 4.6: Xml file chooser.

When the file is chosen a new graphical component will appear to confirm whether the operations works properly.

2. Target configuration button : same thing as Source configuration button.



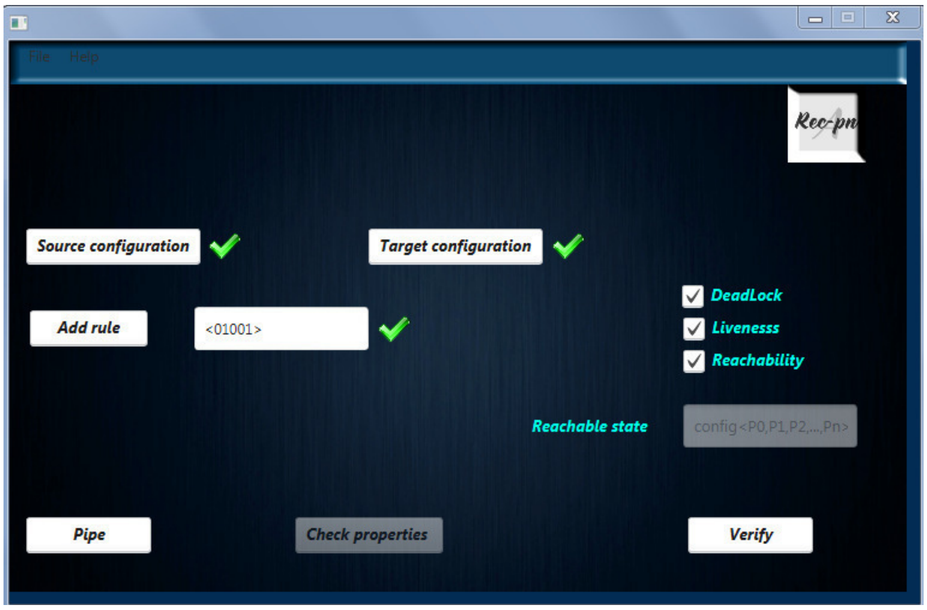
```
▼<pnml>
  ▼<net id="Net-three" type="P/T net">
     <token id="Default" enabled="true" red="0" green="0" blue="0"/>
    ▼<place id="P6">
      ▶<graphics>...</graphics>
      ▼<name>
          <value>P6</value>
        ▶<graphics>...</graphics>
        </name>
      ▼<initialMarking>
          <value>Default,1</value>
        ▶<graphics>...</graphics>
        </initialMarking>
      ▶<capacity>...</capacity>
     </place>
    ▶<place id="P7">...</place>
    ▶<place id="P8">...</place>
    ▼<transition id="T6">
      ▶<graphics>...</graphics>
      ▼<name>
          <value>T6</value>
        ▶<graphics>...</graphics>
        </name>
      ▶<orientation>...</orientation>
      ▶<rate>...</rate>
      ▶<timed>...</timed>
      ▶<infiniteServer>...</infiniteServer>
      ▶<priority>...</priority>
     </transition>
    ▶<transition id="T7">...</transition>
    ▼<arc id="P6 to T6" source="P6" target="T6">
        <graphics/>
      ▶<inscription>...</inscription>
      ▼<tagged>
          <value>false</value>
        </tagged>
        <arcpath id="000" x="236" y="120" curvePoint="false"/>
        <arcpath id="001" x="336" y="147" curvePoint="false"/>
        <type value="normal"/>
     </arc>
    ▶<arc id="P8 to T7" source="P8" target="T7">...</arc>
```

Figure 4.7: xml file of PN.

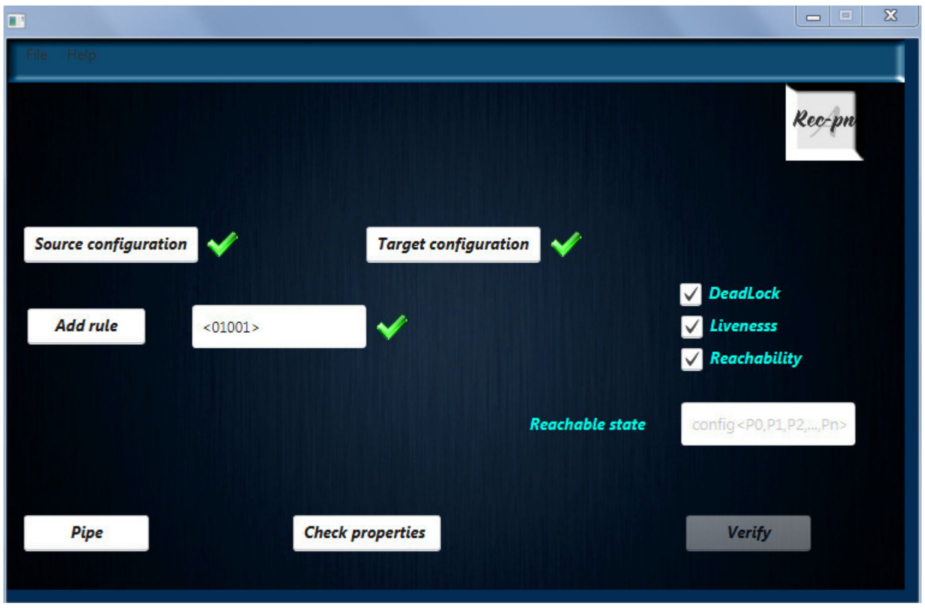
3. Add rule button: it will add the reconfiguration rule to the data structure under controlled conditions, which are: source and target configuration must be imported (it can be viewed by the confirmation components), rule condition must be set in the text field once all this done right a confirmation icon will will be shown.

4. Rule text field: where user writes the enabling marking for the reconfiguration rule(it should be set under this syntax starts with $' <'$, and ends with $' >'$ and the marking is in between). The verify button will be enabled when a rule is added.

5. In order to verify properties user should choose the properties that he wants to verify by checking them in the properties check box.

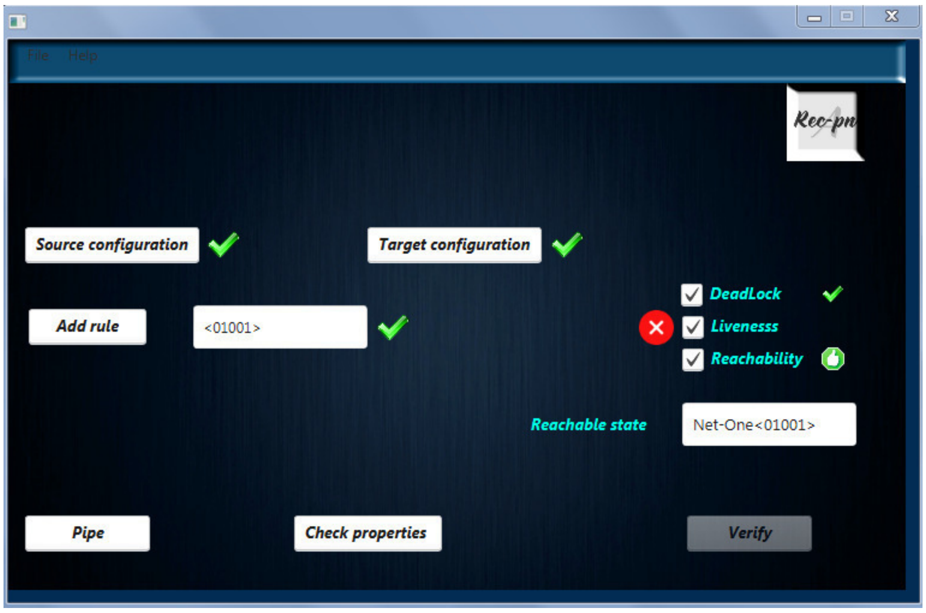


6. Verify button: it will generate a data structure model based on the inputs and then, it will create a file that contains all the markings. When this process ends, the check properties button will be enabled and the state text field will also be enabled in order to check the reachability of a set state. Finally the verify button will be set disabled.



7. State text field: in order to check whether a state is reachable, user should enter the state as it is indicated (configuration's name $< marking >$).

8. Check properties button: it will show the result of the verified properties by showing new components to indicate their presence.



9. Pipe button: it makes a call to an external program called PIPE.

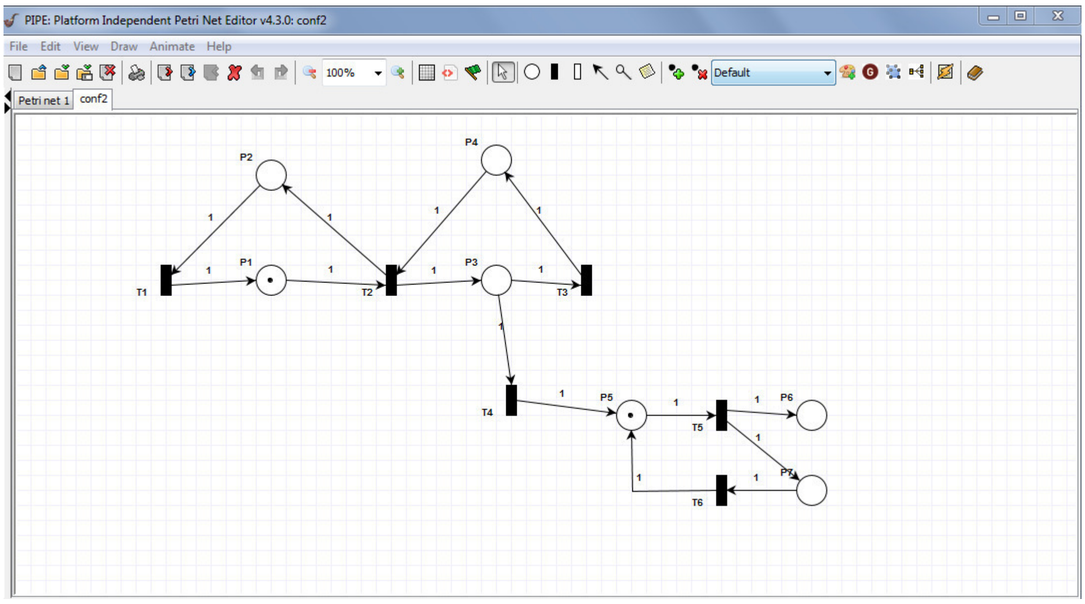

PIPE: Platform Independent Petri Net Editor v4.3.0: conf2
File Edit View Draw Animate Help
100%
Default
Petri net 1
conf2
P1
P2
P3
P4
P5
P6
P7
T1
T2
T3
T4
T5
T6

Figure 4.8: Pipe.

4.4 Conclusion

We have seen in this chapter two parts, the first part is tools and languages of the development, the tools and the programming language that we chose to start this work, and the second part is the implementation phase, the step of producing a tool for modeling and verifying reconfigurable Petri nets.

4.5 General conclusion

During this project, we explored the field of formal verification of reconfigurable systems and more specifically reconfigurable Petri nets. These provide powerful mechanisms for modeling reconfigurable systems. Nevertheless, the complexity of the verification is increased considerably (the modeling power decreases the decision power). Reconfigurable Petri nets provides a framework for modeling the structural changes in reconfigurable systems, and for checking the properties of reconfigurable Petri nets we have came with a solution that consists of developing a tool for modeling and verifying RPNs. Our study was carried out mainly in two stages:

- A theoretical study on Petri nets, reconfigurable system and reconfigurable Petri nets.
- The implementation of a modeling and verification tool for the reconfigurable Petri nets that provides the solution for the structural changes in reconfigurable systems, which has been the main goal of our work. We have been able to achieve the objectives outlined in this project, but there is still some work to do to enrich it, for example:
  - Analyze the unbounded RPN.
  - Define new techniques for structural analysis.
  - Study the reconfiguration in other class of Petri nets.

Bibliography

[1] S. Tigane, L. Kahloul, and S. Bourekkache, "Reconfigurable stochastic petri nets for reconfigurable manufacturing systems," in *Service Orientation in Holonic and Multi-Agent Manufacturing* (T. Borangiu, D. Trentesaux, A. Thomas, P. Leitão, and J. B. Oliveira, eds.), (Cham), pp. 383–391, Springer International Publishing, 2017.

[2] A. Halder, "A study of petri nets modeling, analysis and simulation," Aug 2006.

[3] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr 1989.

[4] J. Campos, G. Chiola, and M. Silva, "Ergodicity and throughput bounds of petri nets with unique consistent firing count vector," *IEEE Transactions on Software Engineering*, vol. 17, pp. 117–125, Feb 1991.

[5] J. Padberg, K. Hoffmann, H. Ehrig, T. Modica, E. Biermann, and C. Ermel, "Maintaining consistency in layered architectures of mobile ad-hoc networks," in *Fundamental Approaches to Software Engineering* (M. B. Dwyer and A. Lopes, eds.), (Berlin, Heidelberg), pp. 383–397, Springer Berlin Heidelberg, 2007.

[6] K. Gabriel and H. Ehrig, *Modelling of Communication Platforms Using Algebraic High-Level Nets and Their Processes*, pp. 10–25. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[7] M. Llorens and J. Oliver, "Structural and dynamic changes in concurrent systems: reconfigurable petri nets," *IEEE Transactions on Computers*, vol. 53, pp. 1147–1158, Sept 2004.

[8] B. Tarnauca, D. Puiu, V. Comnac, and C. Suciu, "Modelling a flexible manufacturing system using reconfigurable finite capacity petri nets," in *2012 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, pp. 1079–1084, May 2012.

[9] L. Kahloul, S. Bourekkache, and K. Djouani, "Designing reconfigurable manufacturing systems using reconfigurable object petri nets," *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 8, pp. 889–906, 2016.

[10] J. Padberg and L. Kahloul, *Overview of Reconfigurable Petri Nets*, pp. 201–222. Cham: Springer International Publishing, 2018.

[11] D. Welch and J. Purtilo, “Domain-driven reconfiguration in collaborative virtual environments,” in *Proceedings of IEEE 6th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 167–172, Jun 1997.

[12] G. Li, P. Mitrouchev, Y. Wang, D. Brissaud, and L. Lu, “Evaluation of the logistic model of the reconfigurable manufacturing system based on generalised stochastic petri nets,” *International Journal of Production Research*, vol. 50, no. 22, pp. 6249–6258, 2012.

[13] J. C. Lyke, C. G. Christodoulou, G. A. Vera, and A. H. Edwards, “An introduction to reconfigurable systems,” *Proceedings of the IEEE*, vol. 103, pp. 291–317, March 2015.

[14] C. Ellis, K. Keddara, and G. Rozenberg, “Dynamic change within workflow systems,” in *Proceedings of Conference on Organizational Computing Systems*, COCS ’95, (New York, NY, USA), pp. 10–21, ACM, 1995.