N° 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
Département D'Informatique

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**

Présenté par :

- **Kenaouia Manar**
- **Farah Roukaya**

Thème

Proposition d'une sémantique orientée objet pour les réseaux de Petri sous Maude

Soutenue le15/06/2022 Devant le jury composé de :

Mr. Abbas Messaoud	Président
Mr. Kerthiou Smaïl	Examinateur
Mr. Boucherit Ammar	Rapporteur

Année Universitaire: 2021/2022

Table Des Matières

Chapitre 3 : Proposition d'une sémantique OO pour les réseaux de Petri sous Maude

Chapitre 4: Conception et implémentation de l'outil de transformation automatique

Remerciements

في البداية نشكر الله على إتمام عملنا هذا

الحمد لله على الأوقات التي قلّت عزيمتنا فيها، فأرسل إلينا رسائل من عنده شدت عزيمتنا، وقويت بها همتنا، فألهمنا التوفيق.

نشكر مشرفنا الأستاذ: بوشريط عمار على مساعدته وتوجيهه لنا. ونشكر كل من ساعدناعلى إتمام عملنا هذا، كما نشكر لجنة المناقشة على الموافقة على تقييمهذا العمل.

وأخيرا نشكر الأهل والأصدقاء وكل من دعمنا.

Dédicace

Je dédie ce travail de recherche à ceux qui m'ont enseignéà l'école et dans la vie, mes parents: le professeur ***Kenaouia Ali*** *et le professeur* ***Ben Maussa ourida.***

Et pour ceux qui m'ont élevé quand j'étais jeune, mon grand-père ***Ben Maussa Youssef*** *(Rahimahou Allah) , et ma grand-mère (Hafidha Allah) .*

Et à ma sœur et compagne ***Rania*** *, et à mes frères* ***Med Midani, Rayane, Malak et Al-Hadje Al-Hachemi.***

A mon amie ***Aïcha*** *.*

à ma famille .

A tous ceux qui m'ont soutenu et ont cru en mes capacités.

A mes amis qui m'ont support.

A tous ceux qui m'ont souhaité bonne chance.

Je vous souhaite bonheur et contentement.

K.Manar

Dédicace

Je te dédie mon diplôme, ma famille, mes proches, mes amis et ma consolation

Hafidakom Aallah tous, Que Dieu vous bénisse fier et fier de Partout et à tout moment.

F. Roukaya

Résumé

Les réseaux de Petri sont un outil révolutionnaire pour modéliser, simuler et analyser les comportements ainsi que les interactions des systèmes dynamiques et concurrents. Cependant, la sémantique basée sur la logique de réécriture existante des réseaux de Petri standard ne reflète pas la séquence réelle d'évolution de l'état des réseaux de Petri, car elle n'est pas en mesure de montrer naturellement quelles sont les transitions activées dans chaque nouvel état du système. De plus, il ne prend pas en charge le calcul implicite de nombre de jetons dans les places, ce qui complique la description de l'état du réseau de Petri. De telles lacunes peuvent limiter l'utilisation de la sémantique existante à des fins de simulation et son utilisation peut conduire à une évolution et/ou à une description d'état incompréhensible du modèle de réseau de Petri.

Dans le présent travail, nous proposons une sémantique orientée objet basée sur la logique de réécriture pour les réseaux de Petri qui surmonte ces limitations et fournit une description améliorée des règles d'activation et de franchissement afin de mieux décrire et/ou de simuler l'état et l'évolution des systèmes dynamiques.

ملخص

شبكات بتري هي أداة رائدة لنمذجة ومحاكاة وتحليل سلوكيات وتفاعلات الأنظمة الديناميكية والمتزامنة. ومع ذلك، فإن الدلالات القائمة على منطق إعادة الكتابة الحالي لشبكات بتري القياسية لا تعكس تسلسل تطور الحالة الفعلي لشبكات بتري، حيث إنها غير قادرة على إظهار التحولات التي تم تمكينها بشكل طبيعي في كل حالة جديدة من النظام. علاوة على ذلك، لا يدعم الحساب الضمني لعدد العينات(jeton) في الأماكن(places)، مما يعقّد وصف حالة شبكة بتري. كما يمكن أن تحد أوجه القصور هذه من استخدام الدلالات الحالية لأغراض المحاكاة وقد يؤدي استخدامها إلى تطور أو وصف حالة غير مفهوم لنموذج شبكة بتري.

في هذه المذكرة، نقترح دلالات موجهة للكائنات استنادًا إلى منطق إعادة الكتابة لشبكات Petri التي تتغلب على هذه النقائص وتوفر وصفًا محسنًا لقواعد التنشيط وقواعد التشغيل لوصف أو محاكاة حالة أو تطور الأنظمة الديناميكية بدقة.

Abstract

Petri-nets are a ground-breaking tool for modeling, simulating and analyzing behaviors as well as interactions of dynamic and concurrent systems. However, the existing rewriting logic based semantics of standard Petri nets does not reflect the real sequence of Petri net state evolution, since it is not able to naturally show which are the enabled transitions in each new state of the system. Furthermore, it does not support an implicit count of the number of tokens in places which complicate the Petri net state description. Such deficiencies may limit the use of existing semantic in simulation purpose and its use may lead to incomprehensible evolution and/or state description of the Petri net model.

In the present work, we propose a rewriting logic based object oriented semantics for Petri nets that overcomes such limitations and provides an enhanced description of both enabling and firing rules in order to exactly describe and/or simulate the dynamic system state or evolution.

Introduction Générale

Les réseaux de Petri sont toujours connus comme l'un des formalismes les plus efficaces en raison de certaines de leurs caractéristiques distinctives, telles que la notation graphique intuitive, une base mathématique bien définie et une sémantique simple. Les réseaux de Petri [1] sont un formalisme prometteur pour modéliser [2, 3, 4], analyser [5, 6, 7] et simuler [8,9,10] un large éventail de systèmes distribués et de réseaux biologiques. Cependant, malgré tous les avantages qu'ils présentent, les réseaux de Petri souffrent de certaines lacunes, à savoir le besoin croissant de techniques de vérification automatique et d'outils pour analyser rigoureusement le comportement des modèles complexes basés sur les réseaux de Petri [11,12,13,14].

En même temps, les approches orientées objet sont devenues extrêmement populaires dans les environnements de développement de logiciels en raison de leur haut niveau d'abstraction dans la conception, de leurs puissantes fonctionnalités de structuration de la hiérarchie du système et de la promotion de la réutilisation du code. Par conséquent, l'orientation objet est l'une des tendances les plus importantes qui a été largement discutée pour l'intégration avec le formalisme des réseaux de Petri [15,16] puis exploitée précédemment et dans le temps actuel [17, 5, 18].

D'autre part, Maude [19] est un langage de programmation exécutable et un cadre bien adapté pour spécifier, exécuter et analyser les réseaux de Petri, grâce à l'expressivité de la logique de réécriture [20] et de ses outils d'analyse associés tels que l'outil d'analyse d'accessibilité et le vérificateur de modèle LTL. Dans ce contexte, les auteurs de [20,21] ont proposé une sémantique basée sur la logique de réécriture pour les réseaux de Petri qui a été adoptée par la plupart des travaux existants, par exemple [22, 23, 24, 25].

L'objectif principal de ce travail est de réaliser une sémantique orientée objet basée sur la logique de réécriture pour les réseaux de Petri qui sera mieux adaptée pour la simulation et la vérification formelle des réseaux de Petri, en bénéficiant des outils d'analyse associés au système Maude. Plus précisément, la sémantique proposée sera également très avantageuse pour de nombreux autres avantages qui ne sont pas suffisamment considérés dans ce travail. Par exemple, la possibilité d'afficher l'état

d'activation d'une transition (c'est-à-dire si une transition est activée ou non), prenant en charge le calcul implicite du nombre de jetons dans les places, et ainsi, la capacité de décrire les réseaux de Petri avec des arcs inhibiteurs …etc.

Ce mémoire de fin d'étude est organisé comme suit : dans le chapitre 1, nous présentons quelques notions de bases les réseaux de Petri. Le chapitre 2 présente les concepts les plus importants sur le système Maude et la logique de réécriture. Dans le chapitre 3, nous présentons la sémantique proposée pour les réseaux de Petri. Enfin, dans le chapitre 4 nous présentons la conception et l'implémentation de l'outils de génération automatique de spécification Maude suivant la sémantique proposée.

Chapitre 1

Les Réseaux de Petri

Introduction

Les réseaux de Petri représentent un outil graphique mathématique pour la modélisation et la vérification du comportement dynamique des systèmes à événements discrets. Ils sont une forme simple de système qui nous aide à comprendre le système.

Grâce aux modèles de graphes et aux équations algébriques, nous pouvons analyser un système qui simule des activités dynamiques et simultanées. Les réseaux de Petri ont été utilisés au cours des dernières décennies pour la simulation, la modélisation et la vérification dans de nombreux travaux de recherches académiques et industriels.

Dans ce chapitre, nous étudierons les principaux concepts et propriétés des réseaux de Petri et leur représentation.

1 Réseaux de Petri :

1.1 Définition :

Le réseau de Petri (RdP) est une structure de graphe mathématique qui facilite la modélisation de systèmes concurrents complexes [26], simule un large éventail d'applications et permet également une bonne analyse du système.

Le réseau de Petri est constitué d'un ensemble de place et de transitions reliés par des arcs vectoriels. Les arcs sont le lien entre les lieux et les transitions, et ces transitions s'effectuent selon certains critères, où le réseau est représenté par le quadruplet $R = < P, T, Pre, Post >$ où :

- P représente l'ensemble fini de **places**, $\boldsymbol{P_i} = \{P_1, P_2, \dots\}$.
- T représente l'ensemble fini de **transitions,** $\boldsymbol{T_j} = \{T_1, T_2, \dots\}$.
- $Pre : P \times T \rightarrow N$. tel que $Pre(p,t)$ = nombre de jeton nécessaire dans la place p (en amont) pour l'activation puis le franchissement de la transition t. l'ensembles des $Pre(p,t)$ constitue la matrice de pré-incidence.
- $Post : P \times T \rightarrow N$. tel que $Post(p,t)$ = nombre de jeton produits dans la place p (en aval) lors du franchissement de la transition t. l'ensembles des $Post(p,t)$ constitue la matrice de pré-incidence [27].

Il est noté que les places et transitions sont reliées par des arcs orientés. C'est pourquoi il est 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 omis s'il est égal à 1.

1.2 Représentation d'un réseau de Petri :

1.2.1 Représentation graphique :

Un réseau de Petri est un graphe biparti qui comporte deux types de sommets, des places et des transitions reliées alternativement par des arcs. Ce graphe est représenté par les symboles suivants :

- Représente la place qui est un cercle
- Il représente la transition, qui est un rectangle
- Représente les arcs qui n'associent pas des valeurs de la même famille
- les jetons sont représentés par des points noirs

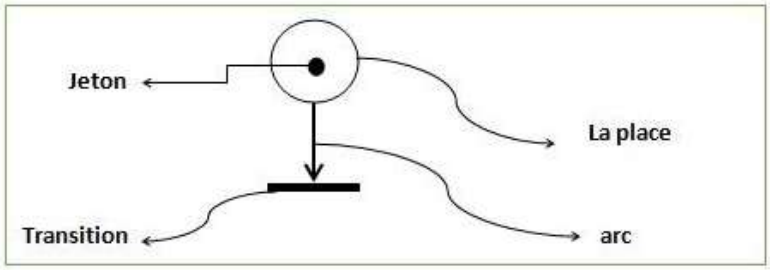


Figure 1: Représentation graphique d'un RdP.

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

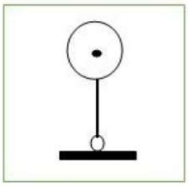
Condition : modélisée à l'aide d'une place, est un prédicat logique d'un état du système, Elle est soit vraie, soit fausse. La satisfaction d'une condition est modélisée à l'aide d'un jeton.

Condition vraie (présence du nombre suffisant de jetons)

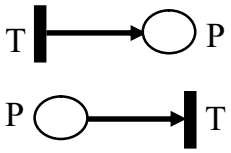
Condition fausse (non présence du nombre suffisant de jetons)

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.

Remarques :



1. Les arcs peuvent avoir un statut particulier (arc inhibiteur). Dans ce cas, le jeton est utilisé seulement pour le test et ne sera pas consommé lors du franchissement d'une transition. Cet arc porte un cercle au point de sa connexion avec la transition.
2. Une transition peut avoir deux cas particuliers :
   - Transition source **:** elle n'a pas de places (amont) en entrée.
   - Transition puits **:** elle n'a de places (aval) en sortie.



1.2.2 Représentation matricielle :

Pratiquement, Il est aussi possible de représenter un RdP par une ou des matrices d'incidences d'ordre : N × M, où N (resp, M) représente le nombre de places (resp, transitions).

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

Remarques :

1. Les matrices ***Pre(P,T)*** et ***Post(T,P)*** peuvent être regroupées dans une même matrice C qui s'appelle la matrice d'incidence (matrice d'effet) comme suit : ***C = Post(T,P) - Pré(P,T)***.

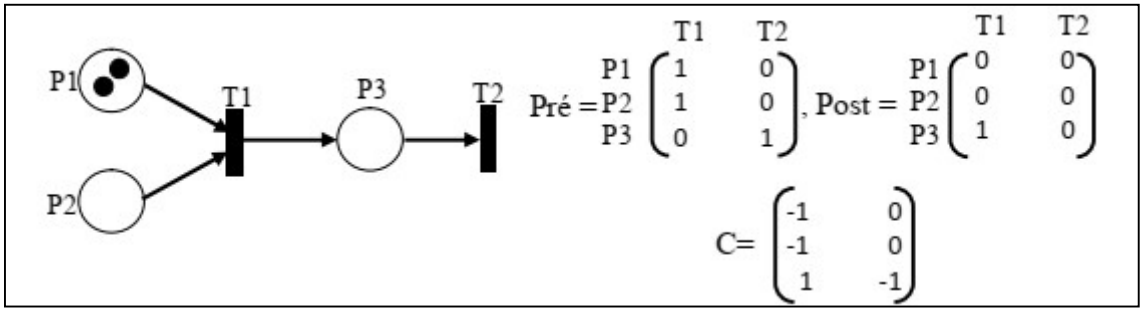


Figure 2. Exemple de réseau de Petri

2. La matrice Pré(P,T): peut représenter les arcs inhibiteurs avec des valeurs négatives.

1.2.3 Représentation PNML:

Il existe une autre représentation basée sur le langage PNML, qui est un format de représentation des modèles de réseaux de Petri basé sur le langage 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: Example de PNML [1]

En effet, une description PNML contient la définition structurelle de base d'un réseau de Petri sous la forme d'un graphe orienté étiqueté (Net, Place, Transition, arc). Le langage PNML offre un format d'échange universel de réseau de Petri pour permettre aux outils de réseaux de Petri tels que PIPE, P3 et WoPeD d'échanger les modèles développés.

2 Dynamique et évolution d'un RDP

2.1 Etat du système (Marquage) :

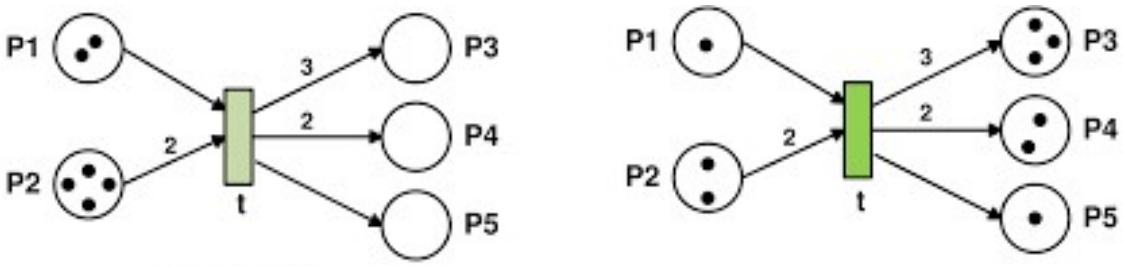
Généralement, l'état d'un réseau de Petri est représenté par le nombre de jetons (positif ou nul) contenus dans la place P_i. Ce nombre peut être négatif pour indiquer un arc inhibiteur. Cette description du réseau de Petri est souvent appelé marquage. Le marquage est utilisé pour décrire l'état du système avec une plus grande précision à un certain moment. Mathématiquement, le

marquage M est un vecteur colonne de dimension égal au nombre de places du RdP. Le marquage dit initial décrit l'état initial du système modélisé (M_0). Le marquage du RdP de la Figure 2 est donné comme suit : M (3, 1, 0) ou bien M (p_1)= 2 , M(p_2)=0, M(p_3)=0,

2.2 Evolution du système (franchissement) :

Un réseau de Petri s'évolue lorsqu'on exécute une transition : des jetons sont retirés dans les places en entrée de cette transition et des jetons sont déposés dans les places en sortie de cette 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 :



(b) t est active avant franchissement (a) t est active après franchissement

Figure 4: Exemple d'activation et franchisement.

Après le franchissement d'une transition, des jetons sont retirés de ses places en amont et des jetons sont déposés dans ses places en aval. Le franchissement d'une transition permet donc d'atteindre un nouveau marquage par exemple M_1 à partir de M_0 de la Figure 4 comme suit :

$$M_0(2,4,0,0,0) \rightarrow M_1(1,2,3,2,1).$$

Remarque :

- L'activation d'une transition n'implique pas qu'elle sera fortement franchie ; mais, cela ne représente qu'une possibilité de franchissement ou d'évolution de l'état du RdP.
- Par convention, un RdP ne peut évoluer que par franchissement d'une seule transition active à cet instant pour éviter le problème de conflit.
- La transition source est toujours active et franchissable.

2.3 Séquence de franchissement :

Le franchissement de plusieurs transitions à partir d'un marquage initial M_0, de T_1 puis T_2 ou bien T_3 (voir la Figure 5) conduit au marquage $M_1 ou\ M_2$. On appellera T_1T_2 ou T_1T_3 séquences de franchissements et on notera par exemple:

$$M_0, \xrightarrow{T1\ T2} M_1 \text{ , ou encore } M_0, \xrightarrow{S} M_1 \text{ avec } S = T_1T_2 \text{ . ou bien,}$$

$$M_0, \xrightarrow{T1\ T2} M_2 \text{ , ou encore } M_0, \xrightarrow{S} M_2 \text{ avec } S = T_1T_3 \text{ .}$$

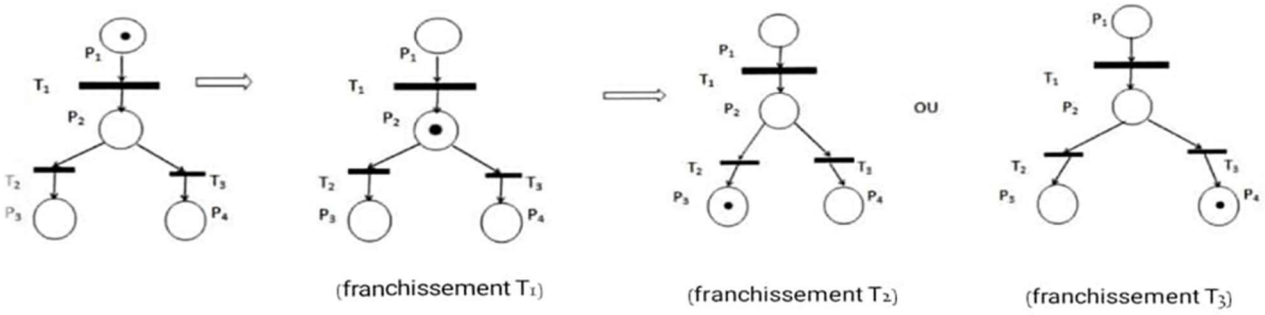


Figure 5 : Exemple pour une séquence franchissement .

2.4 Quelques propriétés des réseaux de Petri :

2.4.1 Conflit et parallélisme :

- Conflit structurel : se provoque lorsque plusieurs transitions (au moins 2 transitions) ont au moins une place commune en entrée[28].
- Conflit effectif dans un RdP sauf : se provoque lorsque le nombre de jetons dans la place commune est inférieure à la somme des poids d'arcs sortant de cette place.
- Parallélisme : Elle est franchissement équilibrée entre deux places

2.4.2 Blocage :

Un RdP est en état de blocage s'il n'est pas possible de franchir aucune de ses transitions. C-à-d, aucune condition d'activation de toutes les transitions n'est remplie.

2.4.3 Vivacité :

- **Transition vivante** : Une transition Tj est vivante pour un marquage initial Mo, si pour tout marquage accessible, il existe toujours une possibilité de franchissement pour Tj.
- **RdP vivante** : un RdP est dit vivant pour un marquage initial Mo si toutes ses transitions sont vivantes pour ce marquage Mo.

2.4.4 Bornitude :

On dit qu'un RdP est borné s'il existe un nombre naturel k où tous les marquages du RdP atteints à partir de m0 sont inférieurs ou égaux à k [29].

3 Exemple de Modélisation par réseau de Petri

Un des problèmes classiques pour la modélisation avec les réseaux de Petri est le problème de signalisation dans un carrefour ou bien "feu rouge". Dans ce problème, les transitions représentent les changements de couleur, et les places permettent de procéder à ces changements dans le bon ordre (rouge ne suit jamais vert par exemple). La figure suivante montre cette modélisation.

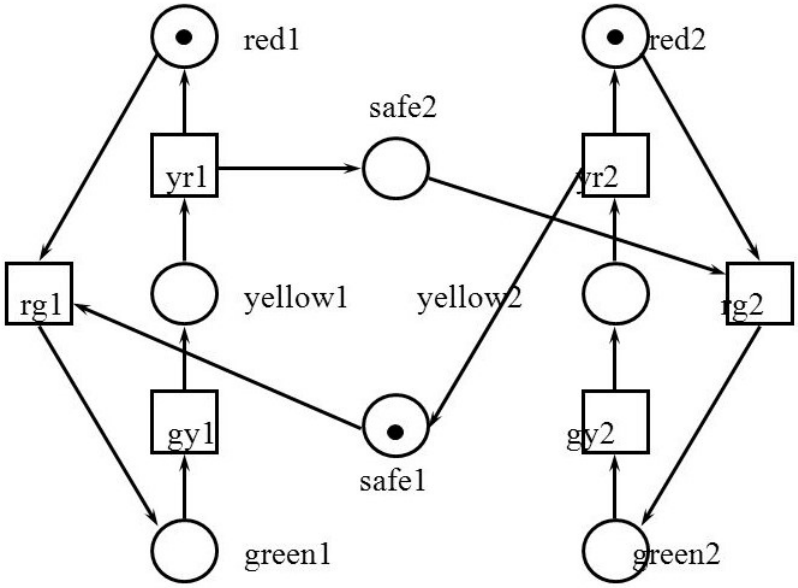


Figure 6. Modélisation de Feu Rouge par réseaux de Petri

red1 : le 1er feu rouge en état rouge

yr1 : le 1er feu rouge passe du jaune au rouge

yellow1 : le 1er feu rouge en état jaune

gy1 : le 1er feu rouge passe du vert au jaune

green1 : le 1er feu rouge en état vert

rg1 : le 1er feu rouge passe du rouge au vert

safe1 et safe2: deux places régulatrices assurant l'alternativité entre le 1er feu rouge et le 2ème feu rouge. C-à-d, aucun des feux rouges reste toujours rouge. Les autres places et transitions (red2, yellow2, green2, yr2, gy2 et rg2) du 2ème feu rouge sont similaires autres places et transitions du 1er feu rouge.

4 Quelques domaines d'application :

Les réseaux de Petri ont été largement utilisés pour la modélisation dans des différents domaines, dont :

- L'industrie : la modélisation des chaînes de production (de fabrication).
- L'informatique : la modélisation des systèmes informatiques, des protocoles de communication et la programmation concurrente.
- Diagnostic (Intelligence artificielle) : pour trouver l'erreur d'origine dans la ligne d'erreur.
- La chimie et biochimie : Les réactions chimiques peuvent être modélisées par les réseaux de Petri.

5 Conclusion

Les réseaux de Petri représentent un outil graphique de modélisation de système complexe. Sa puissance d'expression permet d'étudier des systèmes composés de sous-systèmes fonctionnant en parallèle, communiquant et partageant des ressources. Dans ce chapitre, nous avons abordé les concepts de base du réseau de Petri qui seront utile pour la compréhension de la sémantique qui sera proposée dans le chapitre 3.

Chapitre 2

Le système Maude

Introduction

La logique de réécriture est une logique de calcul simple qui peut naturellement exprimer à la fois le calcul simultané et la déduction logique avec une grande généralité. En d'autres termes, il s'agit d'une logique unifiée et réflexive basée à la fois sur la théorie de la réécriture et sur la logique équationnelle.

Dans ce chapitre, nous étudierons les règles les plus importantes de la logique de réécriture et son système Maude, qui est un système puissant et compétitif, pour apprendre les bases de l'utilisation de la logique de réécriture.

6 Logique de réécriture

La logique de réécriture est un cadre sémantique général, dans lequel non seulement des applications, mais des formalismes entiers et d'autres langages peuvent s'exprimer naturellement. Cette logique a été introduite par José Meseguer [30], suite à ses travaux sur les logiques générales pour décrire les systèmes concurrents. La logique de réécriture permet de raisonner correctement sur des comportements de systèmes concurrents, ayant des états et évoluant en termes de transitions.

7 La théorie de la réécriture

La théorie de la réécriture $\mathfrak{R} = (\Sigma, E, L, R)$ est la structure la plus puissante dans la logique de réécriture qui sert à décrire un système concurrent. Où, la structure statique du système est décrite par une signature (Σ, E). C-à-d, une théorie équationnelle définissant la structure algébrique particulière des états du système (multi-ensemble, arbre binaire…) qui sont distribués selon cette même structure. D'autre part, la structure dynamique du système est décrite par les règles de réécriture étiquetées R. Pratiquement, les règles de réécriture précisent quelles sont les transitions (élémentaires et locales) possibles dans l'état actuel du système concurrent et formalise leurs changements.

Chaque règle (notée $[t] \rightarrow [t']$) correspond à une action pouvant survenir en concurrence avec d'autres actions. C'est pourquoi on dit que la logique de réécriture est une logique qui capture clairement le changement concurrent dans un système [31].

2.1 Théorie de réécriture étiquetée

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

- Σ est un ensemble de symboles de fonctions, et de sortes.
- E un ensemble de Σ-équations (l'ensemble des équations entre les Σ-termes).
- L est un ensemble d'étiquettes.
- 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 [30].

2.2 Les systèmes de réécriture conditionnels

Les systèmes de réécriture peuvent être naturellement étendus à des systèmes de réécriture conditionnels par l'ajout des conditions sur l'application des règles de réécriture.

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 :

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

Telle que une règle $\boldsymbol{R}$ exprime que la classe d'équivalence contenant le terme $\boldsymbol{t}$ peut se réécrire en la classe d'équivalence contenant le terme $\boldsymbol{t'}$ 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 $\boldsymbol{C}$, 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 $\boldsymbol{C}$, et nous écrivons :

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

3 Règles de déduction :

Le calcul dans un système concurrent est une séquence de transitions (règles de réécriture) exécutées à partir d'un état initial donné [32]. Il correspond à une preuve ou une déduction dans la logique de réécriture. Cette déduction est intrinsèquement concurrente et permet de raisonner correctement sur l'évolution du système d'un état à un autre.

- **Principe de déduction :**

Etant donné une théorie de réécriture R= (Σ, 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:

1. La réflexivité :

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

2. La congruence :

Pour chaque fonction $f \in \sum_n$, $n \in N$.

$$\frac{[t_1] \rightarrow [t'_1] \quad \ldots\ldots \quad [t_n] \rightarrow [t'_n]}{[f(t_1 \ldots\ldots t_n)] \rightarrow [f(t'1 \ldots\ldots t'n)]}$$

***3. Le remplacement** :*

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

$$\frac{[w_1] \rightarrow [w'_1] \quad \ldots\ldots \quad [w_n] \rightarrow [w'_n]}{[t(\overline{w} / \overline{x})] \rightarrow [t'(\overline{w'} / \overline{x})]}$$

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

4. La transitivité :

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

4 Langages basés sur la logique de réécriture :

4.1 Les systèmes Maude

C'est un langage de programmation et de description basé sur la réécriture, c'est un langage simple et expressif qui propose un petit nombre d'expressions grammaticales, mais il est spécifique et permet une description fluide de plusieurs types d'applications différentes. Ce dernier est un langage de programmation puissant et compétitif qui prend facilement en charge les prototypes rapides [33].

4.2 Les caractéristiques du Maude :

Maude présente de nombreuses caractéristiques très importantes telles que :

➲ **Simplicité :**

La programmation en Maude est très simple et facile à comprendre. Il existe des équations et des règles, et il y a dans les deux cas une simple sémantique de réécriture dans laquelle les instances du côté gauche sont remplacées par les instances correspondantes du côté droit. En plus, Maude possède un ensemble réduit de mots clés et de symboles, et quelques orientations et conventions à maintenir.

➲ **Expressivité :**

Il est possible d'exprimer naturellement dans Maude un très grand nombre d'applications, en commençant par les systèmes déterministes arrivant aux systèmes concurrents. Il offre aussi un du cadre sémantique, dans lequel non seulement les applications, mais des formalismes entiers (langages, logiques et autres) peuvent être naturellement exprimées.

➲ **Puissance :**

Il est possible d'utiliser ce langage non seulement pour les spécifications exécutables et le prototypage des systèmes, mais aussi pour la programmation réelle et le développement.

➲ **Multi-paradigme :**

Il combine la programmation déclarative et orientée objet. Dans Maude, les spécifications non exécutables sont appelées *les théories*, et leurs axiomes sont utilisés pour imposer des exigences formelles sur les modules programmes et sur les interfaces des modules paramétrés.

4.3 Différents modules d'une spécification Maude

Le système Maude offre trois types de module pour décrire la spécification d’un système.

a) Modules fonctionnels :

Ce type de module est utilisé pour décrire la partie statique d'un système. Ce dernier est définit par les types de données, les opérations utilisées ainsi que les équations. Les types de données sont constitués de termes sans variables. 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. Une équation est déclarée par le mot clé **eq** ou **ceq** si elle est conditionnelle. 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**. Un module fonctionnel a la structure suivante : **fmod …… endfm**.

Exemple :

Soit le module fonctionnel suivant :

```
fmod BASIC-NAT is
sort Nat .                                        *** une sorte
op 0 : -> Nat [ctor] .                            *** une opération
op s_ : Nat -> Nat [ctor] .
op _+_ : Nat Nat -> Nat .
op max : Nat Nat -> Nat .
vars N M : Nat .                                  *** deux variables
eq 0 + N = N .                                    *** une équation
eq s M + N = s (M + N) .
eq max(0, M) = M .
eq max(N, 0) = N .
eq max(s N, s M) = s max(N, M) .
endfm
```

b) Modules Système

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) [34]. Ils sont introduits par les mots clés **mod** <*Corps du module*> **endm** où le corps du module spécifie une *théorie de réécriture* R = *(∑ , E∪ 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'* **if** *cond* .

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

Exemple :

Soit le module systèmes suivant qui implémente un opérateur de choix (?) entre deux entiers :

```
mod CHOICE-INT is
including INT .
op _?_ : Int Int -> Int .            *** définition de l'opérateur
vars I J : Int .
rl [choose_first] : I ? J => I .          ***  régle de réécriture
rl [choose_second] : I ? J => J .
endm
```

c) Modules orientés objet

Les systèmes concurrents orientés objet peuvent être définis par le biais des modules orientés objets présentés par le mot-clé **omod ..... endom** utilisant une syntaxe plus pratique que celle des modules systèmes. Particulièrement tous les modules orientés objet incluent implicitement le module *CONFIGURATION*, possède la structure d'un multi-ensemble formé d'objets et de messages et déclaré en utilisant la syntaxe du langage Maude grâce à l'opération :

Op _: Configuration Configuration → Configuration [ctor assoc comm id:null]

➥ Un "*objet"* est alors noté ainsi: **<O:C |a1:v1…,a_n:v_n>**

- O : est l'identificateur de l'objet,
- C : est la classe correspondante de l'objet,
- a_i , i = 1 à n : sont les attributes de l'objet,
- v_i , i = 1 à n : sont les valeurs des attributs.

➥ *"Les classes"* sont déclarées en utilisant la syntaxe suivante:

Class C | a_1:s_1…,a_n:s_n

Où C est le nom de la classe et s_i sont les types ("sortes*"*) respectifs des attributs *ai*.

- "*Un message"* est déclaré en utilisant le mot-clés: *msg* . Par exemple, une déclaration d'un message de données ("*Data*") envoyé et reçu par deux objets identifiés ("*Oid*") : émetteur et récepteur peut être déclaré comme suit :

Msg to : Oid Oid Data → Msg

Les interactions concurrentes entre les différents messages sont axiomatisées par des règles de réécriture conditionnelles.

Bien que les modules système de Maude soient suffisants pour la spécification des systèmes orientés objets, il y a des avantages conceptuels importants fournis par la syntaxe de modules orientés objets. Ces derniers permettent à l'utilisateur de penser et exprimer ses idées en orientés l'objet. Les modules orientés objets sont transformés en des modules systèmes pour des buts d'exécution. L'exemple suivant réalise les opérations de crédit, débit sur un compte banquaire et de transfert entre deux comptes à travers d'un module orientée objet.

Exemple :

```
omod accnt is
Protecting nat .
class accnt |bal: nat.
msg credit debit : oid nat => message.
msg transfer-from - to - : nat oid oid => message.
vars a, b :oid.
vars m ,n, n' :nat.
rl[crdt] credit (a,m) < a; accnt | bal:n> =><a; accnt | bal : n+m> .
rl[dbt] debit (a,m)< a; accnt | bal:n> =><a; accnt | bal : n-m>
  if  (n >= m) .
rl[trsf] transfer (m) from (a) to (b) < a; accnt | bal:n> =><b;
accnt | bal : n'> =><a; accnt |      bal : n-m><b; accnt | bal :
n+m>  if (n >= m) .
endom .
```

4.4 Autres langages basées sur la logique de réécriture :

4.4.1 CafeOBJ :

C'est une nouvelle génération des langages de spécifications algébriques qui a été développée au Japon. L'équipe de CAFEOBJ est guidée par le prof. Kokichi Futatsugi, à JAIST (Japon), qui est un des fondateurs originaux d'OBJ2. CAFEOBJ est un langage de spécifications algébrique *exécutable*, et un successeur moderne OBJ, il est prévu pour les spécifications des systèmes

réels, la vérification formelle des caractéristiques, prototypage rapide, ou même de programmation.

4.4.2 ELAN :

Ce langage fournit un environnement pour les systèmes de spécification et prototypage des déductions dans un langage basé sur des règles commandées par des stratégies. Son but est de soutenir la conception des preuves de théorèmes.

5 Conclusion

Dans ce chapitre, nous avons présenté les concepts généraux de la logique de réécriture. C'est un cadre formel de raisonnement correct et d'analyse d'une grande variété de systèmes concurrents en particulier, ayant des états et évoluant en termes de transitions. Ses différents concepts sont implémentés à travers son environnement Maude**.**

Chapitre 3

Proposition d'une sémantique OO pour les réseaux de Petri sous Maude

Introduction

D'une part, les réseaux de Petri représentent un outil mathématique pour la modélisation des systèmes dynamiques ce qui nous aide à comprendre le comportement des systèmes de manière plus simple. Cet outil a été utilisé dans de nombreux domaines.

D'autres part, ces derniers nécessitent l'utilisation d'outils formels pour vérifier la concurrence, le comportement et la fiabilité des modèles basés sur eux. Maude fournit une batterie (collection) d'outils de vérification et aussi un cadre unificateur pour la spécification des réseaux de Petri via la logique de réécriture.

Dans ce chapitre nous allons proposer une sémantique orienté objet sous Maude pour les réseaux de Petri. En même temps, on va comparer cette proposition avec la sémantique originale et la sémantique améliorée pour les réseaux de Petri afin de montrer les points forts de notre proposition.

1 Le système Maude et Réseaux de Petri

Maude [19] est un langage exécutable de haut niveau efficace basé sur la logique de réécriture [35], qui inclut les paradigmes fonctionnels et orienté objet comme sous-langages. Par conséquent, Maude fournit une sémantique logique très appropriée pour spécifier des systèmes concurrents dans un style orienté objet de haut niveau. Grâce à l'ensemble des outils d'analyse associés [36, 37, 38], Maude possède d'excellentes capacités pour devenir un bon choix pour l'analyse formelle des systèmes orientés objet [39, 40]. Pratiquement, Maude fournit une syntaxe simple pour décrire les concepts orientés objet (Object, Msg (messages) and Configuration) par l'intermédiaire du module prédéfini `CONFIGURATION`, et ainsi, modélise l'état concurrent d'un système (or configuration) comme un multi-ensemble d'objets et de messages.

Dans ce contexte, les réseaux de Petri ont été bien étudiés dans les deux travaux pionniers [20, 21], où les auteurs ont proposé une sémantique basée sur la logique de réécriture pour les réseaux de Petri.

2 Travaux connexes

Une sémantique basée sur la logique de réécriture a été adoptée pour décrire les réseaux de Petri, car cette sémantique était destinée aux réseaux de Petri standard et ensuite a été généralisée à un large éventail de réseaux de Petri. Pour illustrer brièvement les bases des sémantiques existantes, nous considérons l'exemple du réseau de Petri donné dans la Figure 7.

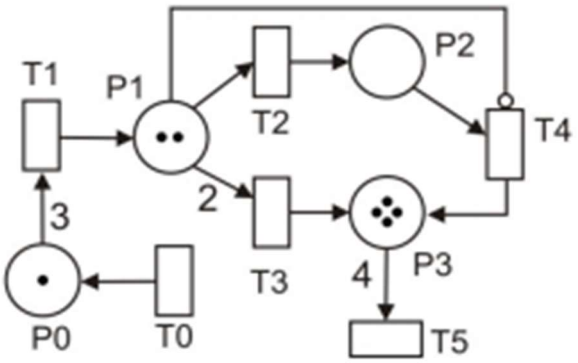


Figure 7 : Exemple de réseau de Petri

2.1 Sémantique originale

2.1.1 Aspects structurels

Pour décrire un réseau de Petri, il faut utiliser des types de données abstraits (appelés en Maude, **`sorts`**) : **`Place`** et **`Marking`**. Ces types sont définis comme des **`sorts`** et des **`subsorts`** comme suit :

```
sorts Place Marking .

subsorts Place  => Marking .
```

Les places sont annoncées comme suit :

```
ops P0 P1 P2 P3  =>  Place .
```

L'état actuel est déterminé via le marquage du réseau de Petri en utilisant l'opérateur d'union multi-ensembles (multiset) comme suit :

```
op null : => Marking .

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

Autrement dit, chaque marquage est représenté par un élément du type **`Marking`** qui est caractérisé par les propriétés (**`assoc`**) (resp, **`comm`**) pour dire que c'est un opérateur associatif (resp, commutatif). **`null`** représente l'élément vide (neutre) sur le type marquage. Le marquage initial est également exprimé sous forme d'un opérateur puis déterminé par une équation comme suit :

```
op initial : => Marking .

eq initial = P0 P1 P1 P3 P3 P3 P3 .
```

De cette façon, on déclare seulement les places contenant des jetons à l'état initial.

2.1.2 Aspects Comportementaux :

Du fait que le marquage d'un réseau de Petri est lié aux franchissements de ses transitions, 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] : (termset-1) => (termset-2)
```

Dans cas, le terme **`termset-1`** (resp, **`termset-2`**) contient l'ensemble des places d'etrées (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`**.

Nous allons maintenant appliquer ces étapes pour décrire la spécification de notre exemple du réseau de Petri (la Figure 7).

```
fmod PETRI-NET-SIGNATURE is
sorts Place Marking .
subsorts Place < Marking .
op null : -> Marking .
ops P0 P1 P2 P3 : -> Place .
op __ : Marking Marking -> Marking
[assoc comm id: null] .
op initial : -> Marking .
eq initial =P0 P1 P1 P3 P3P3P3 .
endfm
```

```
mod PETRI-NET is
protecting PETRI-NET-SIGNATURE .
var M : Marking .
rl [T0] : M => M P0 .
rl [T1] : P0 P0P0 => P1 .
rl [T2] : P1 => P2 .
rl [T3] : P1 P1 => P3 .
***rl [T4] : inexpressible (arc inhibiteur)
rl [T5] : M P3 P3P3P3 => M .
endm
```

Listing 1. Module fonctionnel et système

Comme indiqué dans cette spécification, la partie statique dans réseaux de Petri est définie dans, (Module fonctionnel) et donc la partie dynamique est exprimée par les règles de réécriture dans Module système.

2.2 Sémantique améliorée

La sémantique améliorée a été proposée dans [19] pour offrir plus d'avantages par rapport à la sémantique originale et afin de rendre la spécification des réseaux de Petri plus claire et puissante. Pour illustrer les éléments de cette sémantique, nous prenons toujours l'exemple de la Figure 7 et lui appliquerons cette la sémantique.

2.2.1 Aspects structurels

L'idée principale de la sémantique améliorée est basée sur la description détaillées d'une place d'un réseau de Petri. Donc, pour décrire un réseau de Petri, une place est identifiée par son nom (**`sort: PlaceName`**) qui est attaché au numéro du jeton qu'il porte (**`sort : int`**). Cette sémantique garde le même principe pour décrire le marquage d'un réseau de Petri. Autrement dit, la description du partie dynamique est basée sur l'utilisation du concept mathématique de base qui est multi-ensembles (multi-set). Par conséquent, ces éléments peuvent être décrit comme suit :

```
sorts PlaceName Place Marking .

subsorts Place < Marking .

ops P0 P1 P2 P3 : => PlaceName .

op <_,_> : PlaceName Int => Place [ctor] .

op _ _ : Marking Marking => Marking [assoc com id: null] .
```

Dans cette spécification améliorée, le nombre de jetons dans une place est toujours présent dans la spécification. En plus, lors de la description du marquage initiale, il est nécessaire de décrire l'état de toutes les places du réseau de Petri pas seulement les places contenant des jetons. Ceci est illustré dans la spécification suivante :

```
Op initial : => Marking .

Op initial = <P0,1>,<P1,2>,<P2,0>,<P3,4>.
```

2.2.2 Aspect Comportementaux

Le nouveau marquage M' d'un réseau de Petri est spécifié après le franchissement d'une transition dans une marquage M suivant la formule qui suit :

$$M' = M - pré\ (P_i\ ,T) + post\ (T\ ,\ P_j\ ).$$

Où avec chaque franchissement d'une transition il y aura un changement (mise à jour) de l'état des places de cette transition, i.e, des jetons seront supprimés des places d'entrées et d'autres ajoutés dans les place de sorties. Généralement, le franchissement d'une transition est conditionnel, où la règle de réécriture sera accompagnée d'une condition. La syntaxe d'une telle règle de réécriture aura la forme suivante:

```
crl [(T-labal)]: (Left-Hand-Side) -> (Right-Hand-Side) if  cond .
```

Où **cond** représente la condition d'activation de la transition (ou condition de franchissement). C-à-d, il précise le nombre de jetons nécessaire dans les places d'entrées.

Nous appliquons la sémantique améliorée pour avoir la spécification du réseaux de Petri de la Figure 7.

```
fmod PETRI-NET-SIGNATURE is
  proteting INT .
  sorts PlaceName  Place  Marking .
  subsorts Place < Marking .
  ops P1 P2 P3 P4 : =>PlaceName .
  op < _ , _> : PlaceName INT => Place [ctor].
  op _ _ : Marking Marking => Marking  [ctorassoc comm Id : null ] .
  op null : Marking .
  op initial : => Marking .
  eq initial = <P0,1> <P1,2> <P2,0> <P3,4> .
endfm
mod PETRI-NET is
  protecting PETRI-NET-SIGNATURE .
var  x1,x2,x3.
rl [T0]: <P0,x1> => <P0,x1+1 >.
crl [T1]: <P0,x1><P1,x2> =><P0,x1-3><P1, x2+1> if ( x1>=3) .
crl [T2]: <P1,x1><P2,x2> =><P1,x1-1><P2,x2+1> if (x1>=1) .
crl [T3]: <P1,x1><P3,x2> =><P1,x1-2><P3,x2+1> if (x1>=2).
crl [T4]: <P1,x1><P2,x2><P3,x3> =><P1,x1><P2,x2-1><P3,x3+1 >
          if (x1<1) ∧ (x2>=1) .
crl [T5]: <P3,x1> =><P3,x1-4> if (x1 >=4) .
endm
```

2.3 Limites des sémantiques précédentes

Bien que la sémantique originale offre une description simple des réseaux de Petri et il a été adoptée dans plusieurs travaux de recherche. Cette sémantique présente certaines limites majeures dans la description des réseaux de Petri. Par exemple, le nombre des jetons est exprimé avec le type le place qui était censé représenter les places lui mêmes ce qui conduit à un conflit sémantique. En plus, le nombre de jetons qui se trouvent dans une place n'est pas compté sans ajouter d'équations supplémentaires, parce qu'il se manque de la multiplicité des parenthèses (répétition des noms de place). Cette dernière limite ne nous permet pas de tester des places qui ont des jetons 0 et nous ne permet pas de décrire les arcs inhibiteurs.

D'autre part, bien que la sémantique améliorée couvre les défauts de la sémantique originale, elle ne peut toujours pas décrire avec précision l'état de réseaux de Petri. C'est-à-dire, il n'est pas possible de dire si l'état de transition est actif ou non, ce qui lui rendre un peu limitée pour la simulation du comportement d'un réseau de Petri.

3 Sémantique proposée

Comme tout autre formalisme orienté objet, un réseau de Petri peut être considéré comme un multi-ensemble d'objets (place et transitions). L'ensemble des objets du système communiquent au moyen de messages qui sont utilisés pour décrire l'état actuel des transitions du réseau de Petri (activées ou non). Ces messages seront modifiés après le déclenchement des transitions associés ou s'ils ne restent pas activés (cas des transitions avec arc inhibiteur). Dans ce contexte, Maude permet de spécifier, d'exécuter et même d'analyser des modèles de réseaux de Petri en tant que système objet [35].

Par conséquent, dans cette section, nous allons présenter une sémantique orientée objet en détaillant les aspects structurels et comportementaux d'un réseau de Petri. La sémantique proposée sera illustrée au moyen du réseau de Petri donné dans la Figure 8. On note que les fondements théoriques montrant la représentation de divers concepts orientés objet dans la logique de réécriture peuvent être trouvés dans [35].

Pour faciliter la manipulation de notre travail au lecteurs, on va copier l'image de la Figure 7 de nouveaux ici.

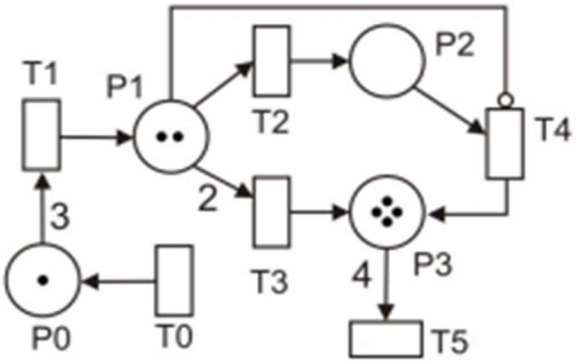


Figure 8 : Exemple de réseau de Petri

3.1 Aspects structurels

3.1.1 Modélisation des places :

Une place est un objet passif dans le système ; elle a un nom (label) et elle contient une information publique (nombre de jetons). En plus, la classe les places pourrait être d'abord déclarée dans Maude avec la syntaxe suivante :

```
class PLACE | N : Int
```

Où **PLACE** désigne le nom de la classe et **N** représente l'attribut stockant le nombre actuel de jetons qu'il contient. Par la suite, une place de réseaux de Petri sera définie comme un objet **Pi** appartenant à une classe **PLACE** comme suit :

```
<Pi:PLACE | N:val >
```

Où **Pi** désigne l'étiquette de place (utilisée comme identifiant d'objet), et **N** est l'attribut de la place qui enregistre le nombre de jetons se trouvant dans la place et val est la valeur correspondante. Les noms des places sont déclarés comme suit :

```
ops P0 P1 P2 P3 : -> Oid .
```

Par exemple, le terme **<P1:PLACE | N:1>** représente la place **P1** qui contient un seul jeton.

3.1.2 Modélisation des transitions :

En revanche, une transition réseaux de Petri sera considérée comme un objet actif puisqu'elle est capable de modifier les attributs d'autres objets (place) et ainsi de changer tout l'état du réseau

de Petri (marquage) après le franchissement. De plus, chaque transition doit envoyer un message (resp, changer le message qui) indiquant son état (activé ou non) chaque fois qu'il y a un changement dans cet état ou même influencer sur l'état d'une autre transition (cas de confluence entre transitions). Pratiquement, une transition sera déclarée comme un identifiant d'objet. Cependant, un message pourrait être déclaré en utilisant le mot-clé op, et avoir comme résultat sort Msg. La déclaration correspondante pourrait être la suivante :

```
ops T0 T1 T2 T3 T4 T5 : ->Oid .
op enable(_,_) : OidBool -> Msg .
```

Cette déclaration définit les transitions (T0 T1 T2 T3 T4 T5) du réseaux de Petri et un message nommé **`enable (_,_)`** qui est paramétré par l'identifiant de l'objet émetteur (transition) et une valeur booléenne pour indiquer s'il est activé ou non.

3.1.3 Modélisation du marquage :

Comme indiqué précédemment, un réseaux de Petri peut être vu comme un multi-ensemble d'objets et de message qui peut être définie comme suit :

```
sorts Object Msg Configuration .
subsorts Object Msg < Configuration .
op none : -> Configuration [ctor] .
op __ : Configuration Configuration -> Configuration
                                        [ctor assoc comm id: none] .
```

L'état initial de notre exemple de réseau de Petri est un terme de la sorte **`Configuration`** qui décrit l'état du réseau (places et transitions) et il pourrait alors être déclaré comme suit :

```
op initial : -> Configuration .
eq initial = < P0:PLACE | N:1 >< P1:PLACE | N:2 >< P2:PLACE | N:0 >
< P3:PLACE | N:4 > enable(T0,true) enable(T1,false) enable(T2,true)
                  enable(T3,true) enable(T4,false) enable(T5,true) .
```

3.2 Aspects comportementaux

Le comportement d'un réseau de Petri est régi par des règles d'activation et de déclenchement. Alors que le premier détermine si et quand une transition T est activée, le second détermine

comment les jetons sont retirés de ses places d'entrée et ajoutés à ses places de sortie. De plus, le déclenchement d'une transition T peut activer ou désactiver d'autres transitions T_j, y compris la transition T elle-même selon le nouveau marquage. De plus, une transition doit être activée avant d'être déclenchée. Par conséquent, il est nécessaire qu'une sémantique proposée pour les réseaux de Petri assure que les règles d'activation priment sur les règles de déclenchement. Dans notre cas, nous avons assuré ce point en utilisant des équations pour décrire les règles d'activation tout en utilisant des règles de réécriture pour décrire celles de déclenchement.

3.2.1 Règles d'activation :

A chaque fois que toutes les places d'entrée Pi d'une transition T sont disponibles — en vérifiant les conditions associées **Pre(P_i,T)** —, la règle d'activation de la transition T est évaluée comme vraie et donc un « message » **enabled(T,true)** est envoyé à indiquent ce changement d'état de la transition T. L'existence de tels messages d'activation dans la spécification reflète le comportement réel d'un réseau de Petri car ils sont nécessaires à des fins de simulation. Par conséquent, les règles d'activation pour chaque transition T seront décrites par une équation comme suit :

```
ceq [<enable-Ti>] : <Left-Hand-Side>= <Right-Hand-Side>if Cond .
```

Le côté gauche d'une telle règle décrit l'état des places d'entrée P_i à la transition T avec un message **enable(T, false)** décrivant l'état de la transition T avant d'être activée. De même, le **Right-Hand-Side** contient le même ensemble de places que le **Left-Hand-Side** avec un changement dans l'état du message **enable(T,true)**. La condition nécessaire pour activer la transition T peut être définie soit en utilisant une seule déclaration ou une conjonction de connecteurs associatifs.

Par exemple, les équations qui décrivent les règles d'activation des transitions T_2 et T_3 dans notre réseau de Petri pourraient être données comme suit :

```
ceq[EN-T2]: <P1:PLACE | N:x> enable(T2,false) = <P1:PLACE | N:x>
enable(T2,true)             if       (x>=1) .

ceq[EN-T3]: <P1:PLACE | N:x> enable(T3,false) = <P1:PLACE | N:x>
enable(T3,true)            if (x>=2) .
```

3.2.2 Règles de franchissement :

Une transition activée T dans un réseau de Petri peut être déclenchée sans condition. Par conséquent, le changement se produit effectivement au niveau de l'ensemble des places d'entrée et de sortie d'une telle transition T en retirant un nombre de jetons égal à **Pre(P_i,T)** de la première et en ajoutant un nouveau nombre de jetons, égal à **Post (T,P_j)** à ce dernier. Ainsi, chaque fois qu'un réseau de Petri est marqué par M, sa nouvelle marquage M' après le déclenchement d'une transition T, est définie comme suit :

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

Pour décrire exactement ces actions, la règle de réécriture correspondante pourrait être déclarée comme suit :

```
rl [<fire-Ti>] : <Left-Hand-Side> => <Right-Hand-Side>.
```

Pour chaque tir de transition T_i, le **Left-Hand-Side** (resp, **Right-Hand-Side**) de sa règle de réécriture doit décrire l'état des deux places d'entrée et de sortie avant (resp, après) le tir. De plus, le côté gauche contient le message **enable(T_i,true)** qui sera changé après le tir en **enable(T_i,false)** dans le côté droit. Par exemple, la règle de réécriture décrivant la règle de déclenchement de la transition T1 sur la Figure 7 pourrait être donnée comme suit :

```
vars x y :-> Int .
rl [FI-T1] : < P0:PLACE | N:x1 >< P1:PLACE | N:x2 > enable(T1,true) =>
< P0:PLACE | N:x1-3 >< P1:PLACE | N:x2+1 > enable(T1,false) .
```

3.3 Cas spéciaux :

• Transitions dépendantes (en conflit) :

Le déclenchement d'une transition T_i peut affecter l'état de toutes les autres transitions qui sont en conflit avec elle et elles peuvent alors ne pas rester activées. Par conséquent, tous les messages d'activation des transitions T_j en conflit avec la transition T_i doivent être ajoutés au **Left-Hand-Side** et changés en **enable(T_j,false)** dans le **Right-Hand-Side** de la règle de déclenchement des transitions T_i. Par exemple, considérons le cas du place P_1 de notre

exemple dans la Figure 7, qui est un lieu d'entrée pour T_2 et T_3. Par conséquent, les règles de réécriture décrivant le tir T_2 et T_3 pourraient être données comme suit :

```
vars x y :-> Int . var z :->Bool .

rl [FI-T2] : < P1:PLACE | N:x1 >< P2:PLACE | N:x2 > enable(T2,true)
enable(T3,t) => < P1:PLACE | N:x1-1 >< P2:PLACE | N:x2+1 >
enable(T2,false)  enable(T3,false) .

rl [FI-T3] : < P1:PLACE | N:x1 >< P3:PLACE | N:x2 > enable(T3,true)
enable(T2,t) => < P1:PLACE | N:x1-2 >< P3:PLACE | N:x2+1 >
enable(T3,false) enable(T2,false).
```

• Transitions source et puits :

Une transition source est toujours activée puisqu'elle n'a pas d'emplacements d'entrée. Il n'aura donc pas d'équation d'activation et son message **enable(T_i, true)** reste inchangeable même après le tir. De plus, la règle de réécriture décrivant un déclenchement de transition source (par exemple T_0 sur notre exemple de la Figure 7 est donnée comme suit :

```
rl [FI-T0] : <P0:PLACE | N:x1> enable(T0,true) =><P0:PLACE | N:x1+1>
enable(T0,true) .
```

En revanche, une transition de puits est une transition qui n'a pas de places de sortie et, par conséquent, elle a une équation d'activation et une règle de déclenchement normales. Par exemple, la règle de réécriture décrivant le déclenchement d'une transition de puits (par exemple, T_5) pourrait être déclarée comme suit :

```
ceq[EN-T5]: <P3:PLACE | N:x1> enable(T5,false) = <P3:PLACE | N:x1>
enable(T5,true) if (x1 >= 4).
rl [FI-T5] : <P3:PLACE | N:x1> enable(T5,true) =><P3:PLACE | N:x1-4>
enable(T5,false) .
```

• Arcs inhibiteurs :

L'arc inhibiteur inverse la logique des conditions d'activation et de déclenchement d'une place, c'est-à-dire qu'une transition ne sera activée que si le place d'entrée contient moins de jetons que

le poids de l'arc inhibiteur. De plus, les jetons dans les places d'entrée des arcs inhibiteurs ne seront pas consommés après le franchissement. Par exemple, l'équation d'activation correspondante d'une transition avec un arc inhibiteur (par exemple la transition T_4 sur la Figure 7) est donnée comme suit :

```
ceq [EN-T4] : <P1:PLACE | N:x1><P2:PLACE | N:x2> enable(T4,false) =
<P1:PLACE | N:x1> <P2:PLACE | N:x2> enable(T4,true)  if (x1 == 0) and
(x2 >= 1) .
```

D'autre part, la règle de réécriture à utiliser pour décrire le déclenchement d'une telle transition T_4 est donnée comme suit :

```
rl [FI-T4] : <P1:PLACE | N:x1><P2:PLACE | N:x2><P3:PLACE | N:x3>
enable(T4,true) => <P1:PLACE | N:x1><P2:PLACE | N:x2-1><P3:PLACE |
N:x3+1> enable(T4,false) .
```

Dans cette règle, nous avons essayé de conserver la structure générale d'une règle de tir sinon, l'état du place lié à l'arc inhibiteur (**<P1:PLACE | N:x>**) pourrait être omis puisque son état ne changera pas après le tir et par conséquent, la règle de réécriture sera la suivante :

```
rl [FI-T4] : <P2:PLACE | N:x1><P3:PLACE | N:x2> enable(T4,true) =>
<P2:PLACE | N:x1-1><P3:PLACE | N:x2+1> enable(T4,false) .
```

De plus, si une transition T_i liée à un arc inhibiteur vers une place Pj est activée, elle peut être désactivée avant même qu'elle ne se déclenche, surtout si la place P_j contient de nouveaux jetons, c'est-à-dire après avoir tiré des transitions T_k qui ont P_j comme place de sortie. Par conséquent, le message d'activation de la transition T_i doit être ajouté au côté gauche et changé en **enable (Ti,false)** dans le côté droit de la règle de déclenchement des transitions T_k Par exemple, la règle de franchissement correspondante de la transition T_1 de la Figure 9) pourrait être donnée comme suit :

```
vars x1, x2  :-> Int . var  t :->Bool .
rl [FI-T1] : < P0:PLACE | N:x1 >< P1:PLACE | N:x2 > enable(T1,true)
enable(T4,t) => < P0:PLACE | N:x1-3 >< P1:PLACE | N:x2+1 >
enable(T1,false) enable(T4,false) .
```

Enfin, nous donnons la spécification complète du précédent réseau de Petri selon la sémantique proposée comme suit :

```
mod PETRI-NET is
  protecting BOOL .  protecting INT .
  including CONFIGURATION .
  op PLACE : -> Cid .
  op N :_ : Int -> Attribute [gather(&)] .
  ops P0 P1 P2 P3 : ->Oid .
  op initial : -> Configuration .
  ops T0 T1 T2 T3 T4 T5 : ->Oid .
  op enable(_,_) : Oid Bool -> Msg .
  vars x1,x2,x3 : Int .
  var t : Bool .
  eq initial = < P0 : PLACE | N : 1 >< P1 : PLACE | N : 2 >< P2 :
PLACE | N : 0 >< P3 : PLACE | N : 4 > enable(T0,true) enable(T1,false)
enable(T2,true)enable(T3,true) enable(T4,false) enable(T5,true) .
ceq[enable-T1] : < P0 : PLACE | N : x1 > enable(T1,false) = < P0 :
PLACE | N : x 1> enable(T1,true) if (x >= 3) .
ceq[enable-T2] : < P1 : PLACE | N : x1 > enable(T2,false) = < P1 :
PLACE | N : x1 > enable(T2,true) if (x1 >= 1) .
ceq[enable-T3] : < P1 : PLACE | N : x1 > enable(T3,false) = < P1 :
PLACE | N : x1 > enable(T3,true) if (x1 >= 2) .
ceq[enable-T4] : < P1 : PLACE | N : x1 >< P2 : PLACE | N : x2 >
enable(T4,false) = < P1 : PLACE | N : x1 >< P2 : PLACE | N : x2>
enable(T4,true) if (x1 < 1 ) and (x2 >= 1) .
ceq[enable-T5] : < P3 : PLACE | N : x1 > enable(T5,false) = < P3 :
PLACE | N : x1 > enable(T5,true) if (x1 >= 4) .
rl[fire-T0] : < P0 : PLACE | N : x1 > enable(T0,true) =>< P0 : PLACE |
N : x1 + 1 >  enable(T0,true) .
rl[fire-T1] : < P0 : PLACE | N : x1 > enable(T1,true) enable(T4,t) <
P1 : PLACE | N : x2 > => < P0 : PLACE | N : x1 - 3 >< P1 : PLACE | N :
x2 + 1 > enable(T1,false)  enable(T4,false) .
rl[fire-T2] : < P1 : PLACE | N : x1 >< P2 : PLACE | N : x2 >
enable(T2,true) enable(T3,t) =>< P1 : PLACE | N : x1 - 1 >< P2 : PLACE
| N : x2 + 1 > enable(T2,false)enable(T3,false) .
rl[fire-T3] : < P1 : PLACE | N : x1 >< P3 : PLACE | N : x2 >
enable(T3,true) enable(T2,t) =>< P1 : PLACE | N : x1 - 2 >< P3 : PLACE
| N : x2 + 1 > enable(T3,false)  enable(T2,false) .
rl[fire-T4] : < P1 : PLACE | N : x1 >< P2 : PLACE | N : x2 >< P3 :
PLACE | N : x3 > enable(T4,true) =>< P1 : PLACE | N : x1 >< P2 : PLACE
| N : x2 - 1 >< P3 : PLACE | N : x3 + 1 > enable(T4,false) .
rl[fire-T5] : < P3 : PLACE | N : x1 > enable(T5,true) =>< P3 : PLACE |
N : x1 - 4 > enable(T5,false) .
endm
```

4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle sémantique basée sur la logique de réécriture pour les réseaux de Petri, où cette sémantique proposée est basée sur le paradigme orienté objet sous Maude. L'avantage majeur de cette sémantique réside dans envoie des messages exprimant le statut de transition après chaque de franchissement dans réseau de Petri étudié. Cette sémantique donne une description plus précise et plus claire des cas de réseaux de Petri, et peut être utilisée pour aider des développeurs à réaliser un outil de simulation basée sur une sémantique formelle.

Chapitre 4 : Conception de l'application de transformation automatique

1 Introduction

Les systèmes doivent être conçus à l'aide des méthodes de modélisation qui facilitent leur développement et leur analyse. Dans ce chapitre, nous montrerons les diagrammes les plus importantes pour notre système de génération automatique de spécification Maude des réseaux de Petri, puis l'environnement utilisé pour le développement.

2 Description générale du système :

Le processus de la génération automatique de spécification Maude des réseaux de Petri dans notre système passe par une suites d'étapes qui peuvent être résumé dans la figure suivante.

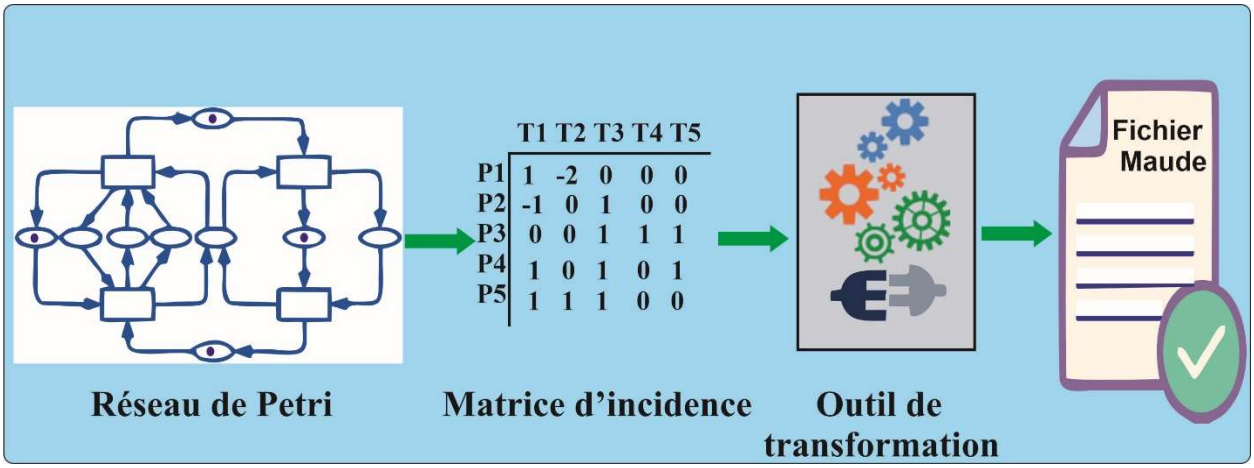


Figure 10: Description des étapes de transformation

Dans notre système de transformation, l'utilisateur doit suivre les étapes suivantes :

- Préparer de la matrice d'incidence à partir d'un réseau de Petri, où le nombre de lignes et de colonnes de la matrice d'incidence est déterminé à partir du nombre de places et de transition du réseau de Petri.
- Sélectionner le choix de l'existence des arcs inhibiteurs.
- Remplir la matrice d'incidence.
- Analyse et manipulation de la matrice d'incidence.
- Génère la spécification orientée objet correspondante.
- Exporter la spécification dans un fichier Maude.

3 Digrammes UML :

Les diagrammes UML permettent de représenter facilement les composants des systèmes orientés objet et de simuler leur structure. Nous utilisons les digrammes suivants :

3.1 Digramme cas d'utilisation :

Le diagramme cas d'utilisation décrit le comportement fonctionnel d'un système logiciel en première modélisation. La Figure 11 décrit le comportement de notre application, où l'utilisateur entre les informations de la matrice d'incidence et à travers laquelle la spécification de Maude pour le réseau de Petri étudié est automatiquement générée et exportée .

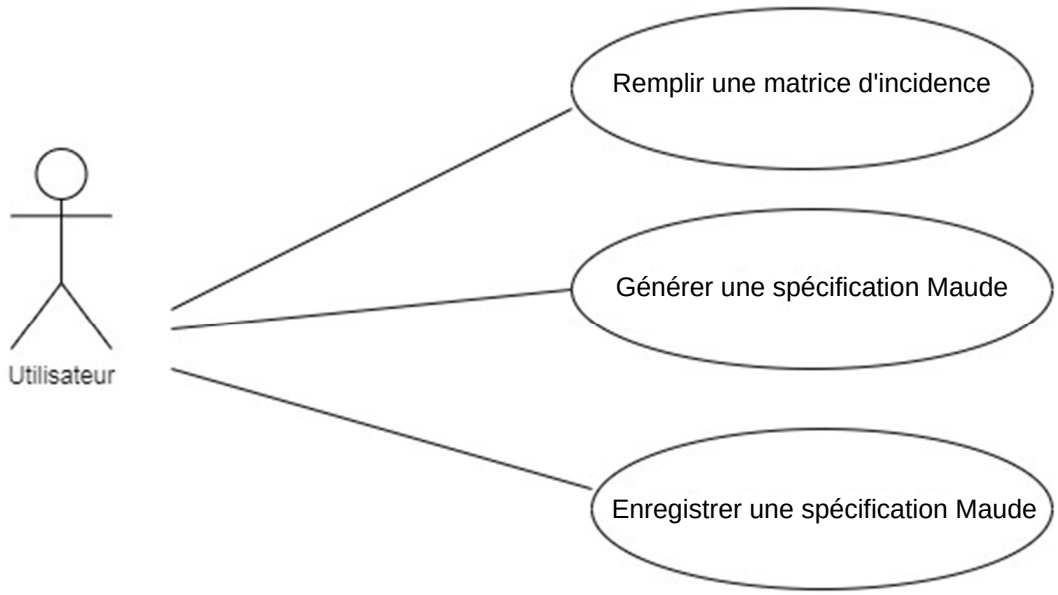


Figure 11:Diagramme cas d'utilisation générale .

3.2 Digramme de séquence :

Le digramme de séquence est une représentation pour décrire les messages entre objets lors d'une interaction dans l'ordre chronologique, et représente également les structures de contrôle entre objets, où des systèmes simples sont modélisés dans un seul diagramme. La Figure 12 représente le digramme de séquence du cas le plus important qui consiste à la génération d'une spécification Maude.

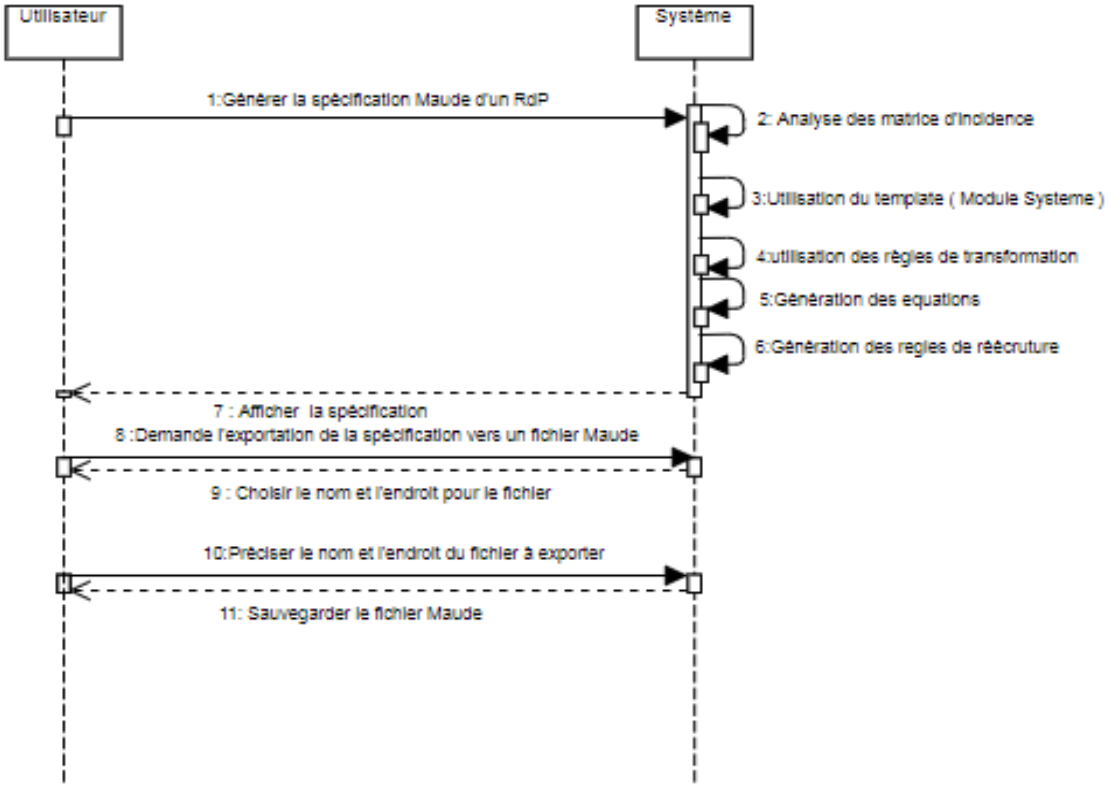


Figure 12 :Diagramme de séquence pour le cas de génération d'une spécification Maude .

Le scénario :

- L'utilisateur demande de générer une spécification à partir de la matrice d'incidence.
- Le système demande d'entrer le nombre des places et des transitions.
- L'utilisateur définit le nombre des places et des transitions.
- Le système demande la possibilité de l'existence des arcs inhibiteur.
- Le système demande de remplir la matrice d'incidence.
- L'utilisateur remplit la matrice d'incidence.
- L'utilisateur demande la génération de la spécification.
- Le système analyse la matrice d'incidence et génère puis affiche la spécification.
- L'utilisateur demande l'exportation de la spécification vers un fichier Maude.
- Le système demande de choisir le nom et le chemin pour le fichier.
- L'utilisateur préciser le nom et l'endroit du fichier à exporter.
- Le système sauvegarde le ficher Maude.

3.3 Digramme de classe :

C'est le diagramme le plus important en UML et il est largement utilisé pour documenter l'architecture d'un système ou un programme car il décrit ce qui devrait être (composants) dans le système conçu. La Figure 13 représente le diagramme de classes de notre système.

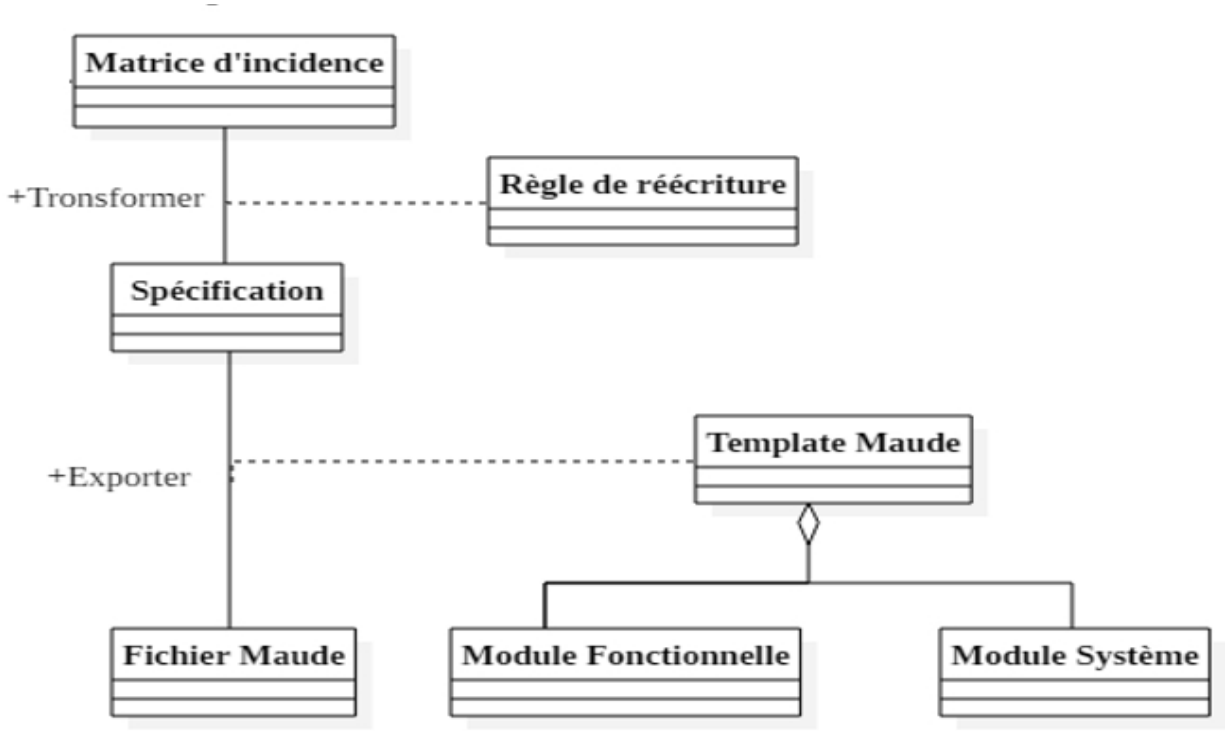


Figure 13:Diagramme de classe générale .

4 Implémentation

4.1 Environnement de développement

Nous avons utilisé le langage de programmation Delphi pour implémenter notre outil. Delphi est un langage de programmation produit par Borland Corporation, basé sur le langage Visual Pascal orienté objet. Delphi est un langage puissant, facile et populaire. C'est un environnement de développement où il est utilisé pour développer des programmes et des applications rapidement sous Windows.

4.2 Présentation de l'application :

4.2.1 Interface principale :

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

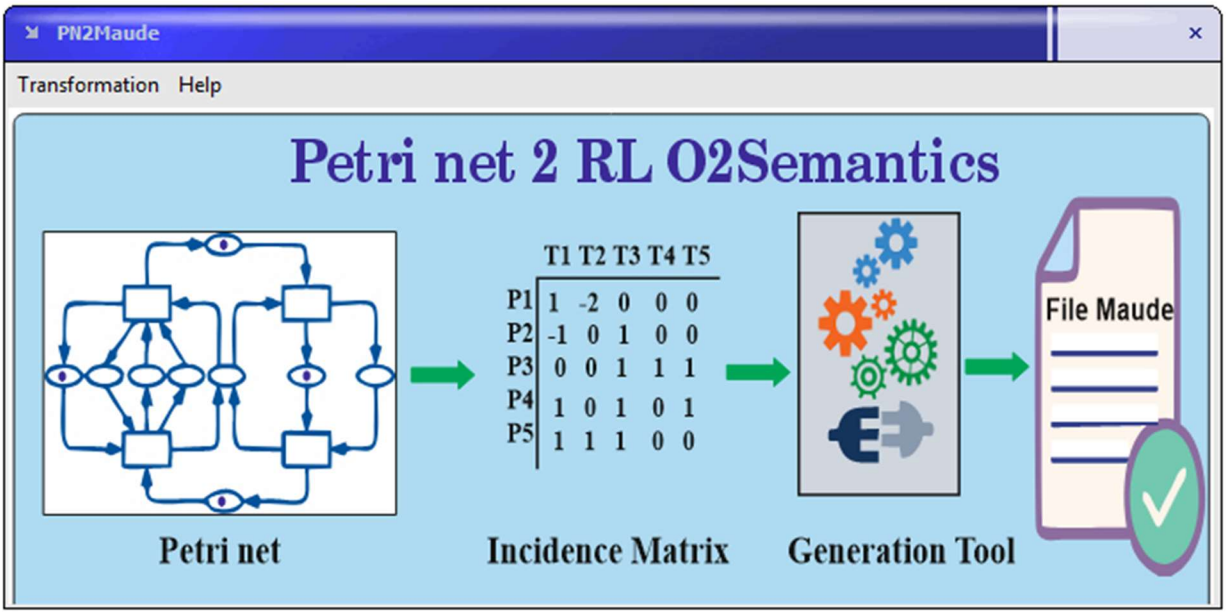


Figure 14: L'interface principale de l'outil

4.2.2 *Cas d'utilisation* :

Cas d'utilisation pour préciser le nombre de places et de transition des matrices d'incidences.

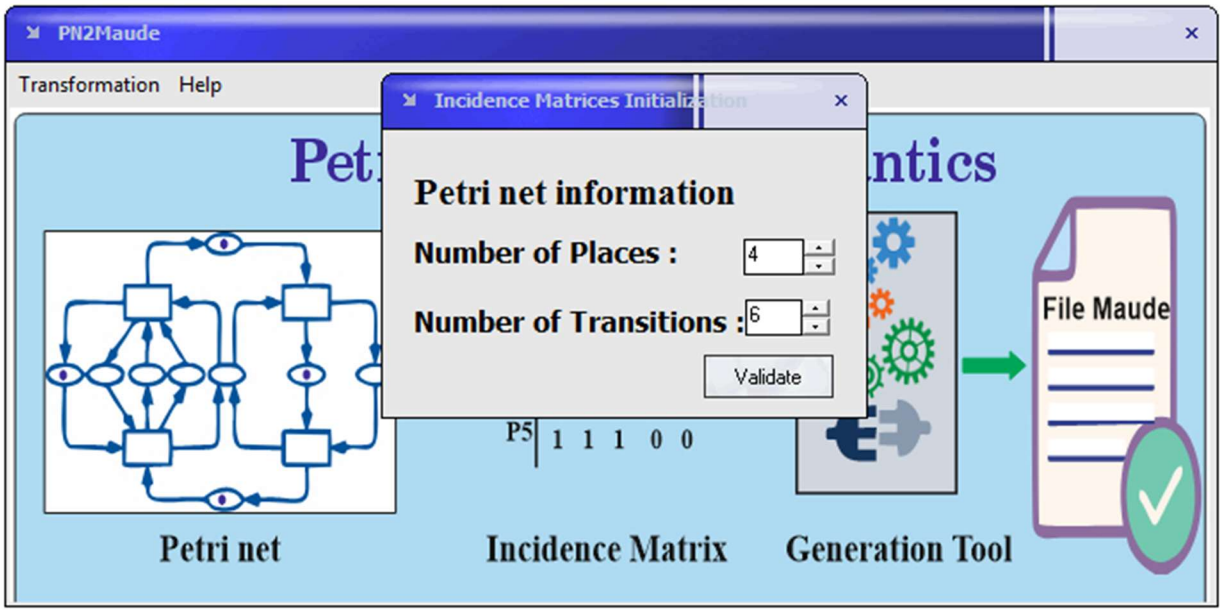


Figure 15: remplissage des matrices d'incidence du RdP.

Puis, l'utilisateur remplit les matrices d'incidences représentant le réseau de Petri.

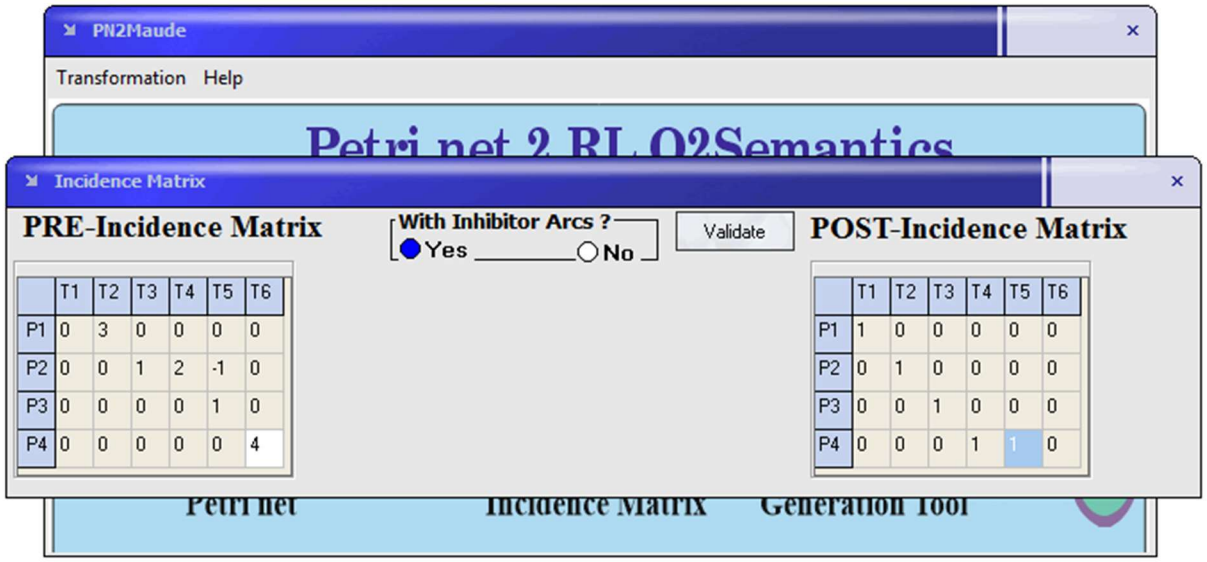


Figure 16: cas de remplissage des matrices d'incidences.

4.2.3 Fenêtre de spécification générée :

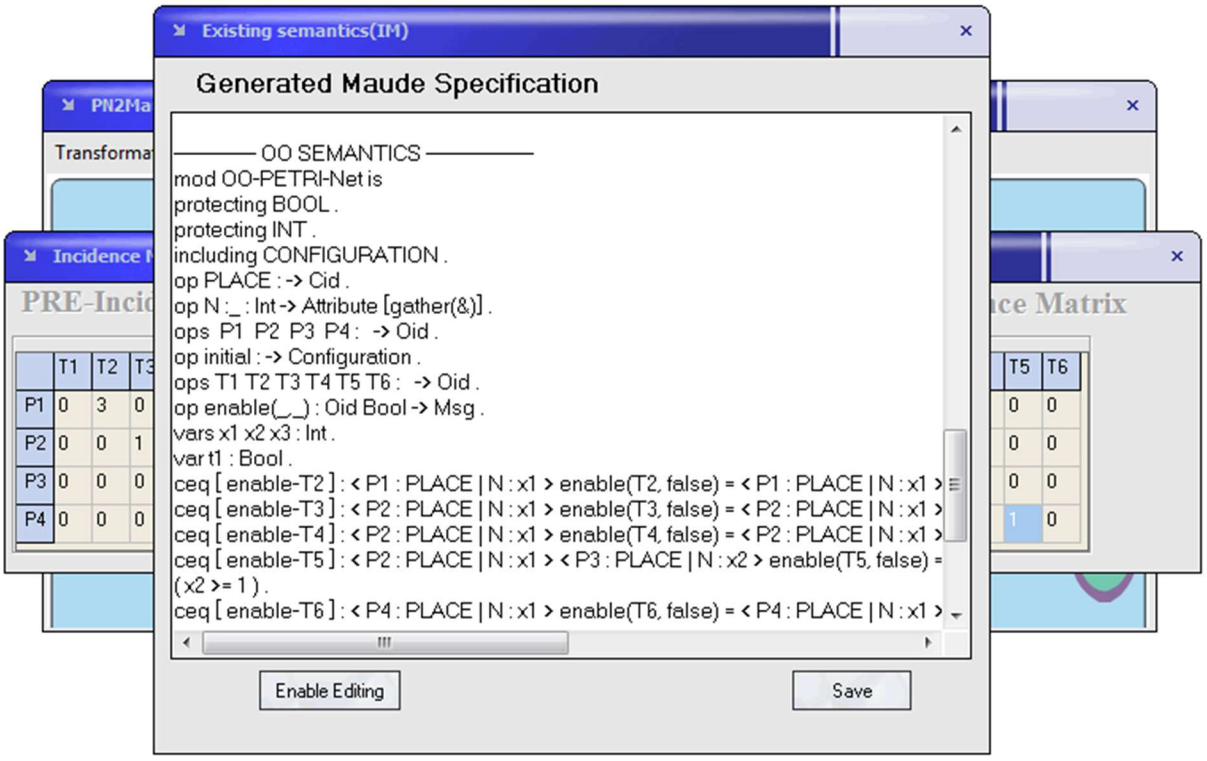


Figure 17:Fenêtre de spécification générée par l'outil.

5 Conclusion :

Dans ce chapitre, nous avons fournis une description générale de notre système. Et nous avons présenté les trois digrammes UML les plus importantes pour notre système (Digramme cas d'utilisation, Digramme de séquence, Digramme de classe). Nous envisageons de continuer le développement de notre outil pour supporter l'utilisation des représentations XML pour les réseaux de Petri.

Références

1. Miryam, B. (2018). Petri nets. In Strategies and Techniques for Quality and Flexibility, pages 81–99. Springer.
2. Silvia, R. Patrick, M. Felix, M. Marta, S. and Fabio, P. (2019). A petri net modeling approach to explore the temporal dynamics of the provision of multiple ecosystem services. Science of The Total Environment, 655:1047–1061, 2019.
3. Bao, S. (2018). Petri net modeling in flexible manufacturing systems with shared resources. Computer Aided Design, Engineering, and Manufacturing: Systems Techniques and Applications, Volume V, The Design of Manufacturing Systems, 5.
4. Hongcheng, Li. Haidong, Y. Bixia, Y. Chengjiu, Z. and Sihua, Y. (2018). Modelling and simulation of energy consumption of ceramic production chains with mixed flows using hybrid petri nets. International Journal of Production Research, 56(8):3007–3024.
5. Zimu, W. Xia, W. Yingjie, J. Yan, Z. and Jingjing, Z. (2019). Guidance performance evaluation method for infrared imaging guided missile based on extended object-oriented petri net. Optik, 185:88–96.
6. Wangyang, Y. Zhijun, D. Lu, L. Xiaoming, W. and Richard, C. (2018). Petri net-based methods for analyzing structural security in e-commerce business processes. Future Generation Computer Systems.
7. Abdul H. (2018). New hybrid petri net application for modeling and analyzing complex smart microgrid system. J Eng Appl Sci, 13(9):2713–2721.
8. Dmitry, Zaitsev. (2018). Simulating cellular automata by infinite petri nets. Journal of Cellular Automata,13.
9. Carvalho, R. Fons V. and Clarimar, C. (2018). Bio-modeling using petri nets: a computational approach. In Theoretical and Applied Aspects of Systems Biology, pages 3–26. Springer.
10. Safae, C. and Mouline, S. (2018). Modelling and simulation of biochemical processes using petri nets. Processes, 6(8):97.
11. Aristyo, B. Pradityo, K. Agustinus, T. Yul, Y. and Widyotriatmo, A. (2018). Model checking-based safety verification of a petri net representation of train interlocking systems. In 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), pages 392–397. IEEE.
12. Zbigniew, S. (2018). Pnes: Tools for the design and analysis petri nets. In 5th International Conferenceon Control, Decision and Information Technologies (CoDIT), pages 391–396. IEEE.
13. Ralf, H. Christoph, B. and Philo R. (2018). Ompetri: A new petri net simulation environment based on openmodelica. In Proceedings of the 10th International Conference on Bioinformatics and Biomedical Technology, pages 57–61. ACM.

14. Chalika, S and Wiwat, V. (2018). Automatic transformation of ordinary timed petri nets into event-b for formal verification. Engineering Journal, 22(4):161–175.

15. Remi, B. (1995). Approaches in unifying petri nets and the object-oriented approach. In 1st Workshop on Object-Oriented Programming and Models of Concurrency, within the 16th International Conference on Application and Theory of Petri nets, volume 27.

16. Michael, Z. and Armin, H. (1999). Techniques for integrating petri-nets and object-oriented concepts. In Working Papers in Information Systems. University of Bayreuth.

17. Xin-yang, W and Xiao-Yue, W. (2015). Extended object-oriented petri net model for mission reliability simulation of repairable pms with common cause failures. Reliability Engineering & System Safety, 136:109{119}.

18. Yang, Y. Wei, X. Sixin, W. and Kunlun, W. (2018). Modeling and analysis of cps availability based on the object-oriented timed petri nets. In 37th Chinese Control Conference (CCC), pages 6172{6177. IEEE.

19. Boucherit, A. Barkaoui, K. & Hasan, O. (2021). An Enhanced Rewriting Logic Based Semantics for High-Level Petri nets. In The International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency.

20. Meseguer, J. (1992). Conditional rewriting logic as a unified model of concurrency. Theoretical computer science, 96(1):73{155}.

21. Mark-Oliver, S. Meseguer, J. and Olveczky, C. (2001). Rewriting logic as a unifying framework for Petri nets. In Unifying Petri Nets, pages 250{303. Springer.

22. Boucherit, A. Khababa, A. and Laura M C. (2018). Automatic generating algorithm of rewriting logic specification for multi-agent system models based on petri nets. Multiagent and Grid Systems, 14(4):403{418.

23. Barkaoui, K. Hicheur, A. Kheldoun, A. and Ding, L. (2016). Modelling and analyzing home care plans using high-level petri nets. In 13th International Workshop on Discrete Event Systems (WODES), pages 284{290}. IEEE.

24. Padberg, J. and Alexander, S. (2016). Model checking reconfigurable petri nets with Maude. In International Conference on Graph Transformation, pages 54{70}. Springer.

25. Kheldoun, A. Barkaoui, K. JiaFeng Z, and Ioualalen M. (2015). A high level net for modeling and analysis reconfigurable discrete event control systems. In IFIP International Conference on Computer Science and its Applications, pages 551{562. Springer.

26. Bendiaf, M. (2018). Spécification et vérification des systèmes embarqués temps réel en utilisant la logique de réécriture (Thèse de Doctorat), Université Mohamed Khider Biskra.

27. Madani, k. Madjouri S. (2020). Réalisation d'un outil de génération automatique de" spécification Maude (nouvelle sémantique) des réseaux de Petri. Université Echahid hamma lakhdar, El-Oued.

28. Hettab, A. (2009) De M-UML vers les réseaux de Petri « Nested Nets » : Une approche basée transformation de graphes. Thèse de Magistère. Université de Mentouri Constantine.
29. ZERNADJI, T. (2009). Une approche de modélisation des logiciels à base de composants par les réseaux de Petri (Thèse de Doctorat, Université de Batna 2).
30. Boucherit, A. (2019). Contribution à la conception d'architecture logicielle sur des systèmes critiques basés agent (Thèse de Doctorat). Université de Ferhat Abbas Setif.
31. Boudiaf N. (2007), "Développement des Outils Basés Maude pour les ECATNets. Domaine d'Application : Analyse des Programmes Ada". Thèse de doctorat, université de Mentouri Constantine.
32. Mcclatchey, R. (2004). REFINER: Environnement logiciel pour le raffinement d'architectures logicielles fondé sur une logique de réécriture (Doctoral dissertation, Université de Savoie).
33. Douibi, H.(2006). Intégration de XML dans le cadre de la logique de réécriture. mémoire de Magister en informatique., Université Mentouri de Constantine
34. Meliouh, A. (2021). UML et Model-Checking pour la Modélisation et la Vérification des Systèmes Embarqués (Doctoral dissertation, Université de Mohamed kheider Biskra).
35. Meseguer, J. (1990). A logical theory of concurrent objects, volume 25. ACM.
36. Manuel, C. Palomino, M. (2001). The itp tool. In Logic, Language and Information. Proc. of the First Workshop on Logic and Language, pages 55{62}.
37. Steven, E. Meseguer, J. and Ambarish, S. (2004). The Maude LTL model checker. Electronic Notes in Theoretical Computer Science, 71:162{187}.
38. Durán, F. Meseguer, J. (2010). A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. In International Workshop on Rewriting Logic and its Applications (pp. 69-85). Springer, Berlin, Heidelberg.
39. Antonio, D. Francisco, D. and Meseguer, José. (2016). Towards generic monitors for object oriented real-time Maude specifications. In International Workshop on Rewriting Logic and its Applications, pages 118{133}. Springer.
40. Olveczky, C. (2017). Concurrent objects in Maude. In Designing Reliable Distributed Systems, pages 155{182}. Springer.