N° d'ordre :
N° de série :

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE ECHAHID HAMMA LAKHDAR EL OUED

FACULTÉ DES SCIENCES EXACTES

Mémoire de Fin D'étude
Présenté pour l'obtention du Diplôme de

MASTER ACADEMIQUE

Domaine : Mathématique et Informatique

Filière : Informatique

Spécialité : Systèmes Distribués et Intelligence Artificielle

THEME

Réalisation d'un outil de génération automatique de spécification Maude (nouvelle sémantique) des réseaux de Petri.

Présenté par : **Madani Karima & Medjouri Sara**

Soutenue le 28-09- 2020

Devant le jury composé de :

Mr. Abbas Messaoud	Président
Mr. Meftah M.C.Eddine	Examinateur
Mr. Boucherit Ammar	Rapporteur

Année Universitaire : 2019-2020

Table des matières

Chapitre 3 : Conception et aspect d'implémentation

Remerciements

Avant de commencer la présentation de ce mémoire, Nous tenons à remercier «ALLAH» du fond du cœur de nous avoir donné la force, la volonté et le courage de réaliser ce travail. Et toute personne qui a contribué de près ou de loin à la réalisation de ce mémoire. Ensuite, bien sûr, nous tiens à remercient mon encadreur Monsieur ***Boucherit Ammar*** *pour son soutien et pour son aide.*

Nous remerciements à oncle ***Abdallah Chetehouna*** *pour tout ce qu'il a à offrir. Ils nous a guident, critiquent, fait des suggestions. Pour eux encouragement facilité la tâche. Nous leur remercie vivement pour tout.*

Nous remercions chaleureusement les membres du jury qui nous ont fait Honneur de participer à notre jury et d'accepter de juger ce travail.

En fin, nous voulons encore remercier tous ceux qui nous ont consolidé et nous ont encouragé entre autre les parents et les amis.

.

Dédicace

Je dédie ce mémoire :

A mon père, mon premier encadrant, depuis ma naissance :

***Hamza Madani** (rahimahou Allah) ;*

*A ma très chère mères : qu'elle trouve ici l'hommage de ma gratitude qui, si grande qu'elle puisse être, ne sera la hauteur de ses sacrifices et ses prière pour moi : **Saida Naceur** ;*

A mes sœurs et mes frères, chacun en son nom ;

*A ma chère amie **Taymia Benseghier** ;*

*A mon binôme **Sara***

A qui je souhaite beaucoup de réussite et de bonheur ;

A tous mes amies et amis qui me sont chers, a tous ceux que j'aime et qui m'aiment : qu'ils trouvent ici l'expression de mes sentiments les plus dévoués et mes vœux les plus sincères ;

Que dieu le tout puissant vous préserve tous et vous procure sagesse et bonheur

M.Karima

Dédicaces

***A** l'homme de ma vie, mon exemple éternel, mon soutien moral et source de joie et de bonheur, celui qui s'est toujours sacrifié Pour moi voir réussir, Mon cher père (**Khaled Medjouri**).*

***A** la lumière de mes jours, la source de mes efforts, la flamme de mon cœur, ma vie et mon bonheur ; maman que j'adore (**Aicha Medjouri**).*

***A**ux personnes dont j'ai bien aimé la présence dans ce jour, à tous mes sœurs (**Nadjet**, **Manel**, **Chafika**, **Kaouthar**), je dédie ce travail dont le grand plaisir leurs revient en premier lieu pour leurs conseils, aides, et encouragements.*

***A**ux personnes qui m'ont toujours aidé et encouragé, qui étaient toujours à mes côtés, et qui m'ont accompagnaient durant mon chemin d'études supérieures, mes aimables amis, collègues d'étude, et mon adorable sœur de cœur, **Safa**.*

***A** mon binôme **KARIMA** et toute la famille.*

***E**t à tous ceux qui ont contribué de près ou de loin pour que ce projet soit possible, je vous dis merci.*

M.Sara

Résumé

Les systèmes informatiques sont devenus de plus en plus une partie intégrante de notre vie et exécutant des fonctions de plus haut niveau et critiques. Dans le but d'assurer leur bon fonctionnement, les réseaux de Petri représentent un outil très reconnu et utilisé pour faire face à la complexité croissante des systèmes dynamiques. Cependant, ils nécessitent d'être validés par des outils formels plus efficaces afin de prouver les propriétés des systèmes étudiés.

Dans ce contexte, le système Maude est un langage de programmation basé sur la logique de réécriture et offre une large gamme des outils formels pour l'analyse des réseaux de Petri.

Notre travail s'inscrit dans ce cadre général et on va proposer une approche pour la génération automatique des spécifications Maude pour les réseaux de Petri afin de bénéficier des outils formelle offerts par Maude dans l'analyse des modèles de réseaux de Petri.

Mots clés : Logique de réécriture, Réseaux de Petri.

الملخص

أصبحت أنظمة الكمبيوتر أكثر جزء لا يتجزأ من حياتنا وتؤدي وظائف ذات مستوى أعلى ووظائف مهمة. من أجل ضمان عملها بشكل صحيح، تعتبر شبكات بتري أداة معترف بها على نطاق واسع تستخدم للتعامل مع التعقيد المتزايد للأنظمة الديناميكية. ومع ذلك، يجب التحقق من صحتها من خلال أدوات رسمية أكثر كفاءة لإثبات خصائص الأنظمة المدروسة.

في هذا السياق، فإن نظام Maude هو لغة برمجة تعتمد على منطق إعادة الكتابة ويقدم مجموعة واسعة من الأدوات الرسمية لتحليل شبكات بتري.

يندرج عملنا ضمن هذا الإطار العام وسنقترح نهجًا للتوليد التلقائي لمواصفات Maude لشبكات بتري من أجل الاستفادة من الأدوات الرسمية التي تقدمها Maude في تحليل نماذج شبكات بتري.

الكلمات المفتاحية: منطق إعادة الكتابة، شبكات بتري.

Abstract

Computer systems have become more and more an integral part of our life and performing higher level and critical functions. In order to ensure their proper functioning, Petri nets are

a widely recognized tool used to deal with the increasing complexity of dynamic systems. However, they need to be validated by more efficient formal tools in order to prove the properties of the studied systems.

In this context, the Maude system is a programming language based on rewriting logic and offers a wide range of formal tools for the analysis of Petri nets.

Our work falls within this general framework and we will propose an approach for the automatic generation of Maude specifications for Petri nets in order to benefit from the formal tools offered by Maude for the analysis of Petri net models.

Key words: Rewriting logic, Petri nets

Table des Figures

Introduction générale

1. Contexte général et Problématique

Les systèmes informatiques sont devenus indispensables dans le domaine industriel pour une production efficace. Ils s'occupent du contrôle des machines, de la communication entre elles ainsi que le contrôle qualité. Implicitement, ceux-ci sont de plus en plus complexes et contiennent plusieurs composants hétérogènes et distribués interagissant et / ou exécutant des fonctions de plus haut niveau. En plus, bien que ces systèmes soient également devenus une partie intégrante de notre vie et aient rendu nos activités quotidiennes très faciles et confortables, ils sont aussi responsables d'un nombre toujours croissant d'erreurs de gravité variable. Par conséquent, le développement de ces systèmes, avec un tel niveau de criticité pouvant impliquer un risque de vie humaine ou un risque financier important, a provoqué un appel l'utilisation des outils puissants de modélisation et nécessite une approche de conception rigoureuse qui permet de répondre aux exigences de fiabilité en premier lieu sans négliger les autres contraintes de développement.

Depuis leur première apparition en 1962, les réseaux de Petri (RdP) ont été considérés comme un outil de modélisation et d'analyse très efficace, largement utilisé dans l'industrie ainsi que dans les laboratoires de recherche grâce au socle théorique (mathématique) sous-jacent au modèle graphique. Cependant, les réseaux de Petri souffrent du manquent d'une sémantique formelle pour permettre une vérification automatique des modèles produits. Par conséquent, il devient nécessaire de chercher des approches pour pouvoir vérifier formellement les propriétés de sûreté et de vivacité des systèmes complexes modélisés.

Dans ce contexte, la logique de réécriture est une logique très expressive qui convient particulièrement à la formalisation de systèmes concurrents, complexes et temps réel[1]. De plus, c'est un cadre unificateur pour une large gamme de réseaux de Petri et de nombreux autres modèles de concurrence[2]. Néanmoins, les spécifications basées sur la logique de réécriture des modèles des réseaux de Petri sont souvent volumineuses et / ou plus difficiles à préparer, raffiner ou à corriger.

Par conséquent, il est nécessaire d'automatiser une telle opération et de gagner du temps dans le processus préparation des spécifications. Autrement dit, la recherche d'une solution qui nous permet de générer rapidement et automatiquement les spécifications formelles des réseaux de Petri sera très avantageuse. En plus, la sémantique existante des réseaux de Petri soufre de

quelques limites. Par conséquent, nous allons générer la spécification Maude pour les RdPs suivant sa sémantique améliorée[3].

Notre travail s'inscrit dans ce contexte où nous voulons exploiter la richesse de la logique de réécriture et permettre aux développeurs de bénéficier plus facilement des outils formels offerts par son langage Maude comme son model-checker.

2. Objectifs et Contributions

L'objectif principal du présent travail est de concevoir et de réaliser un outil de génération automatique de spécification Maude pour les réseaux de Petri. En effet, notre premier point de vue était d'implémenter une des trois approches suivantes :

1. Une approche directe basée sur la représentation mathématique (matrices d'incidences) des réseaux de Petri qui sont saisies par l'utilisateur.
2. Une deuxième approche indirecte basée sur la représentation PNML - obtenue d'un des outils des réseaux de Petri - à partir de laquelle on va extraire les matrices d'incidences.
3. On développe notre propre éditeur de réseaux de Petri et on réalise la transformation avec l'une des deux approches précédentes.

Pratiquent, nous avons eu la chance merveilleuse d'avoir suffisamment de temps, et nous avons réalisé toutes les approches proposées pour les deux sémantiques existante et améliorée. Ce qui va élargir – selon notre point de vue – le spectre d'utilisation du système Maude et de faciliter la validation des modèles de réseaux de Petri pour un grand nombre de chercheurs et développeurs. En plus, on a publié un article sur la deuxième approche proposée[4].

3. Organisation du mémoire

Ce mémoire se compose de deux parties où la première comporte deux chapitres et la deuxième comporte un seul chapitre. La première partie présente les fondements scientifique, i.e. les concepts de base concernant les réseaux de Petri et de la logique de réécriture. La deuxième partie présente la conception et l'implémentation de l'outil de génération automatique des spécifications Maude à partir des réseaux de Petri.

- Le premier chapitre, présente brièvement les notions élémentaires des réseaux de Petri nécessaire pour le reste de notre travail.
- Le deuxième chapitre a été consacré à la présentation des principaux concepts de la logique de réécriture, et de son langage Maude.

➲ Dans le troisième chapitre, nous présentons la conception et quelques détails d'implémentation de l'outil développé suivi d'une conclusion de notre point de vue sur le futur de ce travail.

Chapitre 1

Réseaux de Petri

1. Introduction

Les réseaux de Petri est un outil puissant permettant l'étude de systèmes dynamiques et discrets. En effet, il s'agit d'une représentation mathématique facilitant la modélisation d'un système. En plus, Leur analyse peut révéler des caractéristiques importantes concernant la structure et le comportement dynamique du système. Ensuite, les résultats de cette analyse peuvent être utilisés pour l'évaluation de la performance ainsi que la qualité de service du système comme ils permettent la modification et/ou l'amélioration de son architecture.

Ce chapitre a pour but de présenter quelques définitions et concepts concernant le formalisme de réseau de Petri liés à notre travail, leur représentation et leur analyse.

2. Concept de base

Définition :

Un **réseau de Petri** (RdP) est un quadruplet $R = < P, T, Pre, Post >$

- P représente l'ensemble fini de **places**, $\boldsymbol{P_i} = \{P_1, P_2, \dots\}$.
- T représente l'ensemble fini de **transitions,** $\boldsymbol{T_j} = \{T_1, T_2, \dots\}$.
- $Pre : P \times T \rightarrow N$ « places d'entrée à une transition ». tel que $Pre(p,t)$ = nombre de jeton nécessaire dans la place p pour le franchir de la transition t
- $Post : P \times T \rightarrow N$ « place de sortie à une transition ». tel que $Post(p,t)$ = nombre de jeton produits dans la place p lors du franchissement de la transition t.

Il est noté que les places et transitions sont reliées par des arcs orientés. C'est pourquoi on dit qu'un RdP est un graphe biparti orienté. Plus précisément, on attribue à chaque arc un poids (nombre entier) qui représente le nombre de jetons (consommés ou produits). Par défaut ce nombre est égal à 1.

3. Représentations d'un réseau de Petri

3.1. Représentation graphique

Un réseau de Petri est un graphe biparti – est un quadruplet $R = < P, T, Pre, Post >$ – dont on particularise les deux familles de sommets, des places et des transitions reliées alternativement par des arcs : [5]

- Les *places*, sont représentées par des cercles ou des ronds.
- Les *transitions* sont représentées par des traits ou des rectangles.
- Les *arcs* qui ne relient jamais deux sommets de la même famille.
- Les *jetons* (marquage), sont représentés par des points noirs.

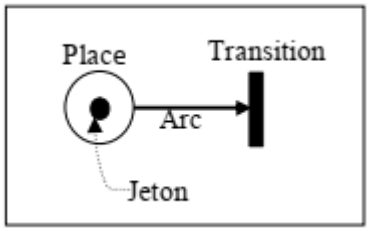
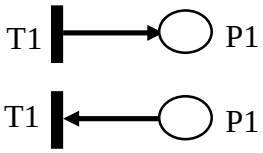


Figure 1: Représentation graphique d'un RdP.

Remarque :

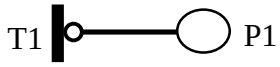
1. Une transition peut avoir deux cas particuliers :

- **Transition source :** pas de place en entrée de la transition.
- **Transition puits :** pas de place en sortie de la transition.



2. Un arc peut avoir un cas particulier appelé : Arc inhibiteur, qui est un arc utilisé seulement pour le test et ne consomme pas les jetons lors du franchissement d'une transition.

Sa représentation graphique est donnée comme suit.



3. Lors de la modélisation d'un système, on distingue les cas suivants :

Condition : modélisée à l'aide d'une place, est un prédicat logique d'un état du système, Elle est soit vraie, soit fausse. La satisfaction d'une condition est modélisée à l'aide d'un jeton.

Condition vraie  Condition fausse 

Évènement : modélisé à l'aide d'une transition, les événements sont des actions se déroulant dans le système, Le déclenchement d'un événement dépend de l'état du système.

Un état du système peut être décrit comme un ensemble de conditions[6].

3.2. Représentation matricielle

Il est possible de représenter un réseau de Petri par des matrices.

$Pre(P_i, T_j) = \begin{cases} k, Si\ l'arc\ W(P_i, T_j)\ existe \\ 0, sinon \end{cases}$ - La matrice Pre (P_i ,T_j) : comporte les poids k des arcs reliant une place d'entrée à une transition.

$Post(T_j, P_i) = \begin{cases} k\ , Si\ l'arc\ W(T_j, P_i)\ existe \\ 0\ , sinon \end{cases}$ - La matrice Post (T_j,P_i) : comporte les poids k des arcs reliant une transition une place de sortie à une transition.

Remarque :

- La matrice Pré (P,T) (resp, Post(T,P)) est appelée matrice d'incidence avant (resp, arrière).
- La matrice C = Post - Pré est appelée matrice d'incidence du réseau[7].
- la matrice Pré (P,T) peut représenter les arcs inhibiteurs avec des valeurs négatives.

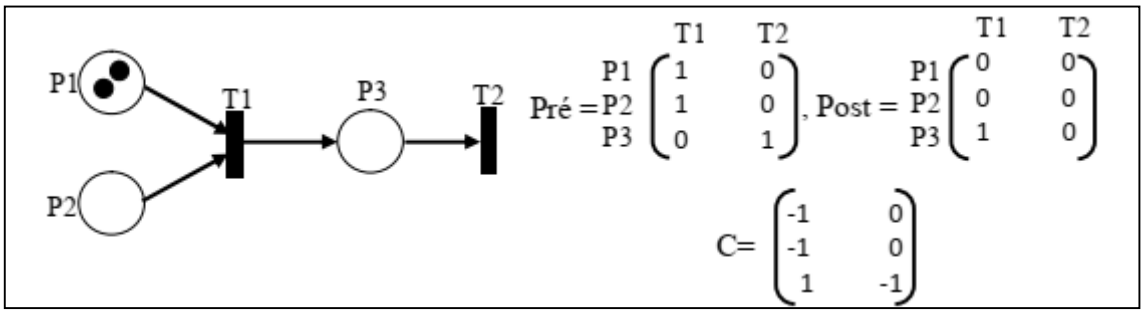


Figure 2 : Représentation matricielle d'un RdP

3.3. Représentation PNML

Le langage PNML est un format de représentation et d'échange des modèles de réseaux de Petri entre les outils dédiés, basé sur XML. Une description PNML est généralement composée d'un ensemble de nœuds comme suit :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pnml>
  <net id="Net-One" type="P/T net">
    <place id="P0">
      <name> <value>P0</value> </name>
      <initialMarking> <value>Default,0</value> </initialMarking>
    </place>
        ......
    <transition id="T0">
      <name> <value>T0</value> </name>
      <timed> <value> false </value> </timed>
    </transition>
        ......
    <arc id="P0 to T1" source="P0" target="T1">
      <inscription> <value>Default,1</value> </inscription>
      <type value="normal"/>
    </arc>
        ......
  </net>
</pnml>
```

Figure 3 : Exemple de fichier PNML

En effet, la description PNML contient la définition structurelle de base d'un réseau de Petri sous la forme d'un graphe orienté étiqueté. PNML offre un format d'échange universel de réseau de Petri pour permettre aux outils de réseaux de Petri d'échanger des modèles.

Pratiquement, il existe de nombreux outils tels que PN Matlab Toolbox[8], Yasper[9], WoPeD[10], P3[11], PIPE[12] capables d'exporter le modèle de réseau Petri au format PNML.

4. Évolution d'un RdP

4.1. Marquage

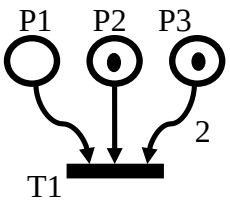
$M(P_1) = 2, M(P_2) = 0, M(P_3) = 0.$ $Donc\ M_0 \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$. Dans la théorie des réseaux de Petri, l'état d'un réseau est souvent appelé marquage. Le marquage M est un vecteur colonne -de dimension égal au nombre de places du RdP- qui sert à définir l'état du système à un instant donné. Plus précisément, les éléments du vecteur correspondent au nombre de jetons (positif ou nul) contenus dans la place P_i. Ce nombre peut être négatif pour indiquer un arc inhibiteur. Le marquage dit initial décrit l'état initial du système modélisé (M_0)[13]. Le marquage du RdP de la Figure 2 est donné comme suit :

En plus, Le fonctionnement d'un RdP correspond à l'évolution de son marquage au cours du temps. Cette évolution est régie par un ensemble des règles de franchissement (ou de tir).

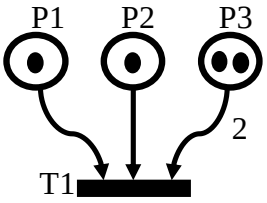
4.2. Franchissement d'une Transition

Une transition est dite « franchissable » si elle valide, i.e. si toutes ses places d'entrée contiennent un nombre de jetons supérieur ou égal à celui indiqué sur les arcs correspondants.

Exemple



T1 n'est pas franchissable car P1 et P3 ne contiennent pas suffisamment de jetons !
(n'est pas valide)



T1 est valide et franchissable

Figure 4 : Exemple de validation / franchissement

Lorsqu'on franchisse une transition, des jetons sont retirés de ses places d'entrée et des jetons sont déposés dans ses places de sortie. Le franchissement d'une transition permet donc d'atteindre un nouveau marquage M_1 à partir de M_0 : $M_0 \rightarrow M_1$.[14]

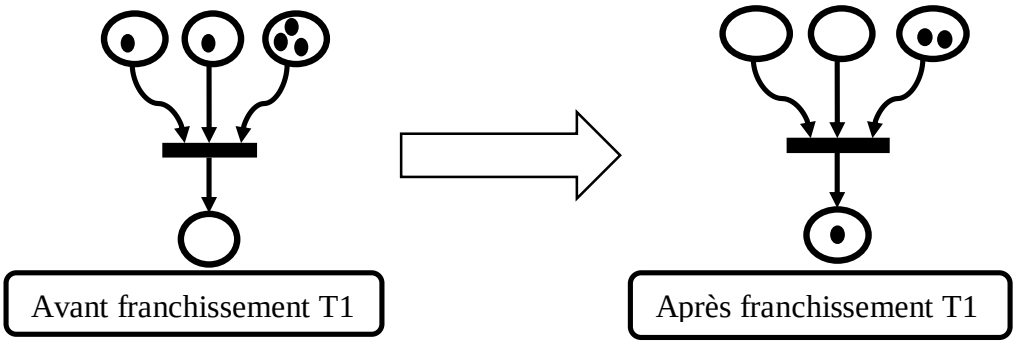


Figure 5 : Exemple de franchissement

Remarque :

- La transition source est toujours franchissable.
- Lorsqu'une transition est validée, cela n'implique pas qu'elle sera immédiatement franchie ; cela ne représente qu'une possibilité de franchissement ou d'évolution du RdP.
- Le réseau ne peut évoluer que par franchissement d'une seule transition à la fois, une transition choisie parmi toutes celles qui sont validées à cet instant.[15]

4.3. Séquence de franchissement

Le franchissement, à partir de marquage initial M_0, de T_1 puis T_2 (voir la Figure 7) conduit au marquage M_2. On appellera T_1T_2 séquence de franchissements et on notera :

$M_0, \xrightarrow{T1\ T2} M_2$, ou encore $M_0, \xrightarrow{S} M_2$ Avec $S = T_1T_2$.

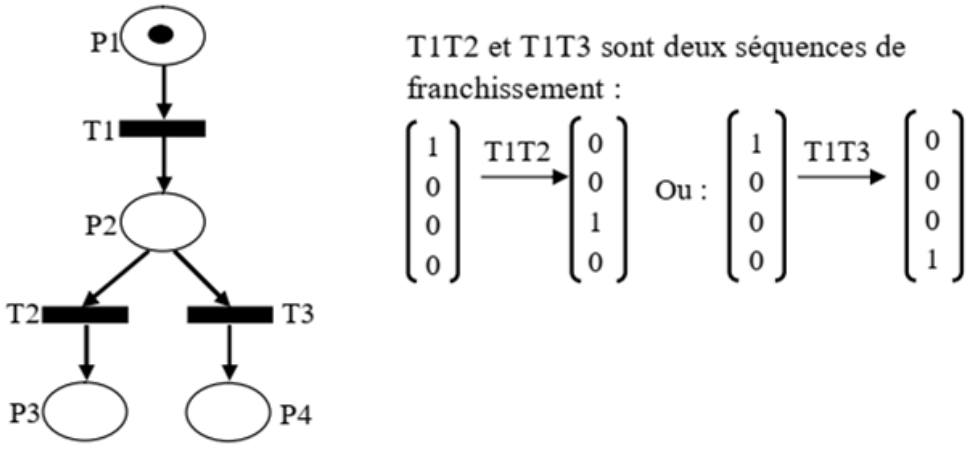


Figure 6 : Exemple de séquence de franchissements

Exemple : on a l'évolution de RdP suivant :

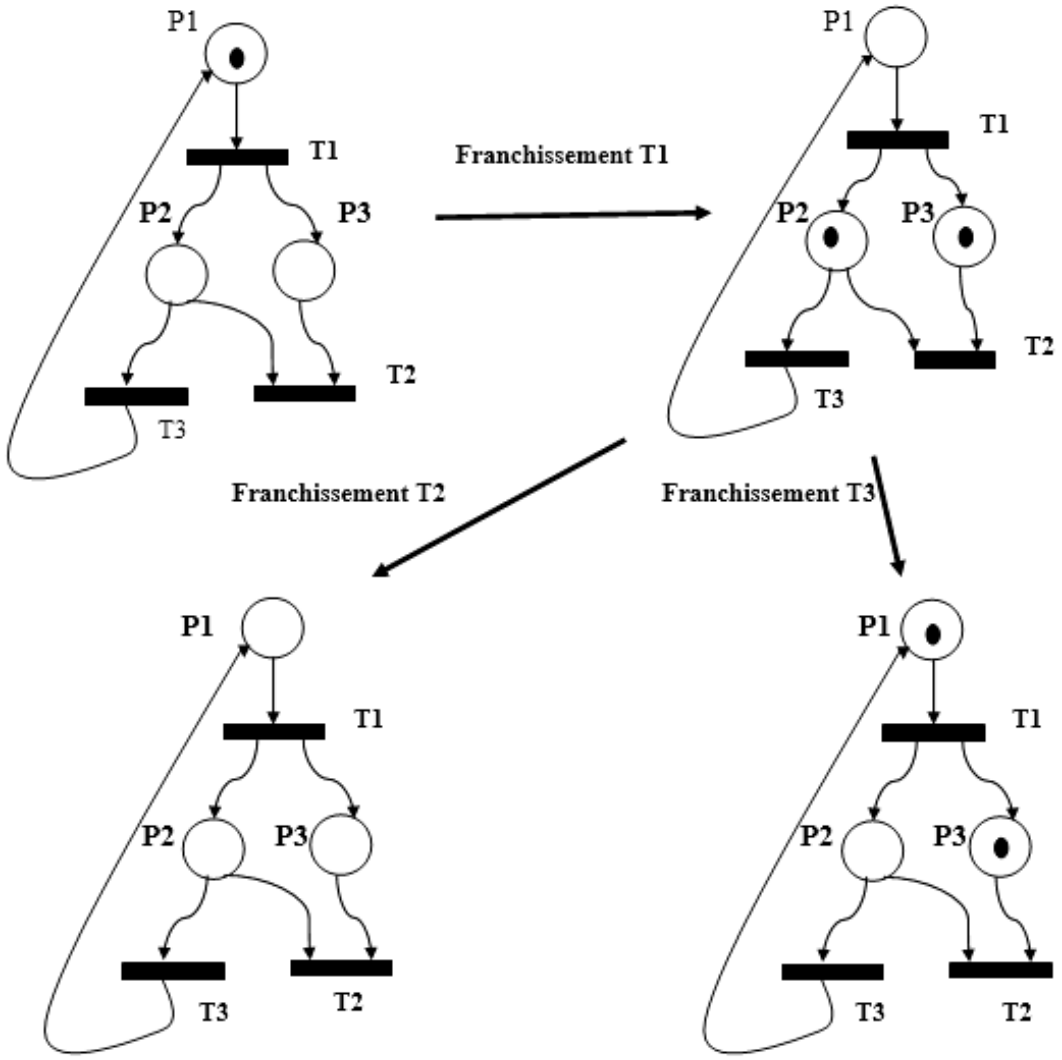


Figure 7 : Exemple de l'évolution de RdP

5. Domaine d'applications des réseaux de Petri

Les réseaux de Petri ont largement utilisés pour la modélisation et on les retrouve dans des différents domaines, dont :

- L'industrie :

La modélisation des chaînes de production (de fabrication).

- L'informatique :

La modélisation des systèmes informatiques.

La modélisation des protocoles de communication.

- La chimie :

Les réactions chimiques peuvent être modélisées par les réseaux de Petri.

6. Exemple de Modélisation par réseaux de Petri

Une piscine comporte c cabines pour se changer et p paniers pour déposer ses vêtements.

- On n'entre dans la piscine que si une cabine est libre. On attend un panier pour se changer et déposer ses vêtements. On libère la cabine et on pénètre dans le bassin.
- On ne quitte le bassin que si une cabine est libre. On se change et on restitue cabine et panier. On quitte la piscine.

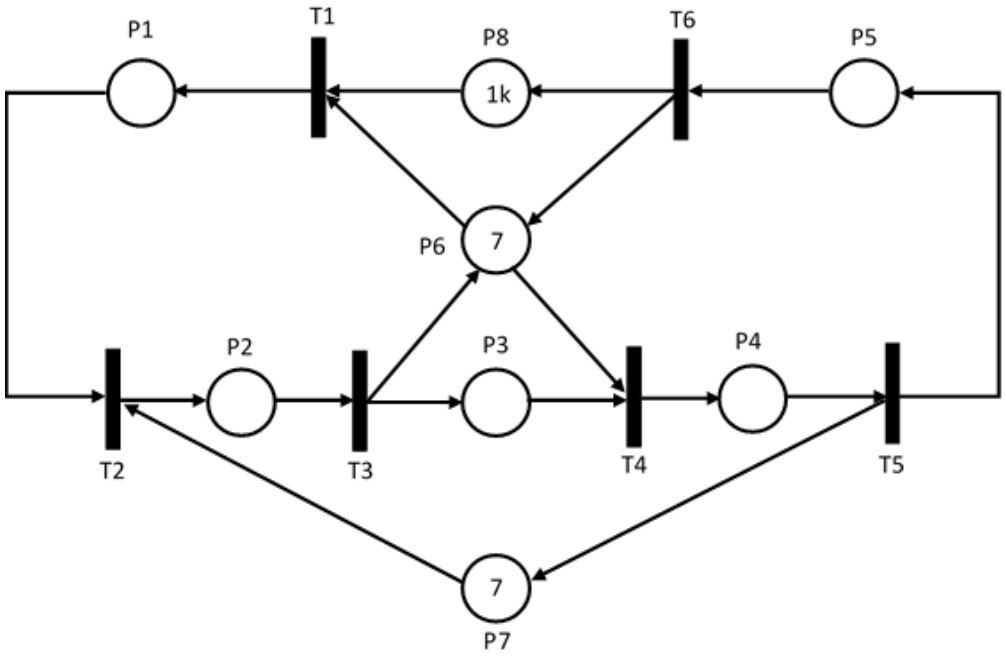


Figure 8 : Modélisation de piscine par réseaux de Petri

T_1 : Le client entre à la piscine
(Si ∃ cabine libre (P_6))
T_2: Le client se prépare et change ses vêtements
(Si ∃ panier libre (P_7))
T_3: Le client pénètre dans le bassin
(Et restitue la cabine (P_6))
T_4: Le client quitte le bassin
(Si ∃ cabine libre (P_6))
T_5: Le client s'habille (et restitue le panier (P_7))
T_6: Le client quitte la piscine

P_1: Client en attente d'un panier
P_2: Client veut changer ses vêtements
P_3: Client dans le bassin
P_4: Client s'habille
P_5: Client habillé prêt à sortir
P_6: Compteur de cabines libres
P_7: Compteur de paniers libres

7. Conclusion

Dans ce chapitre, nous nous sommes intéressés à la présentation des concepts théoriques des réseaux de Petri. L'outil mathématique, inventé par Carl Adam Petri en 1962, permettant la modélisation et l'analyse des systèmes à variables discrètes.

Chapitre 2

Logique de réécriture

1. Introduction

Les techniques de réécriture ont été développées depuis les années 1970 et appliquées en particulier au prototypage des spécifications formelles algébriques et à la démonstration des propriétés liées à la vérification des programmes. A l'origine, le but était de trouver un système de réécriture canonique qui permet de prouver la validité d'u théorème équationnel, en réécrivant chaque membre de l'égalité en un même terme[16].

Autrement dit, la réécriture est un paradigme général d'expression de calcul dans différentes logiques computationnelles. Les calculs prennent la forme de règles dans une syntaxe donnée. Dans une logique équationnelle, celles-ci résultent de l'interprétation d'équations entre termes. Par contre, dans la logique de réécriture, réécrire un terme consiste à le remplacer par un terme équivalent, en vertu des lois de l'algèbre des termes[17]. En particulier, la réécriture pour cette logique permet de calculer une relation de réécrivabilité entre des termes algébriques.

La logique de réécriture est donc une logique de changement où chaque règle de réécriture est une forme générale d'une action atomique pouvant survenir en concurrence avec d'autres actions dans un système concurrent. Cette logique est une conséquence des travaux de José Meseguer[18] sur des logiques générales pour décrire les systèmes concurrents. La logique de réécriture est un cadre logique et sémantique unificateur dans lequel d'autres logiques, modèles de concurrences et langages peuvent être représentées (CCS, LOTOS, Pi-calcul, réseaux de Petri, etc.). Dans la logique de réécriture, une réécriture d'un terme consiste à le remplacer par un terme équivalent, conformément aux lois d'algèbre de termes.

Les spécifications des systèmes dans cette logique de réécriture sont exécutables. Le langage de la logique de réécriture Maude est l'un des langages les plus puissants pour la spécification formelle, la programmation et la vérification des systèmes concurrents.

Dans ce chapitre, nous introduirons la logique de réécriture ainsi que existante pour les RdPs. Ensuite, nous abordons l'environnement Maude, ses spécificités et son fonctionnement.

2. La logique de réécriture

Dans le but d'éclaircir les différentes notions concernant la logique de réécriture, nous allons donner une définition des concepts nécessaires à la compréhension de notre travail.

2.1. Signature

Une signature $\sum$ est une paire (S, Op) définie par un ensemble de sortes S et un ensemble d'opérations Op o'u :

- S est un ensemble fini : $S_1, S_2, \dots, S_n$
- Op est un symbole d'opérations qui peuvent avoir 0 à n arité.

  Chaque opération non nulle à la forme $f : S_1, S_2, \dots, S_n \rightarrow S_{n+1}$.

Pour toute opération f

- $(S_1, S_2, \dots, S_n, S_{n+1})$ est appelé le profil de l'opération f
- $S_1, S_2, \dots, S_n$ domaine de f
- S_{n+1} co-domaine de f
- n est l'arité de$f$. Les constantes sont des opérations d'arité nulle [19]

Exemple

On peut définir la signature des booléens comme suit :

$$\left.\begin{array}{l} \textbf{\textit{Sorts}}\ bool \end{array}\right\} \boldsymbol{S}$$

$$\left.\begin{array}{l} \textbf{\textit{Ops}}\ true : \rightarrow bool \\ false : \rightarrow bool \\ not(\_) : bool \rightarrow bool \\ \_and\_ : bool\ bool \rightarrow bool \\ \_or\_ : bool\ bool \rightarrow bool \end{array}\right\} \boldsymbol{Op} \Big\} \boldsymbol{\Sigma}$$

2.2. Le terme

Soit $\sum = (S, Op)$ une signature et V est un ensemble infini dénombrable de variables, L'ensemble des termes T est le plus petit ensemble contenant les variables et les constantes (T =C U V) stable par l'application des symboles de fonctions de Op à des termes. Autrement dit, un terme est un mot qu'on peut obtenir en appliquant récursivement un nombre fini de fois les règles ci-dessous :

- Tout symbole de constante est un terme.
- Toute variable est un terme.
- Si f est un symbole de fonction d'arité n et si $t_1, \dots, t_n$ sont des termes, alors $f(t_1, \dots, t_n)$ est un terme**.**

Remarque

On appelle terme clos un terme qui ne contient pas de variable.

2.3. ∑-équation

Soit $\sum = (S, Op)$ une signature et V un ensemble de variable, une ∑-équation est une paire (t, t') avec t et t' deux éléments de $T_\Sigma(V)$. Sachant que la notation $T_\Sigma(V)$. est utilisée pour désigner l'ensemble des termes T formés à partir de l'ensemble des variables V vérifiant la signature ∑. Une ∑-équation (t, t') est notée $t = t'$ **.**

2.4. Spécification algébrique

Une signature dans la logique de réécriture est une théorie équationnelle (Σ, E) où Σ est une signature équationnelle formée par un ensemble de sortes et d'opérateurs et E est un ensemble de Σ-équations décrivant les propriétés structurelles d'un système donné.

Exemple

Spécification algébrique des booléens

$\boldsymbol{Sort}\ \ bool$

$\boldsymbol{Opns}\ \ true :\rightarrow bool$

$false :\rightarrow bool$

$not(\_) : bool \rightarrow bool$

$\_and\_ : bool\ bool \rightarrow bool$

$\_or\_ : bool\ bool \rightarrow bool$

$\boldsymbol{Var}\ \ x : bool$

$\boldsymbol{Eqns}\ \ (not(true) = false)$

$(not(false) = true)$

$(true\ and\ x = x)$

$(false\ and\ x = false)$

$(true\ or\ x = true)$

$(false\ or\ x = x)$

3. Théorie de réécriture

Dans la logique de réécriture, un système concurrent est décrit par une théorie de réécriture *R = (Σ, E, L, R)* où *(Σ, E)* désigne la signature (une théorie équationnelle) définissant la structure algébrique particulière des états du système (multi-ensemble, arbre binaire…) qui sont distribués selon cette même structure. La structure dynamique du système est décrite par les

règles de réécriture étiquetées R (L est un ensemble d'étiquettes de ces règles). Les règles de réécriture précisent quelles sont les transitions élémentaires et locales possibles dans l'état actuel du système concurrent.

Chaque règle (notée $[t] \rightarrow [t']$) correspond à une action pouvant survenir en concurrence avec d'autres actions. Donc, la logique de réécriture est une logique qui capture clairement le changement concurrent dans un système.[20]

3.1. Théorie de réécriture étiquetée

Une théorie de réécriture R est un 4-uplet (Σ, E, L, R) tel que :

1. Σ est un ensemble de symboles de fonctions, et de sortes.
2. E un ensemble de Σ-équations (l'ensemble des équations entre les Σ-termes).
3. L est un ensemble d'étiquettes.
4. R est un ensemble de règles de réécriture, défini ainsi $R \subseteq L \times \left(T_{\Sigma,E}(X)\right)^2$; chaque règle est un couple d'éléments, le premier est une étiquette, le second est une paire de classes d'équivalence de termes $T_{\Sigma,E}(X)$ sur la signature (Σ, E), modulo les équations E, avec $X = \{x_1, \dots, x_n, \dots\}$ un ensemble infini et dénombrable de variables.

3.2. Les systèmes de réécriture conditionnels

En ajoutant des conditions sur l'application des règles de réécriture, on étend naturellement les systèmes de réécriture à des systèmes de réécriture conditionnels.

Un système de réécriture conditionnel naturel à des règles de réécriture de la forme R $([t], [t'], C_1, \dots, C_k)$, la notation suivante est utilisé :

$$\boldsymbol{R} : [t] \rightarrow [t'] \; if \; C_1 \wedge \dots \wedge C_k.$$

Où une règle ***R*** exprime que la classe d'équivalence contenant le terme ***t*** peut se réécrire en la classe d'équivalence contenant le terme ***t'*** si la condition de la règle $C_1 \wedge \dots \wedge C_k$. Est vérifiée. Cette dernière est appelée condition de la règle et peut être abrégée par la lettre ***C***, et la règle de réécriture, dans ce cas, est dite conditionnelle. La partie conditionnelle d'une règle peut être vide, dans ce cas les règles sont appelées règles de réécriture inconditionnelles et sont notés par : $\boldsymbol{R} : [t] \rightarrow [t']$

Une règle de réécriture peut être paramétrée par un ensemble de variables $\{ \boldsymbol{x_1} \dots \boldsymbol{x_n} \}$ qui apparaissent soit dans $\boldsymbol{t}, \boldsymbol{t'}$ ou ***C***, et nous écrivons :

$$R: [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)] \; \boldsymbol{if} \; C(x_1, \dots, x_n).$$

4. Règles de déduction

Dans un système concurrent, la séquence des transitions exécutées à partir d'un état initial donné, constitue ce que nous appelons le calcul qui correspond à une preuve ou à une déduction dans la logique de réécriture. Les règles formalisant l'opération de réécriture des termes sont appelées des règles de déductions. Autrement dit, pour Etant donné une théorie de réécriture $R, = (\Sigma, E, L, R)$, nous disons que la séquence $[t] \rightarrow [t']$ est prouvable dans R et on écrit $R \vdash [t] \rightarrow [t']$ si et seulement si $[t] \rightarrow [t']$ est obtenue par une application finie des règles de déduction suivantes [21] :

1) La réflexivité

Pour chaque terme $[t] \in T_{\Sigma,E}(X), \overline{[t] \rightarrow [t]}$ où $T_{\Sigma,E}(X)$ est l'ensemble des Σ termes avec variables construits sur la signature Σ et les équations E

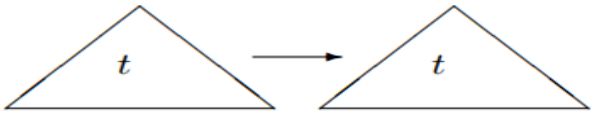


Figure 9 : Représentation graphique de la réflexivité

2) La congruence

Pour chaque fonction $f \in \sum n, n \in \mathbb{N}$.

$$\frac{[t_1] \rightarrow [t_1'] \ \dots \dots \ [t_n] \rightarrow [t_n']}{[f\ (t_1 \dots \dots\ t_n)] \rightarrow\ [f\ (t_1' \dots \dots t_n')]}$$

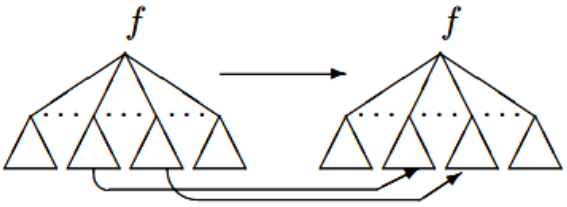


Figure 10 : Représentation graphique de la congruence

3) Le remplacement

Pour chaque règle r : $[t(x_1 \dots X_n)] \rightarrow [t'(x_1 \dots X_n)]$ dans R :

$$\frac{[w_1] \rightarrow [w_1'] \ \dots \dots \ [w_n] \rightarrow [w_n']}{[t(\bar{w}/\bar{X})] \rightarrow\ [t'(\overline{w'}/\bar{X})]}$$

Sachant que $t(\bar{w}/\bar{X})$ dénote la substitution simultanée de x_i par w_i dans t avec $\bar{x}$ représentant $x_1 \dots \dots \dots . x_n$.

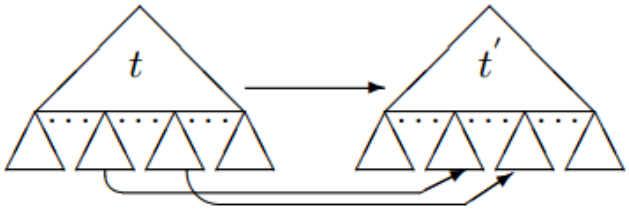


Figure 11 : Représentation graphique du remplacement

4) La transitivité

$$\frac{[t_1] \rightarrow [t_2] \quad [t_2] \rightarrow [t_3]}{[t_1] \rightarrow [t_3]}$$

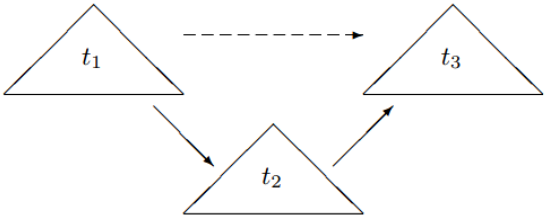


Figure 12 : Représentation graphique de la transitivité

Intuitivement, nous devrions considérer les règles d'inférence ci-dessus comme différentes manières de construire tous les calculs concurrents du système concurrent spécifié par R.

- La règle de **réflexivité** dit que pour tout état t il y a une transition inactive dans laquelle rien ne change.
- La règle de **congruence** est une forme très générale de «parallélisme latéral», de sorte que chaque opérateur f peut être vu comme un constructeur d'états parallèles, permettant à ses arguments non figés d'évoluer en parallèle.
- La règle de **remplacement** prend en charge une forme différente de parallélisme, qui pourrait être appelée «parallélisme sous les pieds», car en plus de réécrire une instance du côté gauche d'une règle sur l'instance de droite correspondante, les fragments d'état dans la substitution des variables de la règle peuvent également être réécrits, à condition que les variables impliquées ne soient pas figées.
- La règle de **transitivité** nous permet de construire des calculs simultanés plus longs en les composants séquentiellement.[22]

Exemple 1 :

Un exemple de système concurrent intégré au niveau de la logique de réécriture est montré à travers le réseau de Petri ordinaire suivant :

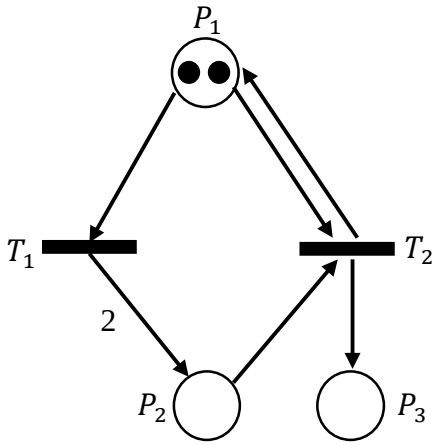


Figure 13 : Exemple de réseaux de Petri

Ce réseau de Petri peut être représenté par la théorie de réécriture suivante :

Rdp= (∑, E, L, R), tels que :

$$\sum = (\{place, marking\}, \{P_1, P_2, P_3 : \rightarrow place, null : \rightarrow place,$$
$$\_ : place \rightarrow marking, \_.\_ : marking \times marking \rightarrow marking\})$$

$$E = \{x.null = x, x.y = y.x, x.(y.z) = (x.y).z\}$$

$$L = \{T_1, T_2\}$$

$$R = \{T_1 : [P_1] \rightarrow [P_2.P_2], T_2 : [P_1.P_2] \rightarrow [P_1.P_3]\}$$

Dans cette théorie de réécriture, les places et le marquage du réseau de Petri sont spécifiés par le biais des sortes *place* et *marking*. Ainsi, l'ensemble des opérations suggérées sert à générer les places et le marquage du réseau. L'ensemble des règles de réécriture permet de décrire les transitions du réseau, la transition T_1 permet de consommer un jeton de la place P_1, et de générer deux jetons au niveau de la place P_2. La transition T_2 permet de consommer un jeton de la place P_1 et un autre de la place P_2, et de générer un jeton au niveau de la place P_1 et un autre dans la place P_3.

- Un comportement simple de ce réseau peut être habituellement vu par son évolution de l'état $[P_1.P_1]$, par exemple, vers un autre état $[P_1.P_3.P_3]$

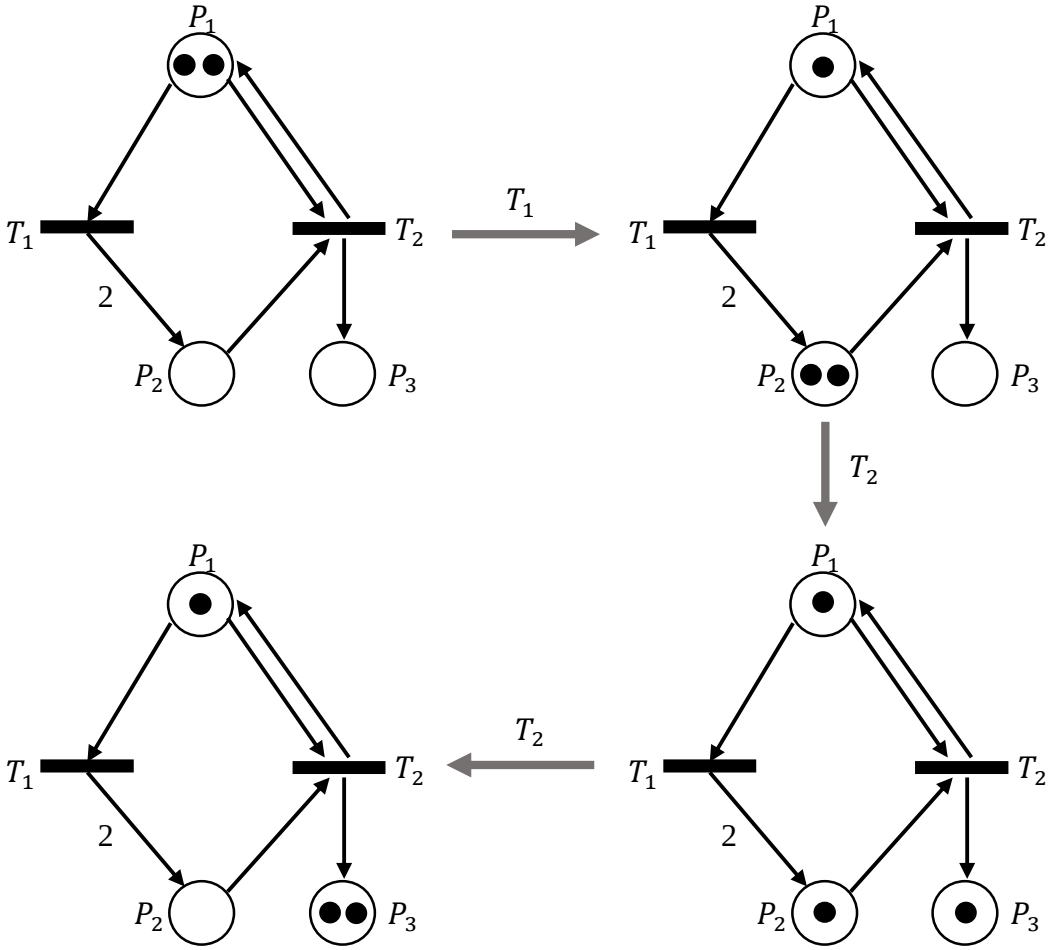


Figure 14: Un exemple d'évolution de réseau de Petri

Ce comportement est déduit par la preuve : $Rdp \vdash [P_1.P_1] \rightarrow [P_1.P_3.P_3]$ Ayant comme terme de preuve noté généralement :

$$T_1 ; T_2 ; T_2$$

Et obtenu ainsi :

$\sum = (\{place, marking\}, \{P_1, P_2, P_3 : \rightarrow place, null : \rightarrow place, \_ : place \rightarrow marking,$
$\_.\_ : marking \times marking \rightarrow marking\})$

$E = \{x.null = x, x.y = y.x, x.(y.z) = (x.y).z$

$R = \{T_1 : [P_1] \rightarrow [P_2.P_2], T_2 : [P_1.P_2] \rightarrow [P_1.P_3]\}$

$$\frac{P_1 : [P_1] \rightarrow [P_1] \; T_1 : [P_1] \rightarrow [P_2.P_2]}{P_1.T_1 : [P_1.P_1] \rightarrow [P_1.P_2.P_2]} \quad \textbf{Congruence}$$

$$\frac{T_2 : [P_1.P_2] \rightarrow [P_1.P_3] \;\; P_2 : [P_2] \rightarrow [P_2]}{T_2.P_2 : [P_1.P_2.P_2] \rightarrow [P_1.P_3.P_2]} \quad \textbf{Congruence}$$

$$\frac{P_1.T_1 : [P_1.P_1] \rightarrow [P_1.P_2.P_2] \;\; T_2.P_2 : [P_1.P_2.P_2] \rightarrow [P_1.P_3.P_2]}{(P_1.T_1);(T_2.P_2) : [P_1.P_1] \rightarrow [P_1.P_3.P_2]} \; \textbf{\textit{Transitivité}}$$

$$\frac{P_3 : [P_3] \rightarrow [P_3] \;\; T_2 : [P_1.P_2] \rightarrow [P_1.P_3]}{T_2.P_3 : [P_1.P_2.P_3] \rightarrow [P_1.P_3.P_3]} \; \textbf{\textit{Congruence}}$$

$$\frac{(P_1.T_1);(T_2.P_2) : [P_1.P_1] \rightarrow [P_1.P_3.P_2] \;\; T_2.P_3 : [P_1.P_2.P_3] \rightarrow [P_1.P_3.P_3]}{(P_1.T_1);(T_2.P_2);(T_2.P_3) : [P_1.P_1] \rightarrow [P_1.P_3.P_3]} \; \textbf{\textit{Transitivité}}$$

Nous constatons bien qu'à ce niveau, le comportement d'un système concurrent (modéliser dans notre cas par un réseau de Petri) est décrit formellement par des déductions dans cette logique.

Exemple 2 :

Soit la théorie de réécriture suivante exprimant l'ensemble des booléens tel que :

$\Sigma = (\{bool\}, \{T: \rightarrow bool, F: \rightarrow bool, \_and\_ : bool \times bool \rightarrow bool\})$

$E = \{x \; and \; y = y \; and \; x, x \; and \; (y \; and \; z) = (x \; and \; y) and \; z\}$

$L = \{l_0, l_1\} \quad R = \{l_0 : [T] \rightarrow [T \; and \; T], \; l_1 : [F] \rightarrow [F \; and \; T]\}$

Etant donné la formule suivante :

$[F \; and \; F] \rightarrow [F \; and \; T \; and \; F \; and \; T \; and \; T]$

1) Nous pouvons prouver cette formule dans la théorie proposée en utilisant la règle de réécriture l_1 comme prémisse pour la règle de déduction congruence pour l'opération _and_ $\in \sum$:

$$\frac{[F] \rightarrow [F \; and \; T][F] \rightarrow [F \; and \; T]}{[F \; and \; F] \rightarrow [F \; and \; T \; and \; F \; and \; T]} \; \textbf{\textit{Congruence}}$$

2) Nous allons prendre la règle l_0 et la réflexivité comme prémisse pour la règle de déduction congruence avec la même opération _and_ $\in \sum$:

$$\frac{[F \; and \; T \; and \; F] \rightarrow [F \; and \; T \; and \; F][T] \rightarrow [T \; and \; T]}{[F \; and \; T \; and \; F \; and \; T] \rightarrow [F \; and \; T \; and \; F \; and \; T \; and \; T]} \; \textbf{\textit{Congruence}}$$

3) Ces deux derniers résultats seront utilisés comme des prémisses pour la règle de déduction transitivité pour obtenir le résultat attendu :

$$\frac{[F \; and \; F] \rightarrow [F \; and \; T \; and \; F \; and \; T][F \; and \; T \; and \; Fand \; T] \rightarrow [F \; and \; T \; and \; F \; and \; T \; and \; T]}{[F \; and \; F] \rightarrow [F \; and \; T \; and \; F \; and \; T \; and \; T]}$$

$\textbf{\textit{Transitivité}}$

5. Maude : Système basés sur la logique de réécriture

Maude est un système performant et un langage de programmation déclaratif avec des performances compétitives. Il est caractérisé par sa simplicité, expressivité, efficacité et par sa performance. Autrement dit, Maude est un langage qui offre peu de constructions syntaxiques

et une sémantique bien définie. En particulier, le sous langage de la logique équationnelle de Maude contient essentiellement OBJ3 comme sous langage. Les buts du projet de Maude étaient de supporter la spécification formelle exécutable, et d'élargir le spectre d'utilisation de la programmation déclarative et des méthodes formelles pour spécifier, réaliser et analyser des systèmes de haute qualité dans des secteurs comme : réseaux de communication, bio-informatique et l'informatique répartie, ... etc.

En outre, Maude offre une collection puissante d'outils formels supportant différentes formes de raisonnement logique pour vérifier des propriétés de programme comprenant : Un model-checker, un prouveur de théorème, un outil d'atteignabilité (accessibilité) et un Analyseur de cohérence ...etc.[23]

5.1. Différents modules du système Maude

En Maude, un module fournira une collection de sortes et une collection d'opérations sur ces sortes, ainsi que les informations nécessaires pour réduire et de réécrire des expressions entrées par l'utilisateur dans l'environnement Maude. Il y a trois types de modules sont définis dans Maude.

- Les modules fonctionnels permettent de définir les types de données.
- Les modules systèmes permettent de définir le comportement dynamique d'un système.
- Troisième type de module qui est « module orienté objet » mais il nous n'intéresse pas dans notre étude comme il peut être définie à l'aide des module systèmes.

5.1.1. Modules fonctionnels

Les modules fonctionnels servent pour la définition des types de données ainsi que les opérations qui sont utilisés par ces équations.

Un module fonctionnel est généralement entouré par les mots clés **fmod** <*Corps du module*> **endfm** où le corps du module spécifie une théorie dans la logique équationnelle.

La signature $\sum$ inclut des sortes (indiqués par le mot clé **`sort`**), des sous-sortes (spécifiés par le mot clé **`subsort`**) et des opérateurs (introduits avec le mot clé **`op`**).

La syntaxe des opérateurs est définie par les utilisateurs en indiquant la position des arguments par le symbole (_). L'ensemble ***E*** désigne les équations et les tests d'appartenance (qui peuvent être conditionnels) et ***A*** est un ensemble d'axiomes équationnels introduits comme attributs de certains opérateurs dans la signature$\sum$. Les équations sont spécifiées par le mot clé **`eq`** ou le mot clé **`ceq`** (pour les équations conditionnelles).

Dans un module fonctionnel, les équations sont utilisées comme des règles de simplification par lesquelles chaque expression, après substitution des variables, peut être évaluée et simplifiée à sa forme réduite dite forme canonique. Le résultat de la simplification d'un terme initial est unique quel que soit l'ordre d'application des équations. Les variables peuvent être déclarées dans les modules avec les mots clés **`var`** ou **`vars`**, ou introduites directement dans les équations et les tests d'adhésion, sous la forme d'une expression `var : sort`.

Exemple 3 :

Reprenons l'exemple 2 des réseaux de Petri précédent nous pouvons maintenant lui associer d'abord un module fonctionnel :

```
fmod PETRI-NET-SIGNATURE is
 sorts Place Marking.
 subsorts Place < Marking.
 ops P1 P2 P3: -> Place.
 op null: -> Marking.
 op _ _: Marking Marking -> Marking [ctor assoc comm id: null].
 op initial: -> Marking.
 eq initial = P1 P1.
endfm
```

5.1.2. Modules systèmes

Les modules systèmes sont des théories de réécriture .Ils permettent de spécifier le comportement d'un système concurrent. Les modules systèmes ajoutent à la définition des modules fonctionnels un ensemble de règles de réécritures (conditionnelles et inconditionnelles). Ils sont introduits par les mots clés ***mod*** <*Corps du module*> ***endm*** où le corps du module spécifie une théorie de réécriture R = ($\sum$, $E \cup A$, Φ, R). Les règles de réécriture R sont introduites avec les mots clés **rl** ou **crl.**

Elles sont spécifiées dans Maude avec la syntaxe :

$$crl\ [l]\ :\ t\ =>\ t'\ \textbf{\textit{if}}\ cond\ .$$

Si la règle est non conditionnelle, le mot clé **crl** est remplacé par **rl** et la clause « **if** *cond* » est omise.

Exemple 4 :

Nous reprenons le même exemple 1, pour lequel nous ajoutons des règles de réécriture associées aux transitions de réseau de Petri pour décrire son comportement**.**

```
mod PETRI-NET is
protecting PETRI-NET-SIGNATURE .
rl [T1] : P1 => P2.
rl [T2] : P1 => P1 P2 P3.
endm
```

Le module fonctionnel `PETRI-NET-SIGNATURE` a été importé dans ce module système.

5.2. La réécriture dans Maude

5.2.1. Cas des modules fonctionnels

La façon de réécriture des modules fonctionnels se termine avec une valeur simple comme résultats. Dans de tels modules, chaque étape de la réécriture est une étape du remplacement des termes par des égaux, jusqu'à ce qu'on trouve la valeur équivalente, entièrement évaluée (Forme Normale). Cette opération se fait à l'aide de la commande ***red*** (ou ***reduce***).

5.2.2. Cas des modules systèmes

Dans ce cas, une expression donnée à Maude avec la commande ***rew*** (ou *rewrite*) sera réécrite avec la stratégie de défaut jusqu'à ce que aucune règle ne puisse être appliquée. Et puisqu'un tel calcul en général peut ne pas se terminer, Maude permet à l'utilisateur d'indiquer le nombre maximum, joint entre parenthèses, des applications de règles permises en exécutant la commande rew. Nous donnons au-dessous plusieurs types d'exécutions pour notre module de réseau de Petri avec et sans une limite pour la commande rew.

6. Sémantiques des RdPs dans la logique de réécriture

Dans cette section, nous allons présenter les aspects structurels et comportementaux de la sémantique existante et ceux de la sémantique améliorée du réseau de Petri afin de bien identifier leur différence et aussi pour mieux faciliter la tâche de transformation. Ce dernier distingue clairement les places de leurs jetons afin de surmonter naturellement quelques limites de la sémantique existante.

6.1. Sémantique existante des réseaux de Petri

La sémantique existante a été proposée par Mark-Oliver Stehr et autres[24]. Ensuite, elle a été généralisée pour une large gamme de réseaux de Petri. Pour présenter cette sémantique, on va utiliser le réseau de Petri présenté dans la figure 15 (a), décrivant le comportement du distributeur automatique qui vend des gâteaux et des pommes ; un gâteau coute 1 dollar et une pomme 3 quarters (pièce de 25 centimes). En plus, cette machine accepte uniquement les dollars et pour ça la machine permet de convertir quatre quarters en dollars.

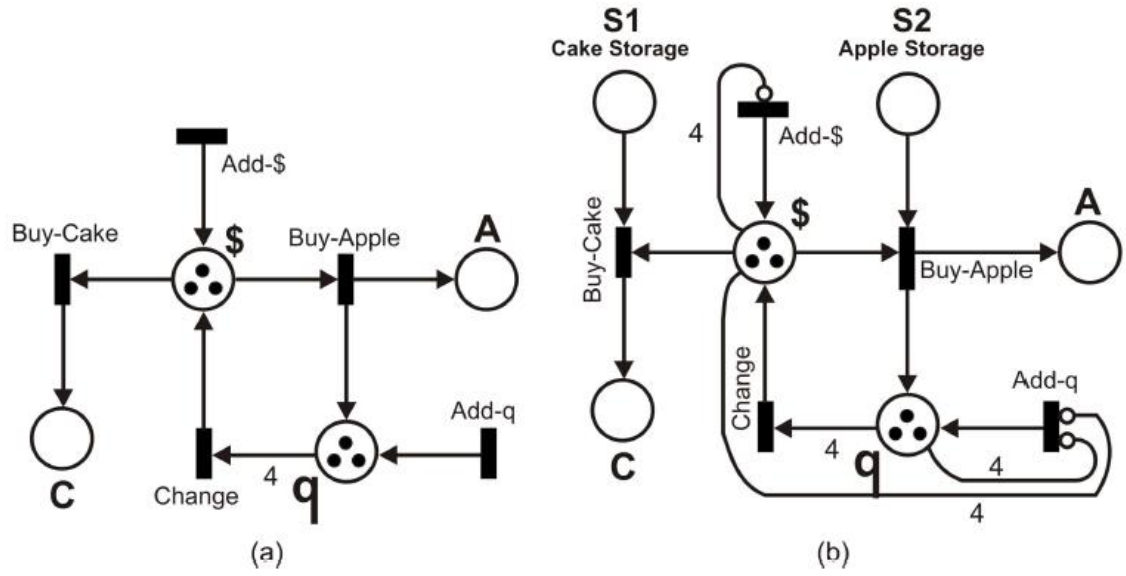


Figure 15 : Réseaux de Petri décrivant le comportement du distributeur automatique

6.1.1. Aspects structurels

Les types essentiels dans un réseau de Petri sont les suivants : **place** et **marking**. Ces types sont définis comme sort et des **subsorts** comme suit :

```
sorts Place Marking .
subsort Place < marking .
```

Ensuite, on doit déclarer les noms des places du réseau de Petri comme suit :

```
ops C A $ q : → Place .
```

Puis, le marquage d'un réseau de Petri peut ensuite être défini avec un opérateur d'union multi-ensembles (multiset) fini, comme suit :

```
op null : -> Marking .
op _ _ : Marking Marking -> Marking [ctor assoc comm id: null] .
```

Selon cette déclaration, nous pouvons voir qu'un marquage est représenté comme un élément du type **Marking** et que l'union de deux marquages est un nouveau marquage. De plus, l'attribut "assoc" (resp, "comm") est utilisé pour déclarer que l'opérateur est associatif (resp, commutatif). Le marquage vide est représenté par la constante nulle.

Enfin, le marquage initial peut être déclaré à l'aide d'un opérateur et une équation comme suit :

```
op initial : -> Marking . eq initial = $ $ $ q q q .
```

Dans ce cas, on voit que seuls les noms des places contenant des jetons apparaissent dans la déclaration de l'état initial et avec une occurrence égale au nombre de jetons qu'ils contiennent.

6.1.2. Aspects comportementaux

Le marquage d'un réseau de Petri est lié aux franchissements de ses transitions. Pour cela, la spécification de l'opération de franchissement d'une transition **T** consiste à le décrire par une règle de réécriture étiquette avec la syntaxe suivante :

```
rl[T] : (termset1) => (termset2)
```

Dans une telle règle, le terme termset-1 (resp, termset-2) contient l'ensemble des places d'entrées (resp, sorties) de la transition T, où le nom de chaque place (d'entrée) (resp, sortie) est répété dans la partie gauche (resp, droite) de la règle de réécriture, autant de fois que Pre(P,T) ou Post(T,P).

Dans ce contexte, une transition source (resp, puit) est un cas particulier et sa règle de réécriture correspondante reste la même et une variable de type **Marking** (M, par exemple) est ajoutée aux termset-1 et termset-2.

En suivant ces étapes, nous donnons la spécification complète du réseau de Petri (Figure 15 (a)) comme suit :

```
fmod PETRI-NET-SIGNATURE  is
sorts Place Marking .
subsorts Place <  Marking .
op null : -> Marking .
ops C A $ q : ->  Place .
op _ _: Marking Marking -> Marking
[assoc comm id : nil] .
op initial : -> Marking .
eq initial = $ $ $ q q q .
endfm
```

```
mod VENDING-MACHINE is
Prot PETRI-NET-SIGNATURE .
var M : Marking .
rl [add-$] : M => M $. .
rl [add-q] : M => M q.
rl [buy-C] : $ => C.
rl [buy-A] : $ => A q.
rl [changer] : q q q q => $.
endm
```

6.1.3. Limites de cette sémantique

Bien que cette sémantique est très familier et compacte en termes de la taille du code source de la spécification d'un RdP, elle présente aussi quelques inconvénients :

1 : La confusion sémantique entre une place et un jeton. Car le type place, qui est censé représenter des places, il sert aussi à indiquer le jeton d'une place.

2 : L'incapacité de compter naturellement (ou d'afficher) le nombre de jetons existants dans une place sans ajouter d'autre équations supplémentaires, car elle gère la multiplicité des arcs avec une manière si triviale (répétition des noms des places). Par conséquent, il n'est pas possible de spécifier naturellement un réseau de Petri avec arcs inhibiteurs.

6.2. Sémantique améliorée des réseaux de Petri

Pour mieux illustrer l'idée de cette sémantique. Et bien identifier les différences avec la sémantique existante, nous allons utiliser le réseau de Petri présenté dans la Figure 15 (b). Dans cette machine, les places S1 et S2 ont été ajoutés pour décrire la capacité du distributeur automatique (supposée à 50) pour la livraison d'articles (gâteau et pomme).

6.2.1. Aspects structurels

La définition d'une place consiste à utiliser son nom (**sort : Placename**) suivi du nombre de jetons qu'elle détient (**sort : Int**). Cependant, le type de base **marking** est conservé pour modéliser le comportement dynamique d'un réseau de Petri. Par conséquent, la nouvelle signature correspondante est donnée comme suit :

```
sorts PlaceName Place Marking .
subsort Place < Marking .
ops C A $ q S1 S2 :-> Placename .
op <_,_> : PlaceName int -> Place [ctor] .
op _ _ : Marking Marking -> Marking [assoc comm id:null] .
```

Par conséquent, la déclaration d'un état initial donne un vue instantané claire du marquage initial du système et permet aux utilisateurs (observateurs ou developpeur) de connaître tous les noms des places ainsi que les jetons qu'elles détiennent depuis le marquage initial.

```
op initial : -> Marking .
eq initial = < $,3 > < q,3 > < c,0 > < a,0 > S1,50> <S2,50>.
```

6.2.2. Aspects comportementaux

Le nouveau marquage M' d'un réseau de Petri marqué avec M, est obtenu après le franchissement d'une de ses transitions T. Ceci est défini comme suit :

$$M' = M - Pre(P_i, T) + Post(T, P_j).$$

En plus, dans cette sémantique, on doit énumérer l'ensemble des places d'entrées et de sorties de la transition T sur les côtés gauche et droit de la règle de réécriture correspondante.

Ensuite, la règle de réécriture décrivant franchissement d'une transition T est souvent conditionnelle avec les la fonction Pre(P,T). Par conséquent, elle est donnée comme suit :

```
crl[Label] : <Let_hand_side> => <Right_hand_side> if Cond .
```

Dans ce cas, les deux côtés d'une règle de réécriture contiennent l'état des places d'entrées et de sorties avant (resp, après) le franchissement. En plus, la condition d'activation ne sera pas si nécessaire dans le cas d'une transition source, car elle est toujours activée.

Enfin, nous présentons la spécification du réseau de Petri (Figure 15 (b)) conformément à cette sémantique améliorée.

```
fmod NEW-PETRI-NET-SIGNATURE-1 is
protecting INT .
sorts PlaceName Place Marking .  subsort Place < Marking .
ops C A $ q S1 S2 : -> PlaceName .
op <_,_> : Placename int -> Place [ctor] .
ops initial null : -> Marking .
op _ _ : Marking Marking -> Marking [assoc comm id: null] .
eq initial = <$,3> <q,3> <C,0> <A,0> <S1,50> <S2,50>.
endfm
```

```
mod NEW-VENDING-MACHINE-1 is
inc NEW-PETRI-NET-SIGNATURE- 1 .
vars x y z t : Int .
crl [add-$] : < $,x > => < $,x + 1 > if (x < 4) .
crl [add-q] : < $,x > < q,z > => < $,x > < q,z + 1 > if (z < 4)
       and (x < 4) .
crl [buy-c] : < $,x > < C,y > < S1,z > => < $,x - 1 > < C,y + 1 >
       < S1,z - 1 > if (x >= 1) and (z >= 1) .
crl [buy-a] : < $,x > < A,y > < q,z > < S2,t > => < $,x - 1 >
 < A,y + 1 > < q,z + 1 > < S2,t - 1 > if (x >= 1) and (t >= 1) .
crl [change] : < $,x > < q,z > => < $,x + 1 > < q,z - 4 >
        if (z >= 4) .
endm
```

7. Conclusion

Nous avons présenté dans ce chapitre, les notions relatives à la compréhension des concepts de la logique de réécriture et du système Maude. Dans un premier temps, nous avons présenté le concept le plus important (théorie de réécriture) pour la description des systèmes concurrents. Ensuite, nous avons introduit le système Maude qui un langage caractérisé par la simplicité, la performance et l'expressivité. Enfin, nous avons présenté les sémantiques (existante et améliorée) basées sur la logique de réécriture pour la description des réseaux de Petri.

Chapitre 3
Conception et aspect d’implémentation

1. Introduction

La tâche de conception des systèmes a toujours besoin d'un langage ou d'une méthode de spécification et de modélisation pour faciliter la création ainsi que l'analyse des modèles du système à développer.

Dans ce chapitre que nous allons présenter les détails de l'approche proposé pour générer la spécification Maude d'un réseau de Petri. Puis, on cite quelques travaux existant pour attirer l'attention à notre contribution. Enfin, on passe à la modélisation et la conception de l'outil de transformation automatique des réseaux de Petri. Pour cela, nous présentons les diagrammes UML les plus essentiels.

2. Travaux similaires et contributions

2.1. Travaux existantes

Bien que la sémantique existante pour la spécification formelle des réseaux de Petri dans la logique de réécriture est un peu ancienne, il n'existe qu'un petit nombre de travaux qui ont été réalisés pour sa génération automatique.

Premièrement, on cite le travail de Noura Boudiaf et autres[25] pour l'édition, la simulation et l'analyse de réseaux de termes algébriques simultanés étendus (ECATNets) via une collection quasi complète d'outils d'analyse formelle. La deuxième approche a été proposée par Elhillali Kerkouche et autres[26] qui sont basé sur l'utilisation des transformations de graphe par l'utilisation de l'outil ATOM3 (A Tool for Multi-formalism and Meta-Modelling). Leur but de la génération automatiquement la spécification Maude était pour des fins de simulation et d'analyse. Troisièment, on cite le second travail de Noura Boudiaf et autres[27] où ils proposent un outil graphique permettant une traduction bidirectionnelle des réseaux de Petri colorés vers Maude et inversement. Quatrièment, le travail le plus étroitement lié à notre sujet est celui Ammar Boucherit et autres[28] où ils ont proposé un algorithme pour la génération automatique de spécification Maude basé sur la sémantique existante pour les réseaux de Petri. En effet, on doit être leur reconnaissant parce que ce travail nous a aidés à bien comprendre pratiquement la problématique de notre sujet.

Il est à noter que les outils précités (sauf celui de Boucherit28) ont été tous présentés sans mettre leur outil disponible aux utilisateurs, ni d'algorithme détaillant le fonctionnement de la procédure de génération automatique. En outre, le présent travail ne peut être comparé

strictement aux travaux précités, car ils ne poursuivent pas le même objectif (ECATNets, réseaux de Petri colorés et la sémantique utilisée).

Enfin, le principal avantage de notre travail par rapport ces travaux est qu'il supporte les réseaux de Petri avec arcs inhibiteurs. En plus, il génère la spécification pour les réseaux de Petri suivant les deux sémantiques, existante et améliorée, pour fournir une base théorique solide et facilite leur simulation et leur analyse formelle à l'aide des outils du système Maude.

2.2. Autres contributions

- Le développement d'un éditeur n'était pas exigé dans la définition de notre sujet, mais parce que nous avons eu beaucoup de temps (du fait de la quarantaine causée par le Covid-19) ; on a pu le développer après la décomposition des tâches lors de la vacance.
- On a développé un outil complet pour la génération automatique des spécifications Maude pour un réseau de Petri selon la sémantique existante et améliorée.

3. Détails de l'approche de transformation

Pour mieux expliquer l'approche proposée, on doit le décomposer en deux soutaches :

3.1. Du fichier PNML vers les matrices d'incidences

On doit noter qu'on a utilisé dans notre prototype les fichiers PNML exportés des outils P3 et PIPE. En plus, le processus d'obtention des matrices à partir d'un fichier PNML est basé sur l'utilisation du langage XSLT et passe par les étapes suivantes :

- **Purification** : est le fait d'extraire les informations plus ou moins essentiel telles que l'ensemble des places, transitions les arcs ainsi que le marquage initial dans le but de faciliter l'opération de création des matrices d'incidences.
- **Création des matrices** : cette opération est basée sur l'opération précédente où le fichier purifié sera manipulé pour créer et remplir les matrices d'incidences.

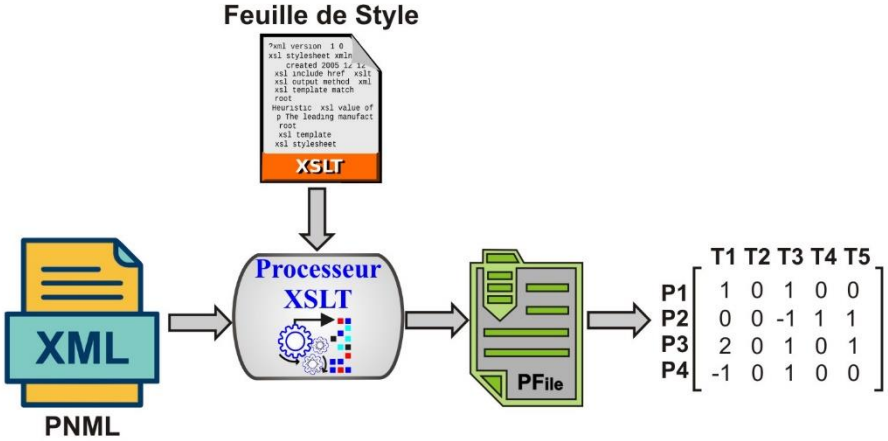


Figure 16: Les étapes de transformation d'un fichier PNML vers matrices d'incidences.

3.2. De la matrice d'incidence vers la logique de réécriture

Le principe de génération des spécifications Maude à partir des matrices d'incidence peut être résumé dans les étapes suivantes :

- Le nombre de règles de réécritures est égale au nombre de transitions composant le réseau de Petri (nombre de colonne de la matrice d'incidence).
- une règle de réécriture décrivant le franchissement d'une transition peut souvent être conditionnelle pour inclure la condition d'activation d'une telle transition.

  Par conséquent, une telle règle est donnée comme suit :

```
crl[Label] : <Let_hand_side> => <Right_hand_side> if Cond .
```

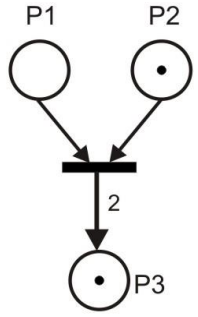
- Label : le libellé (nom) de la transition.
- Left-Hand-Sid : la partie gauche de la règle (LHS).
- Right-Hand-Side : la partie droite de la règle (RHS).
- Cond : est une expression logique qui représente la condition d'activation de la transition et qui est liée à la matrice Pre(P,T).

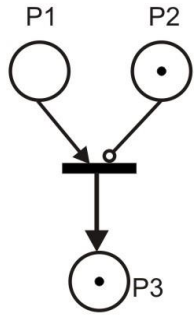
Dans la partie gauche (LHS), on met toutes les places d'entrées et de sorties avec leurs nombres de jetons avant le franchissement. Alors que dans la partie droite (RHS), on met les termes du LHS mais après le franchissement. En plus, on ajoute la condition de franchissement sauf le cas d'une transition source.

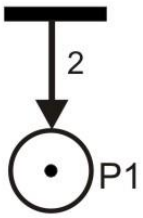
Pratiquement, chaque colonne représente une transition où les éléments non nuls de la matrice d'entrée Pre(P,T) et de celle de sortie Post(T,P) doivent être mis dans la partie gauche (LHS) de la règle de réécriture. En plus, on met aussi les mêmes termes du (LHS) dans la partie droite (RHS), où on met les changements auront lieu sur les places que ce soit d'entrées (décrémentées) ou de sortie (incrémentées). Finalement, la condition est préparée à partir des éléments de la matrice d'incidence Pre(P,T).

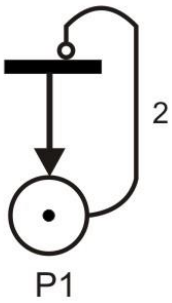
3.3. Description des cas possible d'une transition en logique de réécriture

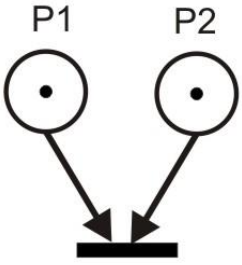
On va essayer de résumer la description des différents cas possible d'une transition en logique de réécriture.

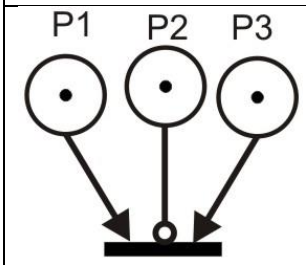
Transition ordinaire	**Règle de réécriture**
	**LHS : <P1, x1 > <P2, x2 > <P3, x3 >** **RHS : <P1, x1 - 1 > <P2, x2 - 1 > <P3, x3 + 2 >**
	Condition
	if (x1 >= 1) and (x2 >= 1).

Transition ordinaire avec arc inhibiteur	**Règle de réécriture**
	**LHS : <P1, x1 > <P2, x2 > <P3, x3 >** **RHS : <P1, x1 - 1 > <P2, x2 > <P3, x3 + 1 >**
	Condition
	If (x1 >= 1) and (x2 < 1) .

Transition source	**Règle de réécriture**
	**LHS : <P1, x1 >** **RHS : <P1, x1 + 2 >**
	Condition
	c'est une règle inconditionnelle

Transition source avec arc inhibiteur	**Règle de réécriture**
	**LHS : <P1, x1 >** **RHS : <P1, x1 + 1 >**
	Condition
	if (x1 < 2).

Transition puit	**Règle de réécriture**
	**LHS : <P1, x1 > <P2, x2 >** **RHS : <P1, x1 – 1 > <P2, x2 – 1 >**
	Condition
	if (x1 >= 1) and (x2 >= 1).

Transition puit avec arc inhibiteur	**Règle de réécriture**
	**LHS : <P1, x1 > <P2, x2 > <P3, x3 >** **RHS : <P1, x1 - 1 > <P2, x2 > <P3, x3 - 1 >**
	Condition
	if (x1 >= 1) and (x2 < 1) and (x3 >= 1) .

Exemple illustratif et récapitulatif

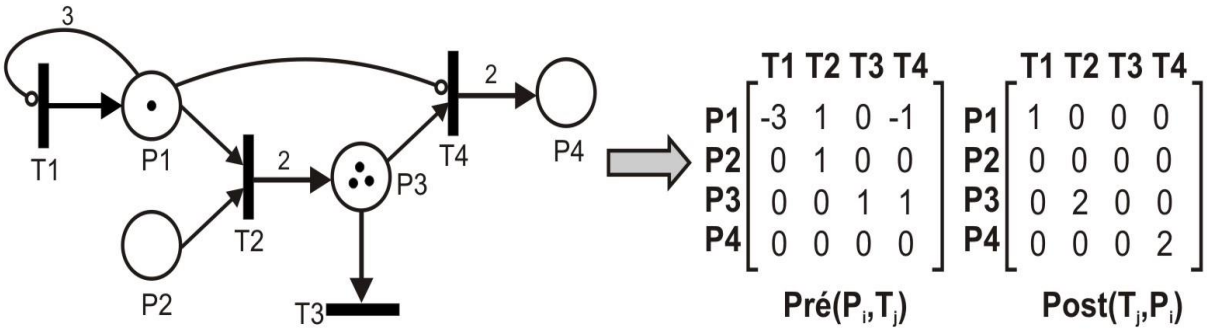


Figure 17 : Un réseau de Petri avec ses matrices d'incidences

La spécification qui sera générée pour ce réseau de Petri doit se passer par les étapes suivantes :

- Préparation du LHS d'une règle de réécriture :

On compte le nombre des éléments non nuls situant dans des emplacements différents dans les deux matrices (appelé NB). NB représente le nombre des termes dans LHS ainsi que le nombre des variables utilisées pour manipuler le franchissement dans une transition T.

- Préparation du RHS d'une règle de réécriture :

On utilise les mêmes termes et variables du LHS et on met les modifications nécessaires pour chaque variable d'une place selon la valeur de l'élément correspondant dans la matrice Post(T,P), Pre(P,T).

- Le type d'une règle est toujours conditionnel sauf si la transition est source car elle sera toujours franchissable, c-à-d, qu'elle ne nécessite pas de condition.
- La condition d'une règle dépend essentiellement par la matrice Pre(P,T).

En suivant ses étapes, on avoir les règles de réécriture correspondantes suivantes :

```
crl [ T1 ] : < P1,x1 > => < P1,x1 + 1 > if ( x1 < 3 ) .
crl [ T2 ] : < P1,x1 > < P2,x2 > < P3,x3 > => < P1,x1 - 1 >
   < P2,x2 - 1 > < P3,x3 + 2 > if ( x1 >= 1 ) and ( x2 >= 1 ) .
crl [ T3 ] : < P3,x1 > => < P3,x1 - 1 > if ( x1 >= 1 ) .
crl [ T4 ] : < P1,x1 > < P3,x2 > < P4,x3 > => < P1,x1 > < P3,x2 - 1 >
       < P4,x3 + 2 > if ( x1 < 1 ) and ( x2 >= 1 ) .
```

4. La conception de l'application

4.1. Description générale du système

L'objectif principal de notre travail est de transformer de réseaux de Petri vers leur nouvelle sémantique basée sur la logique de réécriture. Les étapes à suivre pour réaliser ce travail peuvent être données comme suit :

1 : On prépare le réseau de Petri cible à l'aide d'un des outils existants de réseaux de Petri (PIPE ou P3 par exemple) et exporter sa représentation sous forme PNML et/ou de développer un éditeur (from scratch) pour dessiner des réseaux Petri afin de faciliter l'obtention de la description matricielle d'un RdP.

2 : On exploite le processeur XSLT avec une feuille de style pour purifier le fichier PNML, c.-à-d., éliminer les informations inutiles telles que celles correspondantes à représentation graphique.

3 : Création des matrices d'incidence et leur passage au processeur de transformation.

4 : Exportation des spécifications générées dans un fichier Maude.

La Figure 18 présente les étapes de description générale du système développé.

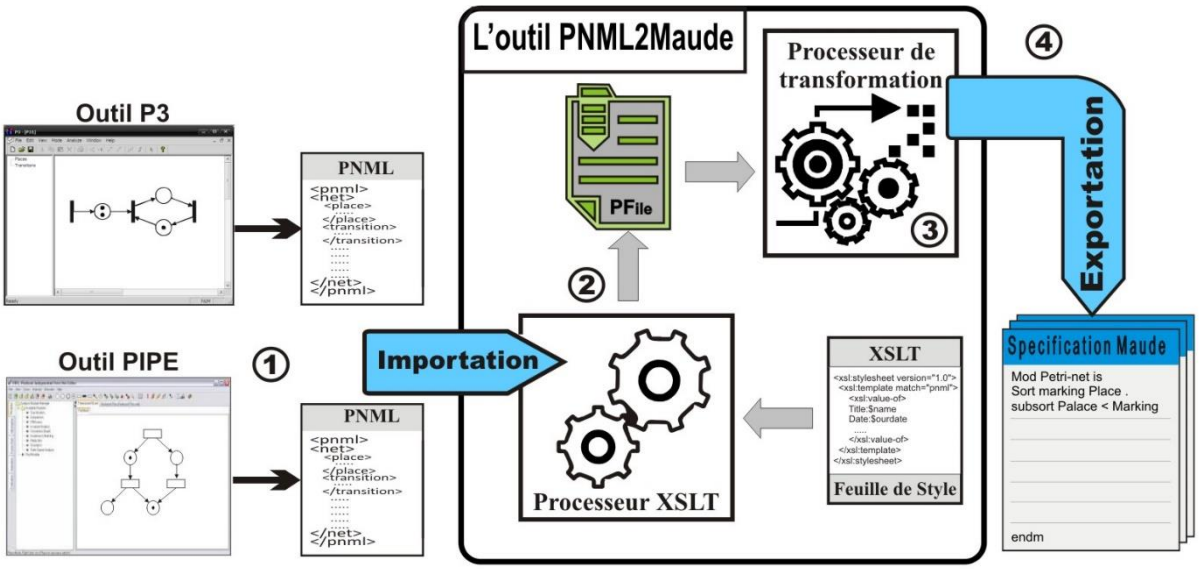


Figure 18 : De description générale du système.

4.2. Les diagrammes UML importantes

Les diagrammes UML sont généralement utilisés pour les gros projets pour mieux acquérir les besoin, faciliter la représentation des composants des systèmes orientés objet et simuler sa structure statique et dynamique.

Dans notre projet, nous nous limiterons à l'utilisation du diagramme de cas d'utilisation, du diagramme de séquence et du diagramme de classe.

4.2.1. Diagramme cas d'utilisation

Le rôle des diagrammes de cas d'utilisation est de donner une vision globale du comportement fonctionnel d'un système logiciel lors de la première étape de modélisation.

Dans notre application l'utilisateur peut dessiner un réseau de Petri en utilisant notre éditeur ou bien un autre éditeur permettant l'exportation de la description sous forme PNML. Ensuite, le cas le plus important est la génération automatique de spécification Maude pour le réseau de Petri choisis.

Le diagramme de cas d'utilisation de notre système est représenté dans la figure suivante :

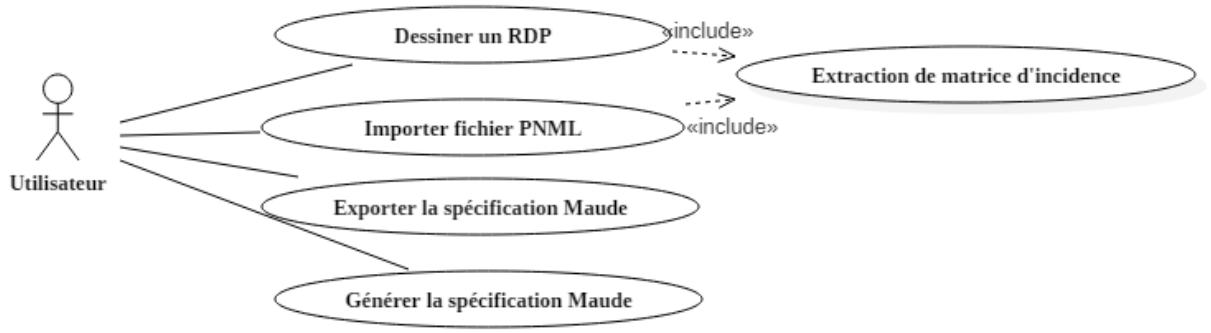


Figure 19: Diagramme cas d'utilisation générale.

4.2.2. Diagramme de séquence

Le diagramme de séquence fait partie des diagrammes comportementaux (dynamique) et plus précisément des diagrammes d'interactions.

- Il permet de représenter des échanges entre les différents objets et acteurs du système en fonction du temps.
- A moins que le système à modéliser soit extrêmement simple, nous ne pouvons pas modéliser la dynamique globale du système dans un seul diagramme.

4.2.2.1.Diagramme de Séquence " Dessiner un RDP "

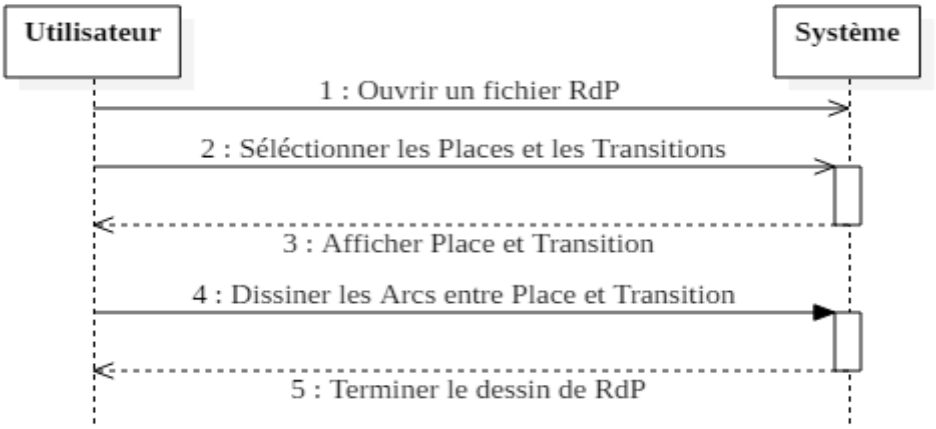


Figure 20 : Diagramme de séquence (Dessiner un réseau de Petri).

Le scénario :

- L'utilisateur demande pour ouvrir un fichier RdP.
- L'utilisateur sélectionne les places et transitions.
- Le système affiche les places et transitions.
- L'utilisateur dessine les arcs entre places et transitions.
- Le système affiche RdP final et fin de dessin.

4.2.2.2.Diagramme de Séquence " Importer un RDP "

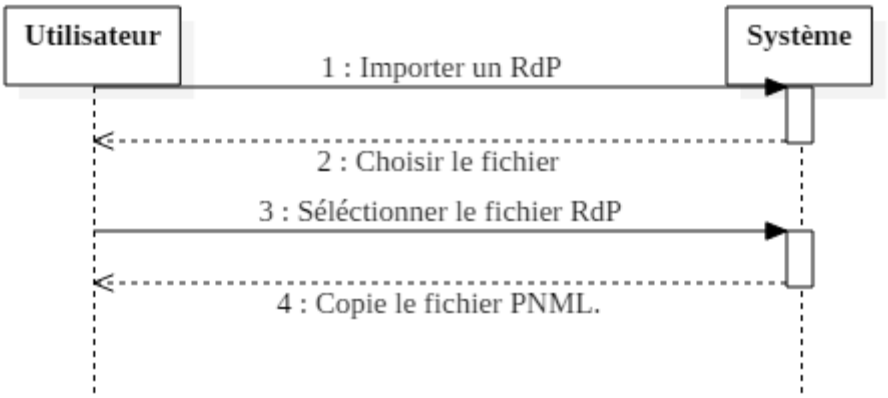


Figure 21 : Diagramme de séquence (Importer un réseau de Petri).

Le scénario :

- L'utilisateur demande pour importer un réseau de Petri.
- Le système demande de choisir le ficher à importer.
- L'utilisateur précise le ficher à importer.
- Le système copie le fichier PNML.

4.2.2.3.Diagramme de Séquence " Générer la spécification Maude"

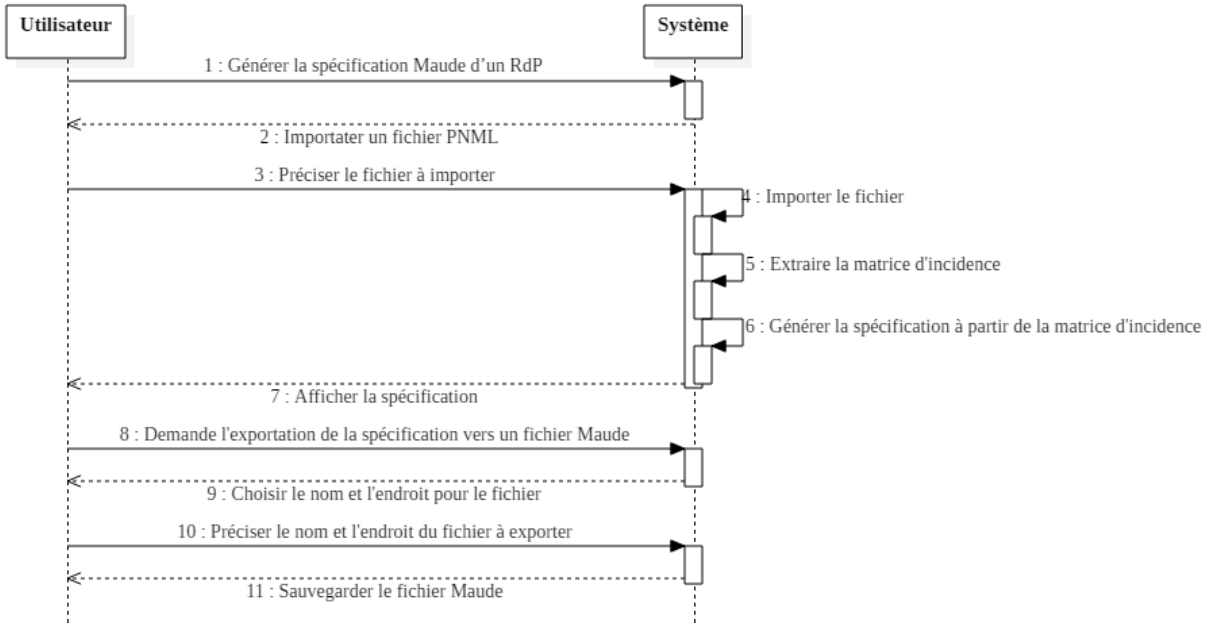


Figure 22 : Diagramme de séquence (Générer la spécification Maude (cas du fichier PNML)).

Le scénario :

- L'utilisateur demande de générer la spécification Maude d'un RdP.
- Le système demande l'importation d'un fichier PNML.
- L'utilisateur précise le fichier à importer.
- Le système importe le fichier.
- Le système extraire la matrice d'incidence.
- Le système génère la spécification à partir de la matrice d'incidence.
- Le système affiche la spécification.
- L'utilisateur demande l'exportation de la spécification vers un fichier Maude.
- Le système demande de choisir le nom et l'endroit pour le fichier.
- L'utilisateur préciser le nom et l'endroit du fichier à exporter.
- Le système sauvegarde le ficher Maude.

4.2.3. Diagramme de classe

Le diagramme de classe a pour but de décrire la structure des entités (classes) manipulées par les utilisateurs, leurs compositions et leurs associations. Comme il permet de fournir une représentation abstraite des objets du système qui vont interagir pour réaliser les cas d'utilisation.

Le diagramme de classes de notre système est présenté dans la figure 23.

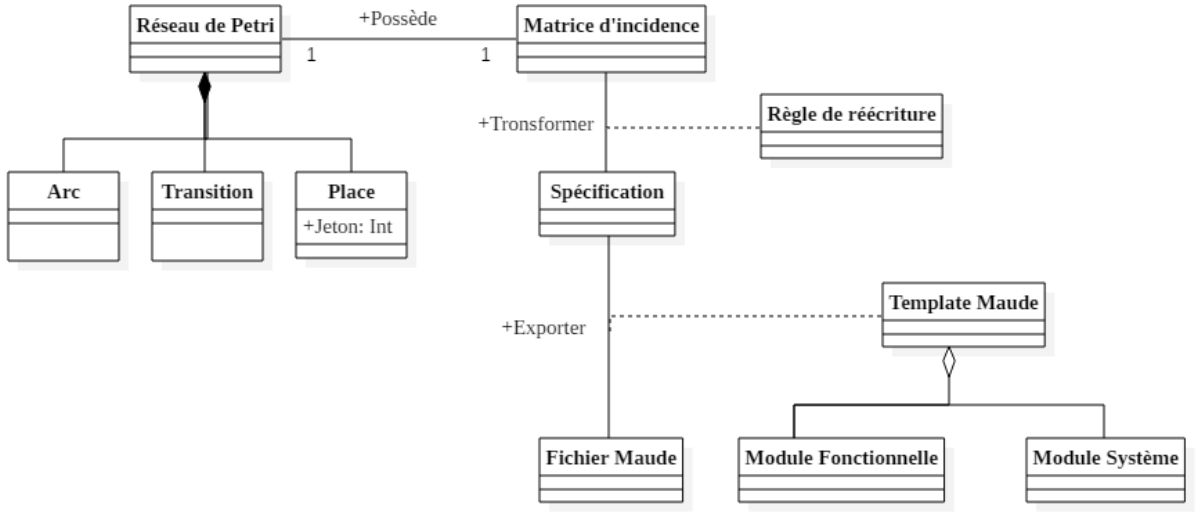


Figure 23 : Diagramme de classe générale.

4.3. L'environnement de développement

Le langage de programmation utilisé pour implémenter l'outil PNML2Maude est Delphi 7, un environnement complet de développement rapide d'applications (RAD) de bureau, mobiles, Web et de console en langage Pascal orienté objet développé par Borland (Inprise Corporation). Les programmes développés sous Delphi fonctionnant sous Windows. Delphi apparut alors comme une alternative viable pour beaucoup de développeurs souhaitant créer des programmes standards sous Windows.

D'autre part, on a utilisé le XSLT, un langage de programmation déclaratif permet de transformer un document XML dans un autre format, tel PDF, XML ou HTML en se basant sur des feuilles de style qui contiennent les règles de transformation (les template rules). Pratiquement, on a utilisé MSXSL, un outil précompilé fournit par Microsoft permettant de réaliser des transformations XSL.

4.3.1. Outils de réseau de Petri

- **P3**

  Ce projet a été créé dans le cadre des efforts de recherche du groupe de recherche GoodOldAI. Le but du projet est de rechercher l'utilisation des réseaux de Petri sur le Web sémantique. Les principaux objectifs de la recherche sont :

- Développement d'un logiciel pédagogique permettant la modélisation en utilisant la notation graphique de réseau de Petri.
- Mise en œuvre de différents outils d'analyse de réseaux de Petri (par exemple arbre d'accessibilité, équations matricielles) échange de modèles de réseaux de Petri avec d'autres outils logiciels utilisant XML et le langage de balisage de réseau Petri (PNML)
- prise en charge de la création de documents Web sous la forme de documents SVG (Scalable Vector Graphic) qui contiendront un graphe de réseau de Petri

Cet outil est téléchargeable d'ici :

http://www.sfu.ca/~dgasevic/projects/P3net/Download.htm

- **PIPE**

PIPE est un outil open source indépendant de la plateforme pour créer, simuler et analyser les réseaux de Petri, y compris les réseaux de Petri stochastiques généralisés. Un guide de l'utilisateur pour savoir comment utiliser les fonctionnalités de PIPE 5 ainsi que l'outil même sont disponible ici :

http://sarahtattersall.github.io/PIPE/

5. Présentation de l'outil développé

5.1. Interface principale

L'interface principale de notre outil est illustrée dans la figure suivante.

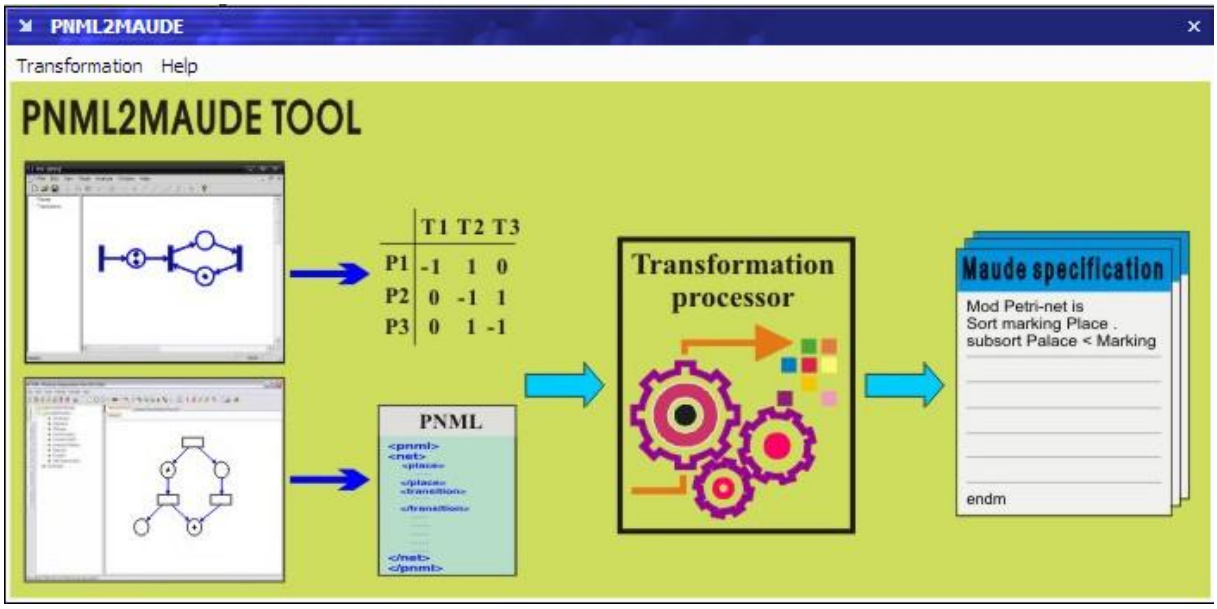


Figure 24: L'interface principale de l'outil PNML2Maude.

5.2. Cas d'utilisation (Barre de menu)

Il est possible d'utiliser notre outil avec trois possibilités de représentation des RdPs.

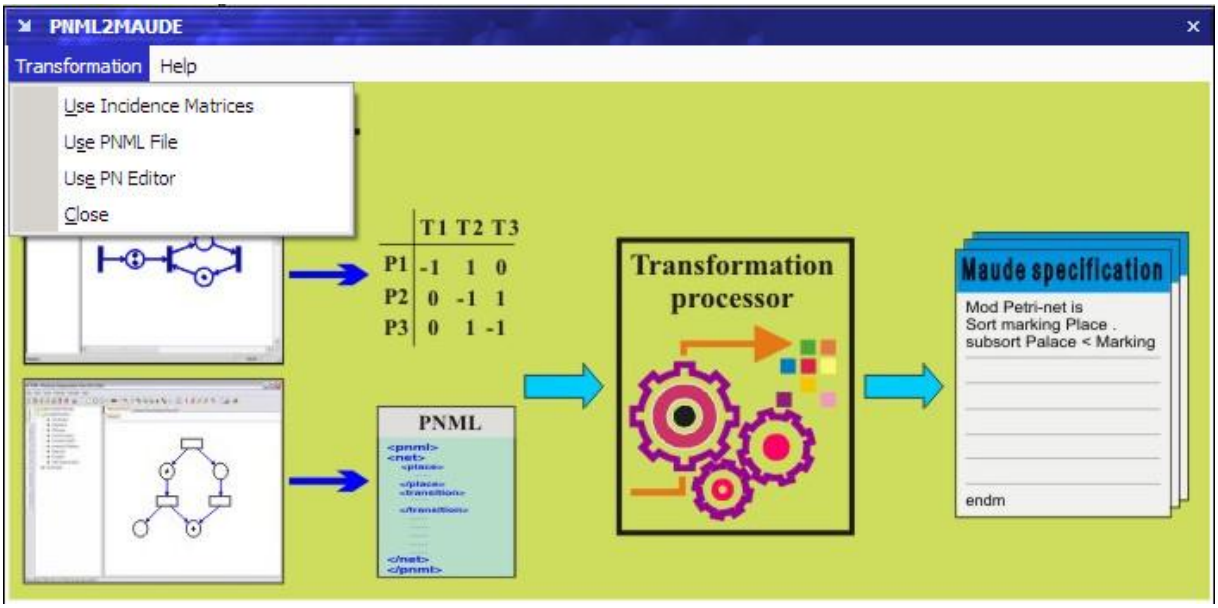


Figure 25: Les cas possible d'utilisation de l'outil PNML2Maude

5.3. Interface pour utiliser un fichier PNML

La Figure 26 présente l'interface de l'outil PNML2Maude pour le cas d'importation d'une représentation PNML du RdP.

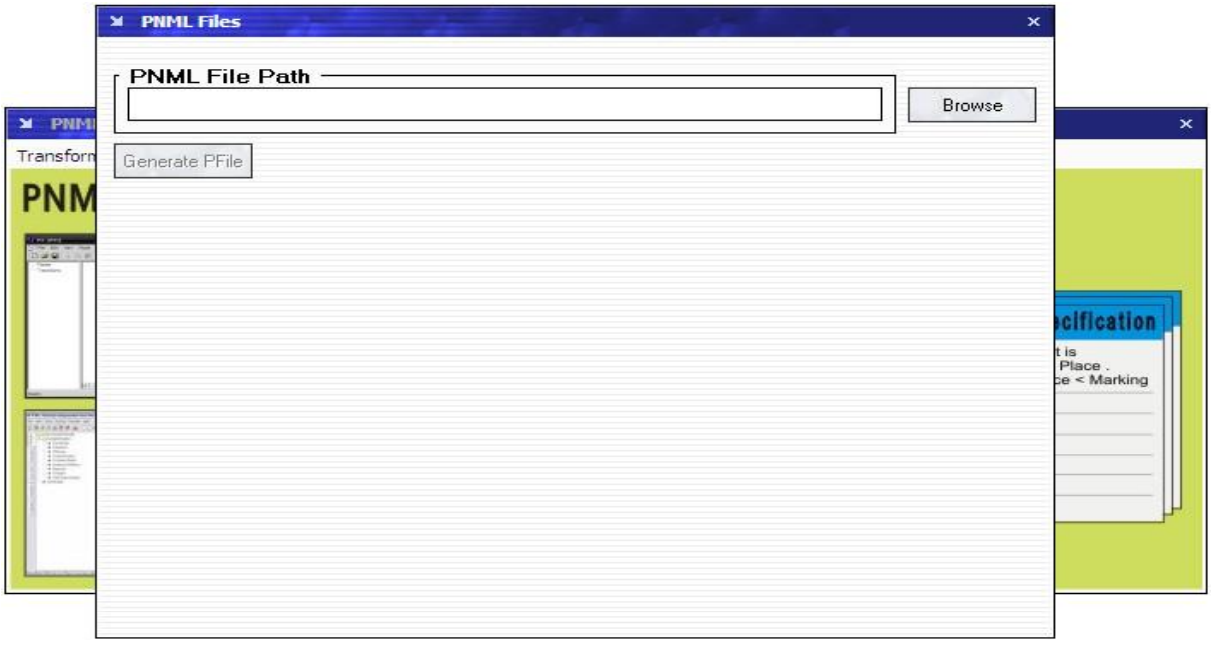


Figure 26: L'interface du PNML2Maude pour importer une représentation PNML

5.4. Interface pour utiliser les matrices d'incidence

Les Matrice d'incidences peuvent être spécifiées comme illustré dans la figure 27.

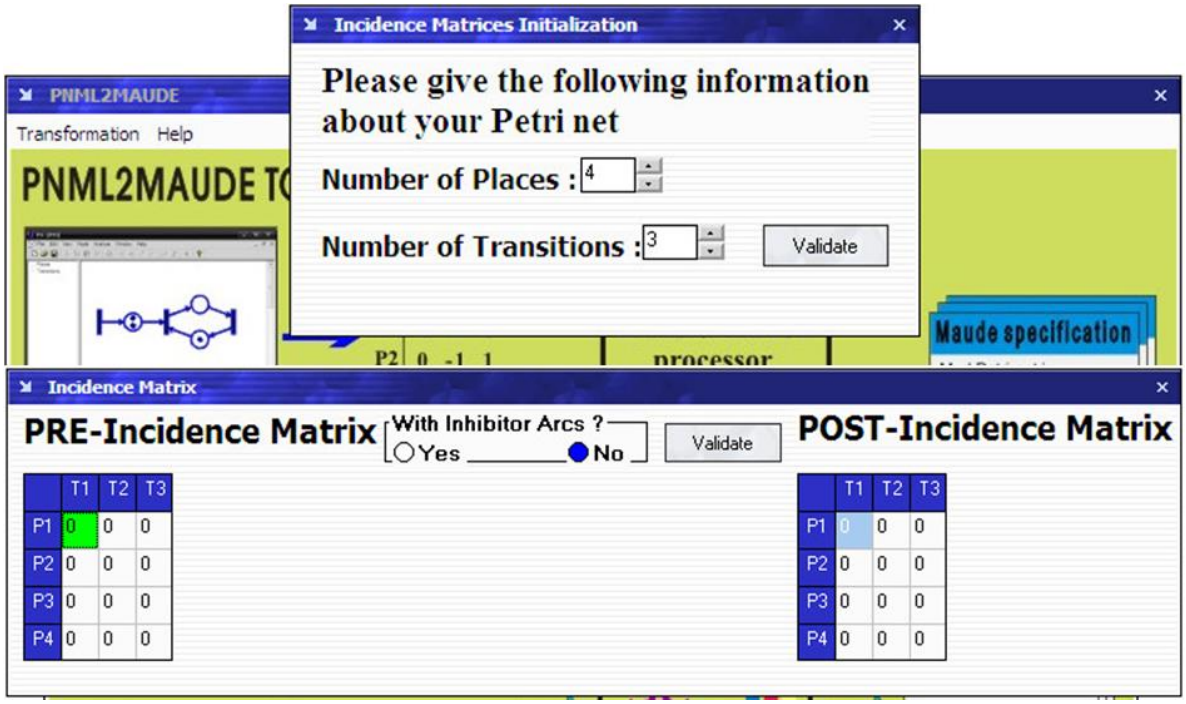


Figure 27: Utiliser la représentation matricielle d'un RdP

5.5. Interface pour utiliser un éditeur RdP

Peuvent utiliser cette interface (figure 28) pour dessiner un RdP.

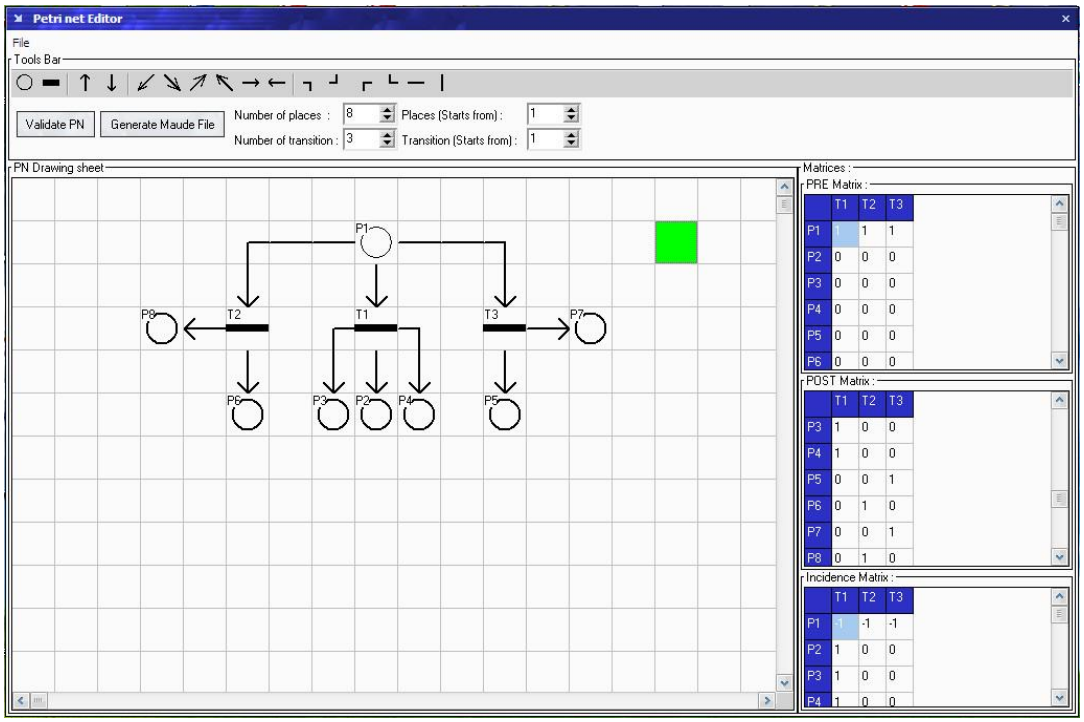


Figure 28: L'interface d'un éditeur RdP

5.6. Fenêtre de spécification générée

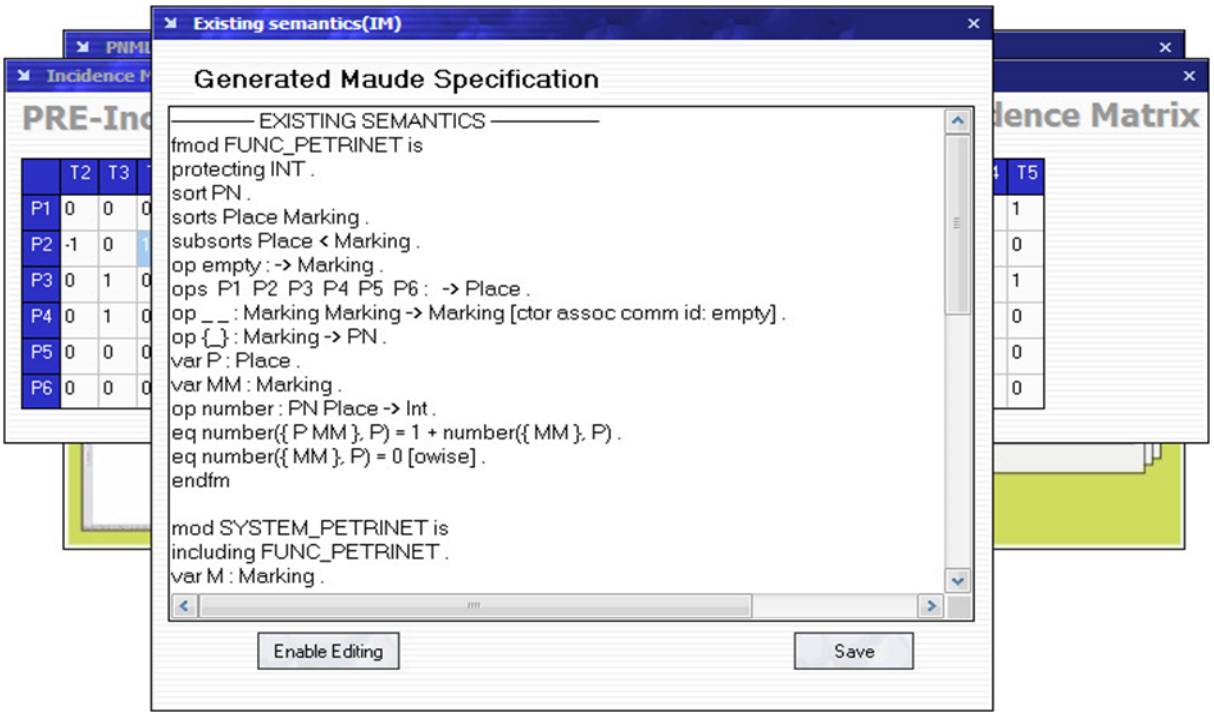


Figure 29: Fenêtre de spécification générée par l'outil PNML2Maude

6. Conclusion et Perspectives

Dans ce chapitre, on a présenté la conception de notre outil. Car les fonctionnalités sont assez claires, on a utilisé seulement le diagramme de cas d'utilisation et le diagramme de séquences pour présenter la partie dynamique. La partie statique a été illustrée par le diagramme de classes.

Puis, on a présenté l'outil développé pour la génération automatique de spécification Maude pour un réseau de Petri suivant la sémantique améliorée en se basant sur sa description PNML.

Enfin, nous allons l'intention – dans le future proche – de développer un une version web de notre outil et/ou de développer un plug-in pour Eclipse afin d'assurer une large distribution de notre outil.

Bibliographie

[1] Ölveczky, Peter Csaba, and José Meseguer. "Specification and analysis of real-time systems using Real-Time Maude." International Conference on Fundamental Approaches to Software Engineering. Springer, Berlin, Heidelberg, 2004.

[2] Meseguer, José, and Grigore Roşu. "Rewriting logic semantics: From language specifications to formal analysis tools." International Joint Conference on Automated Reasoning. Springer, Berlin, Heidelberg, 2004.

[3-23] Boucherit Ammar, « Contribution à la Conception d'Architecture Logicielle Sûre des Systèmes Critiques Basés Agent », thèse de doctorat, 2019.

[4] A. Boucherit, S. Madjouri, K. Madani: XSLT Based Approach for Automatic Generation of Maude Specification for Petri nets. In International Conference on Mathematical Sciences & computer Engineering (ICMSCE 2020).

[5] Scorletti Gérard & Binet G « Réseaux de Petri », Article, université de France, Juin 2006.

[6] Slim BEN SAOUD, « Les Réseaux de Petri », Chapitre 4, page 2/31.

[7] Hamad Hadjer et Naddor Soumia, « Modélisation de processus métiers coopératifs à l'aide de BPMN : vérification et simulation par réseaux de Petri », mémoire pour l'obtention du diplôme de master, université Abdelhamid Ibn Badis Mostaganem, Année universitaire : 2012/2013.

[8] Svadova, Martina, and Zdenek Hanzalek. "PN Matlab Toolbox 2.0." MATLAB 2004-Sborník pĜíspČvkĤ 12. roþníku konference. 2004.

[9] van Hee, Kees, et al. "Yasper: a tool for workflow modeling and analysis." Sixth International Conference on Application of Concurrency to System Design (ACSD'06). IEEE, 2006.

[10] Freytag, Thomas. "Woped–workflow petri net designer." University of Cooperative Education (2005): 279-282.

[11] Gasevic, Dragan, and Vladan Devedzic. "Software support for teaching Petri nets: P3." Proceedings 3rd IEEE International Conference on Advanced Technologies. IEEE, 2003.

[12] Dingle, Nicholas J., William J. Knottenbelt, and Tamas Suto. "PIPE2: a tool for the performance evaluation of generalised stochastic Petri Nets." ACM SIGMETRICS Performance Evaluation Review 36.4 (2009): 34-39.

[13] Drissi et Kaddour Seghir, « Conception d'un outil d'aide à la simulation la vérification et la validation de modèle des réseaux de Petri », Mini projet pour l'obtention du diplôme de licence en informatique, Université de Mostaganem, 2014.

[14] Scorletti Gérard & Binet G « Réseaux de Petri », Article, université de France, Juin 2006.

[15] Slim BEN SAOUD, « Les Réseaux de Petri », Chapitre 4, page 5/31.

[16] D. E. Knuth, P. B. Bendix. Simple word problems in universal algebras , in: Computational Problems in Abstract Algebra, J. Leech (éditeur), Pergamon Press, Oxford, 1970, pp. 263-297.

[17] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency, TCS 96, 1992, pp 73-155.

[18] J. Meseguer. A logical theory of concurrent objects. ACM SIG-PLAN Notices, 25(10), pp 101- 115, 1990. Proc. OOP-SLA ECOOP' 90.

[19] Amina Boudjedir «Utilisation des Systèmes de Réécriture pour la Modélisation et la Vérification des Applications Orientées Aspect». THESE de Doctorat. UNIVERSITÉ BADJI MOKHTAR-ANNABA.2016.

[20] N. Marti-Oliet, J. Meseguer. Rewriting Logic as a Logical and Semantic Framework, Electronic Notes in Théorical Computer Science 4. 1996.

[21] J. Meseguer. A logical theory of concurrent objects. ACM SIG-PLAN Notices, 25(10), pp 101- 115, 1990. Proc. OOP-SLA ECOOP' 90.

[22] José Meseguer, Grigore Roşu « Rewriting Logic Semantics: From Language Specifications to Formal Analysis Tools ».chapter. Page [8-10].May 2004.

[24] Mark-Oliver Stehr, José Meseguer, and Peter Csaba Ölveczky. Rewriting logic as a unifying framework for petri nets.

[25] Noura Boudiaf, Allaoua Chaoui, and Hichem Bakha. A rewriting logic based tool for Ecatnet's analysis: Edition and simulation steps description. Editorial Board, 6(2):16, 2005.

[26] Elhillali Kerkouche and Allaoua Chaou. A graphical tool support to process and simulate ecatnets models based on meta-modelling and graph grammars. INFOCOMP, 8(4):37–44, 2009.

[27] Noura Boudiaf and Abdelhamid Djebbar. Towards an automatic translation of colored petri nets to maude language. International Journal of Computer Science & Engineering, 3(1), 2009.

[28] Ammar Boucherit, Abdallah Khababa, and Laura M Castro. Automatic generating algorithm of rewriting logic specification for multi-agent system models based on petri nets. Multiagent and Grid Systems, 14(4) :403–418, 201.

Annexe

XSLT Based Approach for Automatic Generation of Maude Specification for Petri nets

Ammar Boucherit[1,*], Sara Madjouri[1], and Karima Madani[1]

[1]Computer Science Department, University of El-Oued, Algeria
[*]Corresponding author email: ammar-boucherit@univ-eloued.dz

***Abstract*:** One of the main important tasks of model-driven engineering (MDE) is the automatic generation of model specification. This paper focuses on the presentation of an XSLT (eXtensible Stylesheet Language Transformations) based tool for automatic generation of rewriting logic based Petri net specification. In addition to the well-known advantages such as code reliability improvement as well as the overall development time reduction, this paper presents many other contributions in the field of Petri net based development. The first contribution concerns the specification itself since it is based on an improved rewriting logic based semantics for Petri net. The second contribution is that the Petri net model is based on a universal PNML representation..

***Keywords*:** Petri net, Rewriting Logic, PNML, XSLT

1.Introduction

Petri nets have been widely used as a graphical and semi-formal tool for specification and analysis of concurrent and complex systems. They are gaining a very large popularity in recent years since they allow an easy structural and behavioral description of studied systems [1, 2]. However, traditional methods of informal, manual preparation and ad hoc validation of system specifications have demonstrated their shortcomings. Therefore, looking for a solution that allowed us to quickly, easily and automatically generate formal specifications of Petri nets will be very advantageous.

Rewriting Logic is a very expressive logic that is particularly suitable for formalizing concurrent, complex and real-time systems [3]. In addition, it is a unifying framework for a wide range of Petri nets [4] and many other concurrency models [5]. Nevertheless, specifications based on the rewriting logic of Petri net based systems are often voluminous and/or more difficult to refine or correct. Hence, it is necessary to automate such an operation and save time in the process of writing (preparing) specifications.

Due to their popularity, a large number of tools have been developed for editing, modeling and analyzing Petri net model's properties [6]. In this context, and in order to not limit designers to a specific Petri net tool, the Petri Net Markup Language (PNML) will be used as an input format to the proposed tool.

In this article, we present a prototype implementation of the automatic translation of a Petri description into rewriting logic. The PNML2RWL tool, is written in Delphi and is designed to generate two different specifications based on classical and enhanced rewriting logic semantics for Petri nets.

The paper is organized as follows. In section 2, we give a brief introduction to the PNML description of Petri nets, the classical and improved semantics based on the rewriting logic for Petri nets and the XSLT based transformation. Section 3 presents a brief description of the process of translating the PNML Petri net description into rewriting logic as well as the structure of the tool PNML2Maude and its main modules. In Section 4, we present a simple illustrative example. Finally, Section 5 concludes the paper and draws some perspectives.

2.Preliminaries

2.1. Petri nets and PNML

On one hand, a Petri net is a collection of directed arcs connecting places and transitions. For each transition, the directed arcs describe the set of places which are pre- and / or postconditions. Places may hold zero or more tokens. The Marking M (or State) of a net can be represented by the corresponding number of tokens on their places. The initial values M_0 is called initial marking. A Petri net marking M can be changed when a transitions is fired. When a transition fires, it takes — according to arc weights — a number tokens from the input places; and it then distributes an other number tokens to output places. Figure 1 gives an example of Petri net.

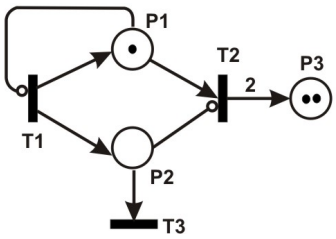


Figure 1. Example of Petri net

On the other hand, Petri nets are a powerful specification language widely used to specify and verify the behaviors of concurrent and complex systems. For that, a large number of Petri net tools have been developed and interoperability between these different tools becomes very essential. Therefore, a Petri net universal interchange format, the PNML was one of these proposals, which focussed on the problem of the different Petri net types to allow Petri net tools to exchange models. Practically, there are many tools such as PN Matlab Toolbox [7], Yasper [8], WoPeD [9], P3 [10], PIPE [11] and ePNK [12] that are able to export Petri net model in PNML format. Indeed, the PNML description contains the

basic structural definition of a Petri net as a labelled directed graph.

2.2. XSLT Transformation

XSLT is a technology that transforms information from an XML document commonly to another type of document such as another document format such as XML, HTML, XHTML, WML, PDF, etc....
The process of transformation is mainly based on an XSLT document called stylesheet that is an XML format file that contains the information needed by the processor to perform the transformation. Practically, an XSLT Stylesheet is associated with an XML-based document in order to create a resulting document of a different or identical format. This principle is illustrated in the Figure 2.

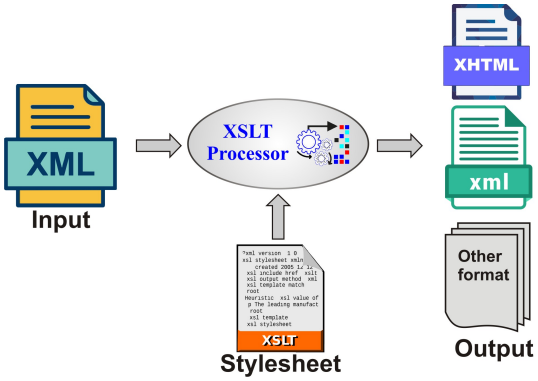


Figure 2. XSLT transformation principle

2.3. Petri nets and Rewriting Logic

Since 1990, rewriting logic [13] has been appeared as a promising logical and semantic framework within which widely different concurrent systems and logics can be naturally specified [14]. Maude system is an implementation of rewriting logic that allows for defining formal executable specifications for many formalisms such as Petri nets. Additionally, Maude offers a powerful verification toolkit, so that Maude's reachability analysis and model-checking which can be used to formally analyse and verify the Petri nets models with respect to different LTL properties [15]. Usually, a Maude specification is composed of a functional module and/or a system module. While functional module describes the static part of a system by an equational theory, the dynamic part is described by a set of rewrite rules and possibly equations in a system module.

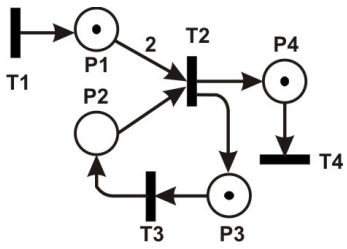


Figure 3. Example 2 of Petri net

According to the existing semantics [4], the Maude specification of Petri net (Figure 3) is given in Listing 1.

```
fmod PETRI-NET-SIGNATURE is
  sorts Place Marking .
  subsorts Place < Marking .
  ops P1 P2 P3 P4 : -> Place .
  op _ _ : Marking Marking -> Marking [ctor assoc comm id: null] .
  op  null : -> Marking .
  op initial : -> Marking .
  eq initial = P1 P4 .
endfm

mod PETRI-NET is
protecting PETRI-NET-SIGNATURE .
var M : Marking .
rl [T1] : M => M P1 .
rl [T2] : P1 P1 P2 => P3 P4 .
rl [T3] : P3 => P2 P4 .
rl [T4] : P3 => P2 .
rl [T5] : M P4 => M .
endm
```

Listing 1. First Maude specification

By the way, we should note that the existing rewriting logic semantics for Petri nets suffers from some limitations as it does not support (i.e. without additional operators) neither the specification of inhibitor arcs nor counting the number of tokens in a specific place. Therefore, we propose an improved semantics for Petri nets that alleviates such limitations. Consequently, the specification of the Petri net given in Figure 1 according to the improved semantics is as follow.

```
fmod PETRI-NET_NEW-SIGNATURE is
protecting INT .
sorts PlaceName Place Marking .
subsort Place < Marking .
ops P1 P2 P3 : -> PlaceName .
op <_,_> : PlaceName Nat -> Place [ctor] .
op _ _ : Marking Marking -> Marking [ctor assoc comm id: null] .
ops initial null : -> Marking .
eq initial = < P1, 1 > < P2, 0 > < P3, 2 > .
endfm
mod NEW-PETRI-NET is
inc PETRI-NET_NEW-SIGNATURE .
var M : Marking .
vars x y : Int .
crl[T1]: < P1, x > < P2, y > => < P1, x + 1 > < P2, y + 1 >
                                  if (x == 0) .
crl[T2]: < P1, x > < P2, y > < P3, z > => < P1, x - 1 > < P2, y >
                                  < P3, z + 2 > if (y == 0) .
crl[T3]: < P2, x > => < P2, x - 1 > if (x == 0) .
endm
```

Listing 2. Second Maude specification

3.The Translation Approach

This section provides an overview about our approach where several steps are involved in the process of translating PNML into the Maude specification. However, many programming details are not further discussed in this paper.

3.1. Overview of the Process

There are five steps to complete the process of translation using PNML2Maude tool:

1. Create a Petri net model in PNML format (using PIPE or P3 for example),
2. Import the PNML file to be translated. The current version of PNML2Maude supports standard Petri nets with or without inhibitor arcs and exported by PIPE or P3 tools,

3. Using XSLT to extract data and values from PNML file and create a purified file (PF),
4. Starting translation by using the purified file and Maude specification templates.
5. Exporting the generated specification into Maude file.

3.2. Tool Description

The tool PNML2Maude is intended to provide a way of automatically transforming a Petri net model, expressed in PNML, as one of the most known format for exchanging Petri nets, into a Maude specification. We have developed a prototype in Delphi with a modular structure to cope with many different translations (see Figure 4). We have also implemented two different translations according to the existing semantics and an improved version. Of course, both take the description from the purified file (generated using XSLT) and generate the corresponding Maude specification.

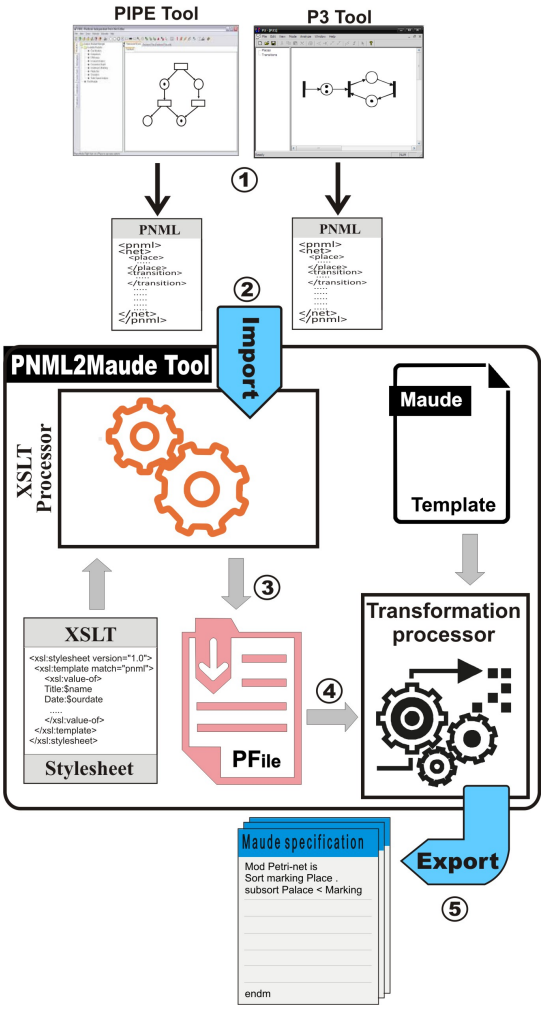


Figure 4. PNML2RWL tool structure

As one can see in Figure 4, our proposed tool is principally composed from two modules:
1) The first module is an XSLT stylesheet that contains a set of rules defining the instructions for performing the transformation of a PNML description into a purified file. Indeed, this XSLT stylesheet is mainly prepared to extract the representative elements of a Petri net from the PNML file. We should note here that we have found some small differences in the syntax of the PNML files exported by the Petri net tools. For that, the XSLT stylesheet developed in the current version of the PNML2Maude tool, is adapted to those of the PNML files of the P3 and PIPE tools. The following Listing 3 shows the corresponding XSLT code.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" encoding="utf-8" />
<xsl:strip-space elements="*"/> <xsl:template match="pnml/net">
<xsl:for-each select="place">  <xsl:value-of select="@id"/>|<xsl:value-
   of select="name"/>|<xsl:value-of select="translate(initialMarking,
   translate(initialMarking,'0123456789',''),'')"/> <xsl:text>|&#10;
   </xsl:text>
</xsl:for-each>
<xsl:for-each select="transition"> <xsl:value-of select="@id"/>|<xsl:
   value-of select="name"/>   <xsl:text>|&#10;</xsl:text> </xsl:for-each>
   <xsl:for-each select="arc[starts-with(@target,'t') or starts with
   (@target,'T')]">
       <xsl:if test="type/@value = 'normal'">inp_<xsl:value-of select=
         "@target"/>:<xsl:value-of select="@source"/>(<xsl:value-of
          select= "translate(inscription, translate(inscription,
          '0123456789', ''), '')"/><xsl:text>)&#10;</xsl:text>
       </xsl:if>
</xsl:for-each>
<xsl:for-each select="arc[starts-with(@target,'t') or starts-with(@
    target,'T')]">
    <xsl:if test="type/@value='inhibitor'">h_inp_<xsl:value-of select=
    "@target"/>:<xsl:value-of select="@source"/>
         <xsl:choose>
             <xsl:when test="inscription =''">
                <xsl:text>(1)</xsl:text><xsl:text>&#10;</xsl:text>
             </xsl:when>
             <xsl:otherwise>
               <xsl:text>(</xsl:text><xsl:value-of select="translate
              (inscription,translate(inscription,'0123456789',''),'')"/>
              <xsl:text>)&#10;</xsl:text>
             </xsl:otherwise>
         </xsl:choose>
   </xsl:if>
</xsl:for-each>
<xsl:for-each select="arc[starts-with(@source,'t') or starts with
    (@source,'T')]"> out_<xsl:value-of select="@source"/>:<xsl:value-of
    select="@target"/>(<xsl:value-of select="translate(inscription,
    translate(inscription, '0123456789', ''), '')"/> <xsl:text>)&#10;
    </xsl:text>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Listing 3. XSLT Code for preparing the purified file

2) The second module is composed of a set of algorithms which are collected to build three matrices: pre-incidence, post-incidence and inhibitor (inh) incidence matrices. In fact, these algorithms are the improvement of the one given in [16] which are limited for Pure Petri nets without inhibitor arcs. For that, we add the inh-incidence matrix to automatically generate the corresponding Maude specifications for a standard Petri even with an inhibitor arcs.

4.Illustrative Example (implementation results)

In this section, we use a simple example of Traffic Light (2 ways) to show the usefulness of the developed prototype for transforming PNML description to Maude specification. In this example, the traffic light has three lights: "Red", "Yellow" and "Green" (one place is used per each color). An additional place "Safe" is used to guarantee a safe alternating between the two ways (only one green light is turned on). The traffic light transitions from one light to another (red to yellow to green to yellow to red again).
Figure 5 shows the Petri net describing such traffic system by using PIPE, and Table 1 presents the meaning of the Petri net places and transitions.

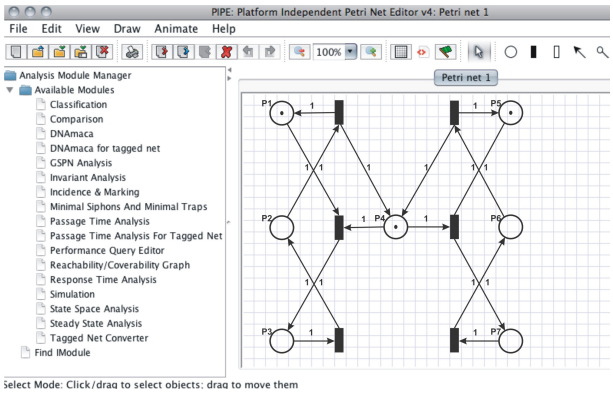


Figure 5. Petri net model of the traffic light control system

Table 1. Meaning of Petri net places and transitions

Place labels with meanings		Transition labels with meanings	
P1	Red_1 State	**T1**	Yellow_1 to Red_1
P2	Yellow_1 state	**T2**	Red_1 to Green_1
P3	Green_1 State	**T3**	Green_1 to Yellow_1
P4	Safe (Alternate)	**T4**	Yellow_2 to Red_2
P5	Red_2 State	**T5**	Red_2 to Green_2
P6	Yellow_2 State	**T6**	Green_2 to Yellow_2
P7	Green_2 State		

The corresponding PNML description is given in the following listing 4.

```
<?xml version="1.0" encoding="ISO-8859-1"?><pnml>
<net id="Net-One" type="P/T net">
<place id="P1"> <name> <value>P1</value> </name>
<initialMarking> <value>Default,1</value> </initialMarking> </place>
<place id="P2"> <name> <value>P2</value> </name>
<initialMarking> <value>Default,0</value> </initialMarking> </place>
<place id="P3"> <name> <value>P3</value> </name>
<initialMarking> <value>Default,0</value> </initialMarking> </place>
<place id="P4"> <name> <value>P4</value> </name>
<initialMarking> <value>Default,1</value> </initialMarking> </place>
<place id="P5"> <name> <value>P5</value> </name>
<initialMarking> <value>Default,1</value> </initialMarking> </place>
<place id="P6"> <name> <value>P6</value> </name>
<initialMarking> <value>Default,0</value> </initialMarking> </place>
<place id="P7"> <name> <value>P7</value> </name>
<initialMarking> <value>Default,0</value> </initialMarking> </place>
<transition id="T1"> <name> <value>T1</value> </name> </transition>
<transition id="T2"> <name> <value>T2</value> </name> </transition>
<transition id="T3"> <name> <value>T3</value> </name> </transition>
<transition id="T4"> <name> <value>T4</value> </name> </transition>
<transition id="T5"> <name> <value>T5</value> </name> </transition>
<transition id="T6"> <name> <value>T6</value> </name> </transition>
<arc id="P1 to T2" source="P1" target="T2"> <inscription> <value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="P2 to T1" source="P2" target="T1"> <inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="P3 to T3" source="P3" target="T3"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="P4 to T2" source="P4" target="T2"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="P4 to T5" source="P4" target="T5"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="P5 to T5" source="P5" target="T5"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="P6 to T4" source="P6" target="T4"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="P7 to T6" source="P7" target="T6"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T1 to P1" source="T1" target="P1"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T1 to P4" source="T1" target="P4"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T2 to P3" source="T2" target="P3"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T3 to P2" source="T3" target="P2"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T4 to P4" source="T4" target="P4"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T4 to P5" source="T4" target="P5"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T5 to P7" source="T5" target="P7"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
<arc id="T6 to P6" source="T6" target="P6"><inscription><value>
Default,1</value> </inscription> <type value="normal"/> </arc>
</net>
</pnml>
```

Listing 4. PNML description of Petri net (Figure 5)

After executing the prototype and importing such PNML description, we have obtained the following purified file.

```
P1|P1|1|
P2|P2|0|
P3|P3|0|
P4|P4|1|
P5|P5|1|
P6|P6|0|
P7|P7|0|
```

List of Places

```
P1|P1|1|
```

Initial Marking

Place Label

Place ID

```
T1|T1|
T2|T2|
T3|T3|
T4|T4|
T5|T5|
T6|T6|
```

List of Transitions

```
T1|T1|
```

Transition Label

Transition ID

```
inp_T2:P1(1)
inp_T1:P2(1)
inp_T3:P3(1)
inp_T2:P4(1)
inp_T5:P4(1)
inp_T5:P5(1)
inp_T4:P6(1)
inp_T6:P7(1)
```

List of Input Arcs

```
inp_T2:P1(1)
```

Input Arc to **T2** from **P1** with weight = **1**

```
out_T1:P1(1)
out_T1:P4(1)
out_T2:P3(1)
out_T3:P2(1)
out_T4:P4(1)
out_T4:P5(1)
out_T5:P7(1)
out_T6:P6(1)
```

List of Ouput Arcs

```
out_T1:P1(1)
```

Ouput Arc from **T1** to **P1** with weight = **1**

Listing 5. Purified file with comments

Thereafter, the first result specification is given in Listing 6.

```
fmod FUNC_PETRINET is
sorts Place Marking .
subsorts Place < Marking .
op empty : -> Marking .
ops P1 P2 P3 P4 P5 P6 P7 : -> Place [ctor] .
op _ _ : Marking Marking -> Marking [ctor assoc comm id: empty] .
endfm
mod SYSTEM_PETRINET is
including FUNC_PETRINET .
rl[T1]: P2 => P1 P4 .
rl[T2]: P1 P4 => P3 .
rl[T3]: P3 => P2 .
rl[T4]: P6 => P4 P5 .
rl[T5]: P4 P5 => P7 .
rl[T6]: P7 => P6 .
endm
```

Listing 6. First result specification

Then, the second result specification is as follow.

```
fmod NEW_FUNC_PETRINET is
protecting INT .
sorts PlaceName Place Marking .
subsort Place < Marking .
ops P1 P2 P3 P4 P5 P6 P7 : -> PlaceName .
op <_,_> : PlaceName Nat -> Place [ctor] .
op _ _ : Marking Marking -> Marking [ctor assoc comm id: null] .
op null : -> Marking .
endfm

mod NEW-SYSTEM_PETRI-NET is
inc NEW_FUNC_PETRINET .
var M : Marking .
vars x y z : Int .
crl[T1]: < P1, x > < P2, y > < P4, z > => < P1, x + 1 > < P2, y - 1 >
                                        < P4, z + 1 > if (y >= 1) .
crl[T2]: < P1, x > < P3, y > < P4, z > => < P1, x - 1 > < P3, y + 1 >
                                        < P4, z - 1 > if (x >= 1) and (z >= 1) .
crl[T3]: < P2, x > < P3, y > => < P2, x + 1 > < P3, y - 1 > if (y >= 1).
crl[T4]: < P4, x > < P5, y > < P6, z > => < P4, x + 1 > < P5, y + 1 >
                                        < P6, z - 1 > if (z >= 1) .
crl[T5]: < P4, x > < P5, y > < P7, z > => < P4, x - 1 > < P5, y - 1 >
                                        < P7, z + 1 > if (x >= 1) and (y >= 1) .
crl[T6]: < P6, x > < P7, y > => < P6, x + 1 > < P7, y - 1 > if (y >= 1).
endm
```

Listing 7. Second result specification

5.Conclusions

Petri net models are often very large and the more the complexity of the studied system increases, the more difficult it is to prepare its Maude specification because of Maude system sensitivity. In this paper, we have proposed an XSLT based methodology for translating PNML based Petri net description into Maude specification. Our aim is to offer to Petri nets practitioners and designers a tool to easily and automatically preparing Maude specification for large Petri net models even if they don't have too much experience with rewriting logic.
The main contribution of the present work is the use of PNML as input for the developed tool PNML2Maude. Consequently, PNML2Maude will not be limited for those that familiar with a specific Petri net tool.
Finally, we aim to continue enhancing our PNML2Maude in two axis :

- Create an application with a more convenient graphical user interface (GUI) allows to edit Petri nets and imports PNML files from more Petri net based tools.

- Develop a web-based application to provide more portability for our tool and it will be available soon at El-Oued University website.

References

1. A. Boucherit, L. M. Castro, A. Khababa, and O. Hasan, "Petri net and rewriting logic based formal analysis of multi-agent based safety-critical systems," *Multiagent and Grid Systems*, vol. 16, no. 1, pp. 47–66, 2020.
2. J. Dong, J. Jiao, H. Xia, and J. Chu, "Safety simulation and analysis for complex systems concurrency based on petri net and stateflow model," in *2019 Annual Reliability and Maintainability Symposium (RAMS)*, pp. 1–7, IEEE, 2019.
3. M. Alpuente, D. Ballis, F. Frechina, and J. Sapiña, "Exploring conditional rewriting logic computations," *Journal of Symbolic Computation*, vol. 69, pp. 3–39, 2015.
4. M.-O. Stehr, J. Meseguer, and P. C. Ölveczky, "Rewriting logic as a unifying framework for petri nets," in *Unifying Petri Nets*, pp. 250–303, Springer, 2001.
5. J. Meseguer, "Conditional rewriting logic as a unified model of concurrency," *Theoretical computer science*, vol. 96, no. 1, pp. 73–155, 1992.
6. "Petri net tools." http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools. Accessed: 2020-06-22.
7. M. Svadova and Z. Hanzalek, "Pn matlab toolbox 2.0," Citeseer.
8. K. van Hee, O. Oanea, R. Post, L. Somers, and J. M. van der Werf, "Yasper: a tool for workflow modeling and analysis," in *Sixth International Conference on Application of Concurrency to System Design (ACSD'06)*, pp. 279–282, IEEE, 2006.
9. T. Freytag, "Woped–workflow petri net designer," *University of Cooperative Education*, pp. 279–282, 2005.
10. D. Gasevic and V. Devedzic, "Software support for teaching petri nets: P3," in *Proceedings 3rd IEEE International Conference on Advanced Technologies*, pp. 300–301, IEEE, 2003.
11. P. Bonet, C. M. Lladó, R. Puijaner, and W. J. Knottenbelt, "Pipe v2. 5: A petri net tool for performance modelling," in *Proc. 23rd Latin American Conference on Informatics (CLEI 2007)*, 2007.
12. "The epnk platform." http://www.imm.dtu.dk/~ekki/projects/ePNK/index.shtml. Accessed: 2020-06-21.
13. J. Meseguer, "Rewriting as a unified model of concurrency," in *International Conference on Concurrency Theory*, pp. 384–400, Springer, 1990.
14. J. Meseguer, "Twenty years of rewriting logic," *The Journal of Logic and Algebraic Programming*, vol. 81, no. 7-8, pp. 721–781, 2012.
15. M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, and C. Talcott, "Maude manual (version 3.0)," *SRI International–University of Illinois at Urbana-Champaign. URL: http://maude. cs. uiuc. edu*, 2019.
16. A. Boucherit, A. Khababa, and L. M. Castro, "Automatic generating algorithm of rewriting logic specification for multi-agent system models based on petri nets," *Multiagent and Grid Systems*, vol. 14, no. 4, pp. 403–418, 2018.