الجمهورية الجزائرية الديمقراطية الشعبية
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DR. YAHIA FARES DE MEDEA

FACULTE DES SCIENCES ET DE LA TECHNOLOGIE
Département du Génie Électrique et de l'Informatique



Projet de fin d'étude

Présenté pour l'obtention du Diplôme
De Master en Informatique
Spécialité : Ingénierie des Systèmes et Technologie de Web

THEME

Méthode Formelle pour la Modélisation et la Vérification des Processus Métiers

Cas d'étude : Le langage BPMN (*Business Process Model and Notation*)

Proposé et dirigé par :

Promoteur : Mr. KHELDOUN Ahmed

Co-promoteur : Mr. RAHIM Messaoud

Préparé et présenté par :

Mr.BABCHIKH Mohamed

Mme.BENILLES Ihsene

Année Universitaire : **2015/2016**

Sommaire

Liste des figures **3**
Liste des tables **5**
Résumé **6**
Introduction générale **7**
Chapitre 1 : Les Réseaux de Petri **8**
1.Introduction 8
2.Réseau de Petri 8
2.1.Concepts et définition 8
2.2.Sémantique de RdP 8
2.3.Règle de franchissement 11
2.4.Propriétés des RdP 14
3.RdP Récursif 19
3.1.Introduction 19
3.2.Définition 19
3.3.Marquage étendu 20
3.4.Franchissabilité et franchissement 21
4.Conclusion 23
Chapitre 2 : Les ECATNets Récursifs **24**
1.Introduction 24
2.Définition 24
3.Marquage étendu 26
4.Franchissabilité et franchissement 28
5.Application 30
6.Conclusion 32
Chapitre 3 : Langage BPMN **33**
1.Introduction 33
2.Historique 33
3.Définition 33
4.Les éléments de base de BPMN 34
- Les évènements 34
- Les branchements 36
- Les activités 37
- Les éléments de connexion 39

5.Définition formelle 39
6.Exemple 1 : Modèle BPMN pour une commande de pizza.....40
7.Exemple 2 : Modèle BPMN pour une réservation de voyage 41
8.Exemple 3 : Modèle BPMN pour une déclaration d'assurance.....42
9.Outils utilisés pour la création de BPMN.....42
10.Conclusion.....42
Chapitre 4 : Transformation BPMN vers RECATNets 43
1.Introduction 43
2.Les méta-modèles utilisés pour la transformation 43
2.1 Méta-modèle BPMN 43
2.2 Méta-modèle RECATNet.....45
3.Les règles de transformation.....47
3.1.Les règles de transformation des éléments de base 48
3.1.1. Règle N°1 : Les flux de séquence 48
3.1.2. Règle N°2 : Les évènements 48
3.1.3. Règle N°3 : Les branchements 50
3.1.4. Règle N°4 : Les tâches 52
3.1.5. Exemple 1 : RECATnet pour une commande de pizza 53
3.2.Les règles de transformation des éléments complexes.....53
3.2.1. Règle N°5 : Les sous-processus 53
3.2.2. Règle N°6 : Les boucles.....54
3.2.3. Règle N°7 : L'activité multi Instances 56
3.2.4. Règle N°8 : Gestion des exceptions 57
3.2.5. Exemple 2 : RECATnet pour une réservation de voyage.....59
3.2.6. Exemple 3 : RECATnet pour une réclamation d'assurance d'un accident 60
4.Conclusion 60
Chapitre 5 : Implémentation de l'approche de transformation 61
1.Introduction 61
2.Schéma globale pour la transformation 61
3.Règles ATL 62
4.Exemple de transformation 66
5.Conclusion.....73
Conclusion générale.....74
Références bibliographiques.....76

Liste des figures

Figure 1.1 : Les composants graphiques d'un RdP....9
Figure 1.2 : Représentation graphique d'un RdP....10
Figure 1.3 : Réseau de Petri après le franchissement de T1....12
Figure 1.4 : Changement de marquage avant et après le franchissement....13
Figure 1.5 : Exemple pour la pseudo-vivacité....14
Figure 1.6 : Exemple pour la quasi-vivacité....14
Figure 1.7 : Exemple pour la vivacité....15
Figure 1.8 : Exemple d'un RdP avec une place puit....15
Figure 1.9 : Exemple d'un RdP sans blocage....15
Figure 1.10 : RdP non-borné....16
Figure 1.11 : RdP 2-borné....16
Figure 1.12 : Graphe d'un RdP....17
Figure 1.13 : Représentation graphique d'un RdPR....20
Figure 1.14 : Graphe d'accessibilité du RdPR de la Figure 1.13....21
Figure 1.15 : Franchissement d'une transition élémentaire....22
Figure 1.16 : Franchissement d'une transition abstraite....22
Figure 2.1 : Représentation graphique d'une transition élémentaire de RECATNet....26
Figure 2.2 : Représentation graphique d'une transition abstraite de RECATNet....26
Figure 2.3 : Exemple de RECATNet....27
Figure 2.4 : Séquence de franchissement de RECATNet précédent....28
Figure 2.5 : RECATNet modélisant le processus de réservation de voyage....31
Figure 3.1 : Les éléments d'un BPMN....34
Figure 3.2 : Les types d'évènements....35
Figure 3.3 : Les branchements en sortie d'une étape....36
Figure 3.4 : Les branchements en entré d'une étape....37
Figure 3.5 : Les types d'activités....38
Figure 3.6 : Les types de flux....39
Figure 3.7 : Exemple d'un BPMN pour une commande de pizza....40
Figure 3.8 : Exemple d'un BPMN pour la réservation de voyage....41
Figure 3.9 : Processus BPMN pour la réclamation d'assurance d'un accident....42
Figure 4.1 : Méta-modèle de BPMN....43
Figure 4.2 : Méta-modèle de RECATNet....45

Liste des figures

Figure 4.3 : Transformation de flux de séquence 48
Figure 4.4 : Transformation de l'évènement de début.....48
Figure 4.5 : Transformation de l'évènement intermédiaire 49
Figure 4.6 : Transformation de l'évènement de fin 49
Figure 4.7 : Transformation de fork.....50
Figure 4.8 : Transformation de join 50
Figure 4.9 : Transformation de data-decision 51
Figure 4.10 : Transformation de merge 52
Figure 4.11 : Transformation de tâche.....52
Figure 4.12 : RECATNet du processus de commande de pizza 53
Figure 4.13 : Transformation de sous-processus 54
Figure 4.14 : Transformation de la boucle « while-do ».....55
Figure 4.15 : Transformation de la boucle « do-until » 55
Figure 4.16 : Transformation de l'activité multi Instance 56
Figure 4.17 : Transformation de l'exception d'une tâche.....57
Figure 4.18 : Transformation de l'exception d'un sous-processus 57
Figure 4.19 : RECATNet du processus de réservation de voyage.....59
Figure 4.20 : RECATNet du processus de réclamation d'assurance d'un accident 60
Figure 5.1 : Schéma pour la transformation d'un modèle BPMN vers RECATNet.....61
Figure 5.2 : Code source du helper "In" 62
Figure 5.3 : Code source du helper "Pin" 62
Figure 5.4 : Transformation de flux de séquence 63
Figure 5.5 : Transformation de StartEvent 63
Figure 5.6 : Transformation de fork Gateway 64
Figure 5.7 : Transformation de sous-processus 65
Figure 5.8 : Exemple de modèle BPMN dans BPMN2 Modeler.....66
Figure 5.9 : L'interface de l'importation du fichier BPMN2 Modeler 68
Figure 5.10 : L'interface « générer Parser » 70
Figure 5.11 : L'interface « Transformation ATL » 71
Figure 5.12 : Le graphe résultat de RECATNet 72

Liste des tables

Table 1.1 : Matrice représente les places avant des transitions **9**
Table 1.2 : Matrice représente les places après des transitions **10**

Résumé :

L'objectif de ce projet est de proposer une approche de transformation de langage BPMN vers un modèle formel appelé *RECATNet*. Ce dernier est très connu par son pouvoir d'expression qui facilite la modélisation/analyse des systèmes complexes incluant des sous-processus, multi-instance et des exceptions. En plus, *RECATNet* peut être décrit facilement en termes de la logique de réécriture et par conséquent, le système Maude peut être utilisé lors de la phase d'analyse.

Le résultat de ce projet est de développer un outil JAVA qui permet la génération automatique des modèles de *RECATNets* à partir des processus métiers exprimés en BPMN.

ملخص:

الهدف من هذا المشروع هو اقتراح منهجية لتحويل لغة BPMN (منهج التخطيـط لإدارة الأعمال) إلى نموذج رســـمي يدعى RECATNet . هـذا الأخيـــر معروف بقدرتــه على التعبير الذي يسهل نمذجــة و تحلـيل الأنظمة المعقدة. وبالإضافة إلى ذلك يمكن وصفها بسهولة من حيث منطق إعـــادة الكتابــة وبالتالي يمكن استخدام نظام Maude خلال مرحلة التحليل.

نتيجة هذا المشروع هي تطوير برنامج JAVA تسمح بالإنشـــاء التلقائي لنماذج RECATNet مـــن خلال أساليب تسيير العمل المعبر عنها بمنـــهج التخطيط لإدارة الأعمال.

Abstract:

The aim of this project is to propose an approach to transform a BPMN language into a formal model called RECATNet. This last is well known for his power of expression that facilitates the modeling/analysis of complex systems. In addition, RECATNet can be easily described in terms of the rewriting logic and therefore the Maude system may be used during the analysis phase.

The result of this project is to develop a Java tool that allows the automatic generation of RECATNets models from business processes expressed in BPMN.

Introduction générale

Business Process Model and Notation (*BPMN*) est une notation graphique standardisée pour modéliser des procédures d'entreprises ou des processus métiers. Le but principal de *BPMN* est de fournir une notation qui soit réellement compréhensible par les concepteurs, mais toute fois, *BPMN* est un langage semi-formel qui n'a pas une sémantique définie rigoureusement. Par conséquent, il souffre d'un manque de sémantique formelle qui peut conduire à des incohérences, ambiguïtés et incomplétude dans les modèles développés. En outre, *BPMN* apporte des fonctionnalités supplémentaires tirées de diverses sources, y compris modèles de flux de travail qui sont en mesure de définir : (1) des sous-processus qui peuvent être exécutées plusieurs fois simultanément; et (2) des sous-processus qui peut être interrompu en raison d'exceptions. Ces caractéristiques augmentent les types d'erreurs sémantiques qui peuvent être trouvés dans les modèles *BPMN*. En conséquence, de nombreux chercheurs ont proposé des méthodes formelles pour la description et la vérification des processus métiers exprimés en langage BPMN. Cependant, l'une des faiblesses de ces propositions est leur manque de soutien à la modélisation des processus BPMN complexes impliquant exception, annulation et instanciation multiple de sous-processus. Pour cela, nous avons besoin d'un formalisme qui permet, d'une part, de modéliser leur structure dynamique, et d'autre part, de vérifier l'exactitude de flux de contrôle de ces processus métiers tout en tenant compte de leur aspect de flux de données.

Dans ce mémoire, nous allons utiliser les *RECATNets* pour la vérification des modèles *BPMN* complexes. Le modèle *RECATNets* propose des mécanismes pratiques pour la manipulation des constructions *BPMN* les plus avancées (impliquant exception, sous-processus et plusieurs instances), donc nous allons utiliser ATLAS Transformation Language (ATL) pour la transformation de modèle *BPMN* vers *RECATNet*. A la fin de ce projet, nous allons développer un outil JAVA pour visualiser graphiquement les *RECATNets* générés.

Chapitre 1 : Les Réseaux de Petri

1. Introduction

La théorie des Réseaux de Petri peut être considérée comme toute jeune, puisqu'elle n'est apparue qu'en 1962, dans la thèse du chercheur Carl Adam Petri, intitulée : « Communication avec les automates », et présenté à l'université de DARMSTADT en Allemagne. Ensuite un groupe de chercheurs dirigés par Anatol W.Holt du Massachusetts Institute of Technology (MIT), fut séduit par ce travail. C'est ce groupe qui jettera les bases, entre 1968 et 1976 de ce qui devint les réseaux de Petri [1].

2. Réseau de Petri

2.1. Concept et définitions

Les RdP sont un outil graphique et mathématique qui s'applique à un grand nombre de domaine où les notions d'évènements et d'évolutions simultanées sont importantes. L'utilisation informatique des RdPs fait largement appel à la notion des réseaux autonomes ne dépondant pas d'évènement externes et celles des réseaux généralisés. Ces descriptions permettent en particulier d'étudier les problèmes d'analyse de programme ainsi que de coordination et synchronisation de processus [1].

L'application des RdPs aux systèmes séquentiels a donné naissance à la classe particulière des réseaux synchronisés sur évènement extérieur et réseaux interprétés [1].

2.2. Sémantique de réseau de Petri

Un réseau de Pétri R est un quadruplet <P, T, Pré, Post> tel que :

- P : ensemble fini de places ($P \neq \emptyset$)
- T : ensemble fini de transitions ($T \neq \emptyset$, $P \cap T = \emptyset$)

- Pré : P * T $\longrightarrow \mathbb{N}$

$$\forall p \in P, \forall t \in T : \text{Pré}(p,t) = n \ (n \geq 0)$$

$$\updownarrow$$

Le franchissement de « t » est conditionné par la présence de n ressources dans « p »

- Post : P * T $\longrightarrow \mathbb{N}$

$$\forall p \in P, \forall t \in T : \text{Post}(p,t) = n \ (n \geq 0)$$

$$\updownarrow$$

Le franchissement de « t » produit « n » ressources dans « p »

La Figure 1.1 montre les symboles utilisés pour les éléments de base d'un RdP. Une place sera dessinée par un cercle, une transition par un rectangle, un arc par une flèche et un jeton par un point.

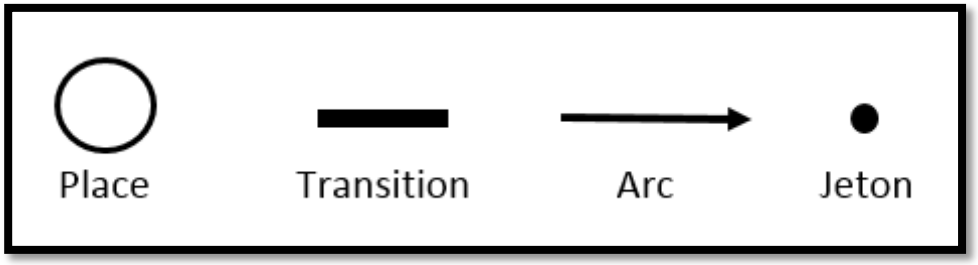


Figure 1.1 : les composants graphiques d'un RDP

Exemple 1:

Soit P = {P1; P2; P3; P4; P5}

T = {T1; T2; T3; T4; T5}.

Soient Pré (représenté par Table 1.1) et Post (représenté par Table 1.2) :

Pré	T1	T2	T3	T4	T5
P1	1	0	0	0	0
P2	0	1	0	0	0
P3	0	0	1	0	0
P4	0	0	0	1	0
P5	0	0	0	1	1

Table 1.1:matrice représente les places avant les transitions

Post	T1	T2	T3	T4	T5
P1	0	0	0	1	0
P2	1	0	0	0	0
P3	1	0	0	0	1
P4	0	1	0	0	0
P5	0	0	1	0	0

Table 1.2:matrice représente les places après les transitions

La Figure 1.2 représente le RdP de l'*Exemple1*, les places P1, P4 contiennent un seul jeton, la place P3 contient trois jetons, la relation place-transition est réalisée si le numéro d'intersection de la place avec la transition dans la matrice pré est supérieur à zéro, la relation transition-place est réalisée si le numéro d'intersection de la place avec la transition dans la matrice post est supérieur à zéro,

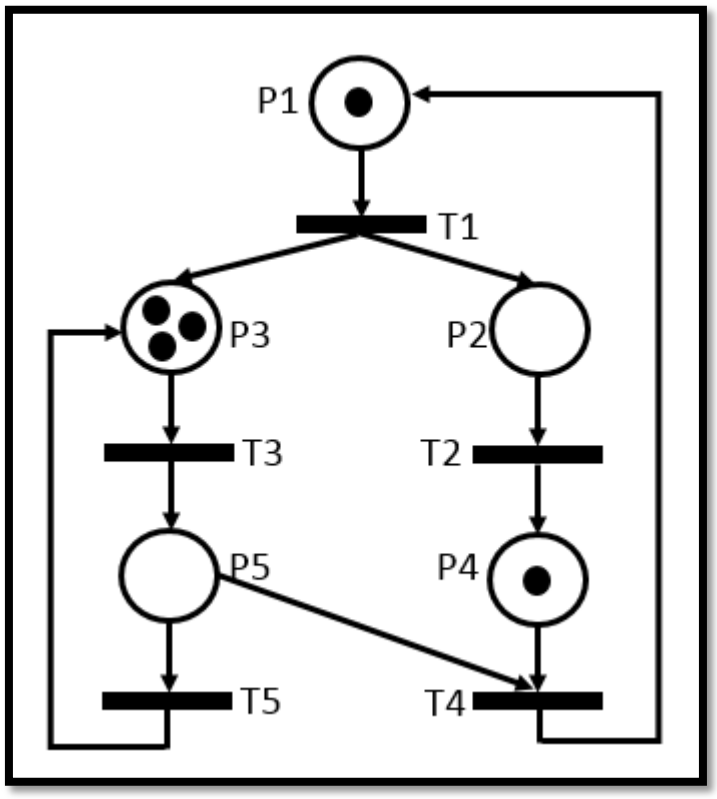


Figure 1.2 : représentation graphique d'un RdP

- **Le marquage**

  Le marquage définie l'état d'un réseau, il associe à chaque place un nombre entier positif ou nul qui représente le nombre de jetons contenus dans cette place.

  M(P') est le marquage de la place «P'», on dit aussi le nombre de jetons contenues dans « P'».

Le marquage des places de la Figure 1.2 est comme suite :

Le marquage initial est $M_0 = \begin{pmatrix} 1 \\ 0 \\ 3 \\ 1 \\ 0 \end{pmatrix}$

2.3. Règle de franchissement

Une transition T' est franchissable si dans chacune de ses places d'entrée il y a au moins le nombre de jetons indiqué par le poids de l'arc correspondant [2]. Formellement :

• T' ∈ T est franchissable à partir de M ssi ∀ P' ∈ P, M(P') ≥ Pré (P',T').

• si « T' » est franchissable, alors son franchissement mène à l'état M' obtenue comme suit :

$$\forall P' \in P,\ M'(P') = M(P') - Pré(P',T') + Post(P',T')$$

Notation : M [T' > M'.

Dans l'Exemple1 nous pouvons dire que 'T1' et 'T3' sont franchissables à partir de M_0 et nous aurons : M_0 [T1 > M_1 et M_0 [T2 > M_2

Le franchissement de « T1 » à partir de M_0 mène au marquage M_1

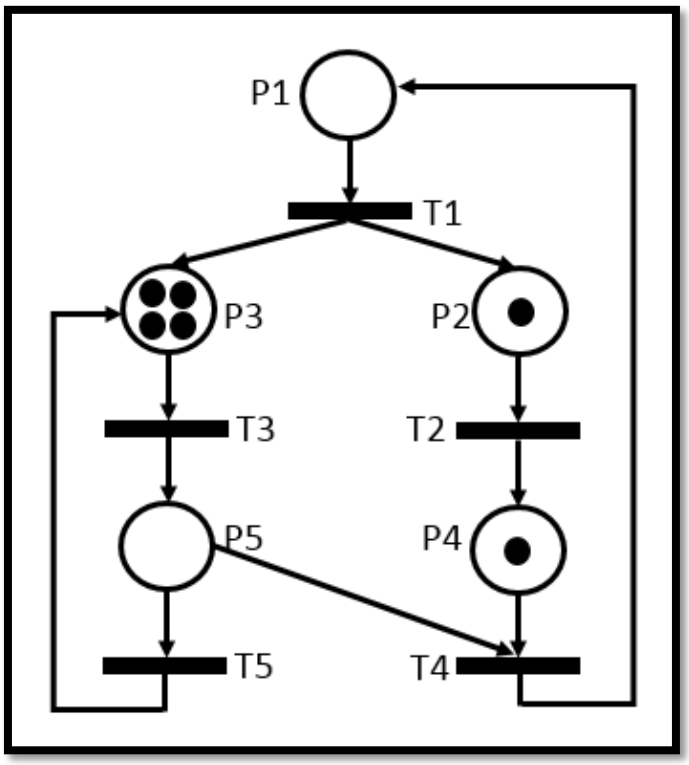


Figure 1.3: Réseau de Petri après le franchissement de "T1"

$$M_1 = \begin{bmatrix} 0 \\ 1 \\ 4 \\ 1 \\ 0 \end{bmatrix}$$

La Figure1.3 représente le graphe du RdP de l'Exemple1 avec le marquage M_1. En effet, le franchissement de T1 consomme le jeton de la place P1 et produit deux nouveaux jetons, un jeton dans la place P2 et l'autre dans la place P3.

Une séquence de franchissements de M_0 à M_n est un mot $\sigma = T_1 \ldots T_n$ tel que il existe des marquages accessibles $M_1, \ldots, M_{n-1}$ vérifiant :

$$M_0[T_1 > M_1 \ldots M_{n-1}[T_n > M_n$$

La Figure1.4 représente le développement de marquage par la séquence de transitions б et {M1, M2, ... , M5} est l'ensemble ses marquages accessibles.

б = T1 T3 T3 T2 T3

M_1, M_2, M_3, M_4, M_5

$M_0[T_1 > M_1[T_3 > M_2[T_3 > M_3[T_2 > M_4[T_3 > M_5$

$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 3 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{б} M_5 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}$$

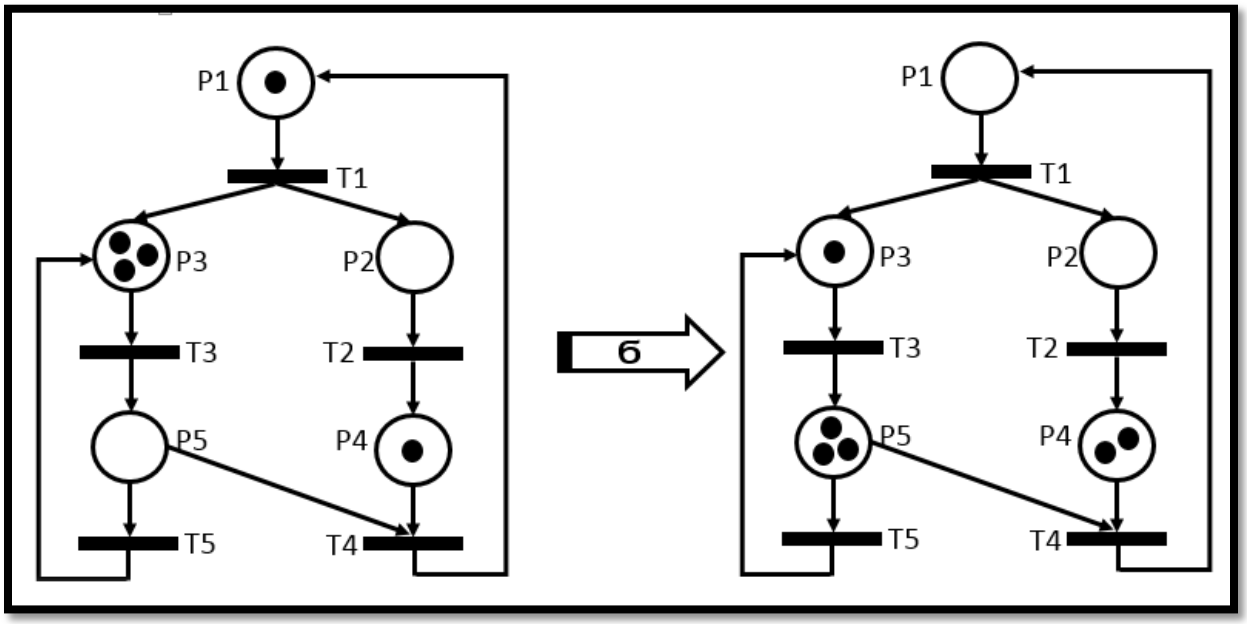


Figure 1.4: Changement de marquage avant et après le franchissement d'une séquence de transitions

2.4. Propriétés des réseaux de Petri

✓ Vivacité

- **Pseudo-vivacité**

Un RdP est pseudo-vivant si pour tout marquage accessible M il existe une transition T telle que M[T >.

La figure 1.5 montre un exemple d'un RdP **Pseudo-vivant,** pour tout marquage accessible, il y a au moins une transition franchissable.

Dans le marquage initial M_0 = (1,0), les transitions T1 et T3 sont franchissable. Si T1 est franchis alors il reste le même marquage, si T3 est franchis alors le marquage devient M_1 = (0,1) et reste toujours le même. Alors la transition T2 est toujours franchissable.

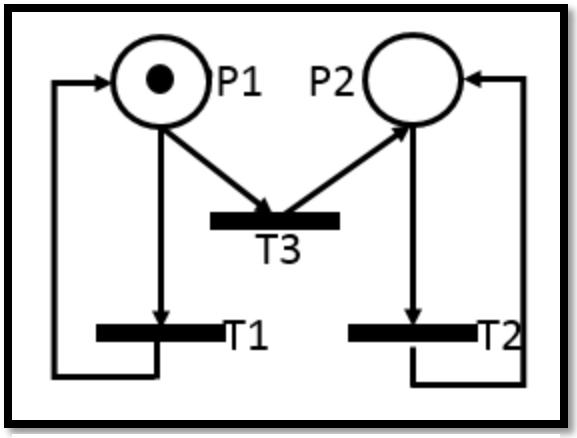


Figure 1.5 : Exemple pour la pseudo-vivacité

- **Quasi-vivacité**

<RdP, M_0> est quasi-vivant si, pour toute transition T, il existe un marquage accessible M tel que M[t >.

La Figure 1.6 représente un exemple d'un RdP **Quasi-vivant**, pour une séquence de franchissement « Ϭ = T1 T2 T3 », toutes les transitions sont franchis.

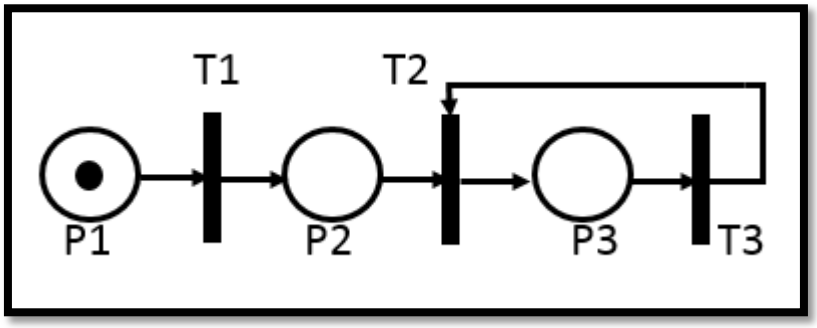


Figure 1.6 : Exemple pour la quasi-vivacité

- **Vivacité**

Un RdP est vivant si pour tout marquage accessible M le RdP est quasi-vivant.

La Figure1.7 montre un exemple d'un RdP **Vivant**, pour chaque franchissement de T1 et T2 les deux transitions sont franchissables.

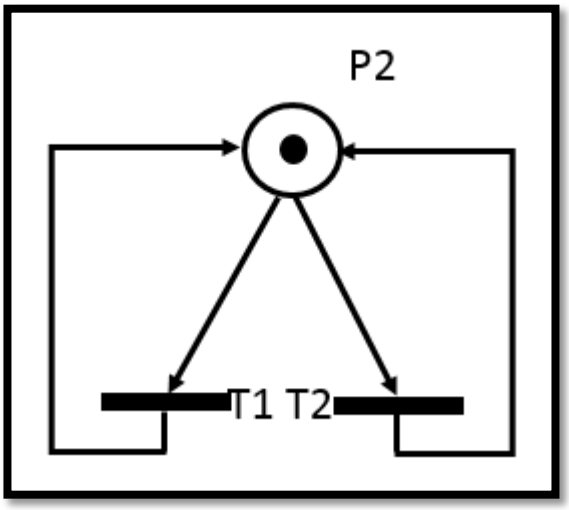


Figure 1.7: Exemple pour la vivacité

✓ **Marquage puits**

C'est un marquage tel qu'aucune transition ne peut être effectuée, la Figure1.8 représente un RdP avec un marquage puits.

✓ **Blocage**

Un réseau marqué est **sans blocage** si aucun de ses marquages accessibles n'est un marquage puits, la Figure1.9 représente un RdP sans blocage.

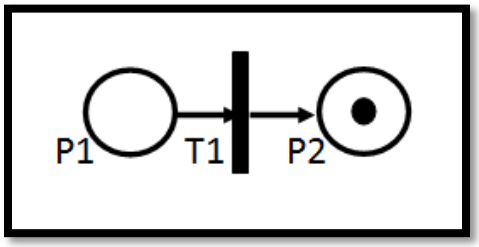


Figure 1.8: exemple d'un RDP avec un marquage puits

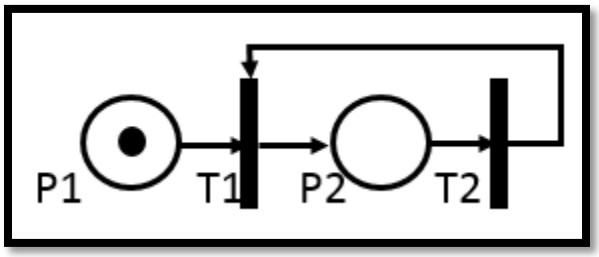


Figure 1.9 : exemple d'un RDP sans Blocage

✓ Caractère borné d'un réseau

Un RdP est non borné si, pour tout entier n, il existe un marquage accessible M' et une place P' tels que M'(P') > n.

Place k-bornée: une place « P'» du réseau marqué (RdP, M_0) est **k-bornée** si pour tout marquage accessible M' depuis le marquage initiale M_0, **M'(P') ≤ k**. Sinon, la place est dite : **non-bornée**.

Soit l'ensemble *A*(RdP, M_0) des marquages atteignables d'un RdP

$$\textbf{P est k-bornée} \leftrightarrow \forall M \in A\ (RdP, M_0),\ M(p) \leq K$$

Réseau borné : Un réseau marqué est borné si toutes ses places sont bornées

Dans la Figure 1.10, la transition T1 est toujours franchissable, donc le nombre de jetons dans P2 augmente toujours.

Puisque P1 est 1-bornée et P2 est non-bornée alors le réseau est non-bornée

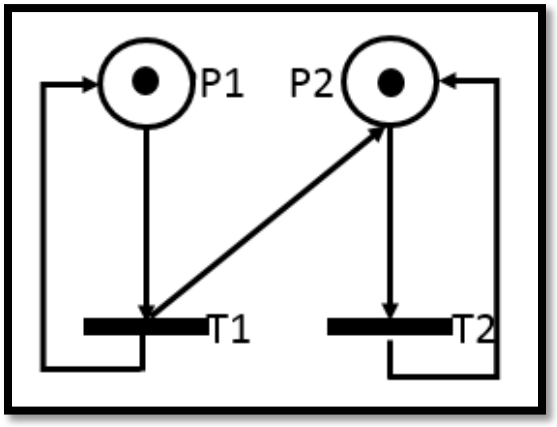


Figure 1.10 : RdP non-bornée

Dans La Figure 1.11 le nombre maximale des jetons que peut contenir une place est 2. Par conséquent, le RdP est 2-borné.

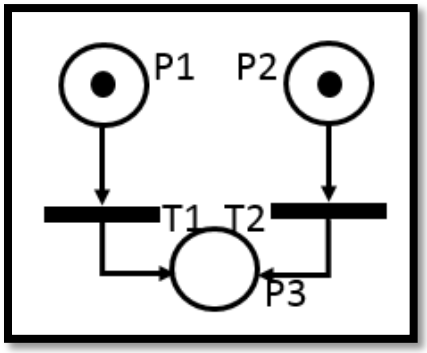


Figure 1.11 : RdP 2-bornée

✓ Conflit

• Conflit structurel

Deux transitions T1 et T2 sont en conflit structurel si et seulement si elles ont au moins une place d'entrée en commun [3]. Formellement :

$$\exists p \in P \text{ telle que Pré } (p , T1) \text{ x Pré } (p , T2) \neq 0$$

Dans le réseau de Petri de la Figure 1.12 les transitions T_1 et T_2 sont en conflit structurel car elles ont toutes les deux la place P2 en entrée.

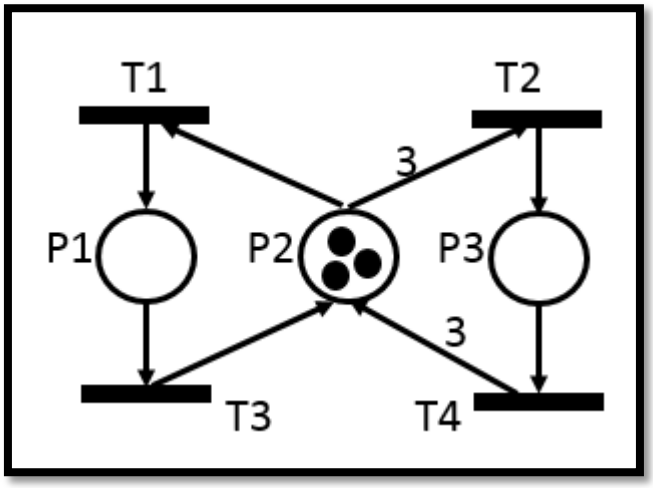


Figure 1.12 : graphe d'un RdP

• Parallélisme effectif

Soit un réseau de Petri de structure R deux transitions T_i et T_j sont en conflit effectif pour un marquage M si et seulement si :

$M \geq$ Pré($*$, T_i) ; $M \geq$ Pré($*$,T_j) et $M \not\geq$ Pré($*$,Ti) + Pré($*$,T_j) .

** Pré* est une application de *P x T* dans N dite d'incidence avant.

Cela veut dire qu'il y a assez de jetons pour que l'une des deux transitions T_i ou T_j soit franchie mais pas les deux à la fois. Dans le réseau de Pétri de la Figure1.12 les transitions T_1 et T_2 sont en conflit effectif pour le marquage initial M_0:

$$M_0 = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} ; \text{Pré}(*, t_1) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} ; \text{Pré}(*, t_3) = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$$

$$\text{Pré}(*, t_1) + \text{Pré}(*, t_3) = \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix}$$

Il y a assez de jetons dans la place P2 pour le franchissement de T_1 ou bien pour le franchissement de T_3 mais pas pour les deux à la fois. Pour le marquage :

$$M = \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix}$$

Une condition nécessaire pour que T_i et T_j soient en conflit effectif est qu'elles soient en conflit structurel.

- **Parallélisme structurel**

Soit un réseau de Petri R, T_i et T_j sont parallèles si et seulement si n'ont aucune place d'entrée en commun. Elles ne sont donc pas en conflit structurel. En conclusion, lorsque l'on considère deux transitions, soit elles sont en conflit structurel, soit elles sont parallèles.

Dans le réseau de Petri de la Fgure1.12 les transitions T_1 et T_3 sont parallèles. En effet, ce qui compte c'est que ces deux transitions n'ont pas de place d'entrée commune. Le fait que la place P1 est place de sortie de T_1 et place d'entrée de T_3, n'est pas un obstacle. La condition pour que T_1 soit franchissable est qu'il y ait au moins un jeton dans la place P2 et celle pour T_3 est qu'il y ait au moins un jeton dans la place P1. Ces deux conditions sont donc bien indépendantes.

3. Réseaux de Petri Récursifs

3.1. Introduction

Le modèle des réseaux de Petri récursifs (abrégés RdPR) est une extension des réseaux de Petri introduit par El Fallah et Haddad en 1996. Ce modèle est utilisé pour la modélisation et la vérification des systèmes dont leur structure change dynamiquement [5].

3.2. Définition

Un RdPR [5] *est un septuple* $R = < P , T , I , W^- , W^+ , \omega , \gamma >$ ou

- P : est un ensemble fini de place.
- T : est un ensemble de transition, une transition T peut être :

- **Transition élémentaire (**T_{el}) : représentée par un rectangle simple comme cela des transitions de réseaux de Petri qu'on a vu. Son franchissement entraine l'exécution de l'action élémentaire qui lui est associée.
- **Transition abstraire (**T_{abs}): représentée par un rectangle à double bord. Son franchissement donne lieu à l'activation d'un RdPR.

- I : est un ensemble fini d'indices, généralement appelé les indices de terminaison.
- W^- : Est la fonction de Pré-conditions.
- W^+ : Est la fonction de Post-conditions.
- ω : Est la fonction du marquage initial définie T_{abs} à $\mathbb{N}^p$, elle associe à chaque transition abstraite T_{abs} un marquage ordinaire.
- γ : Est une famille indexée de représentation effective d'ensembles semi-linéaires de marquages finaux.

La Figure 1.13 montre les composantes graphiques d'un RdPR qui sont:

- {P1, P2, P3, P4, P5, P6} : Ensembles des places.
- T1 : Transition abstraite.
- T2, T3 : Transitions élémentaires.
- <0>, <1> : Les indices de terminaison.
- P4 : Marquage initiale.
- γ_0, γ_1 : Une famille indexée de représentation effective d'ensemble semi-linéaires de marquage finaux.

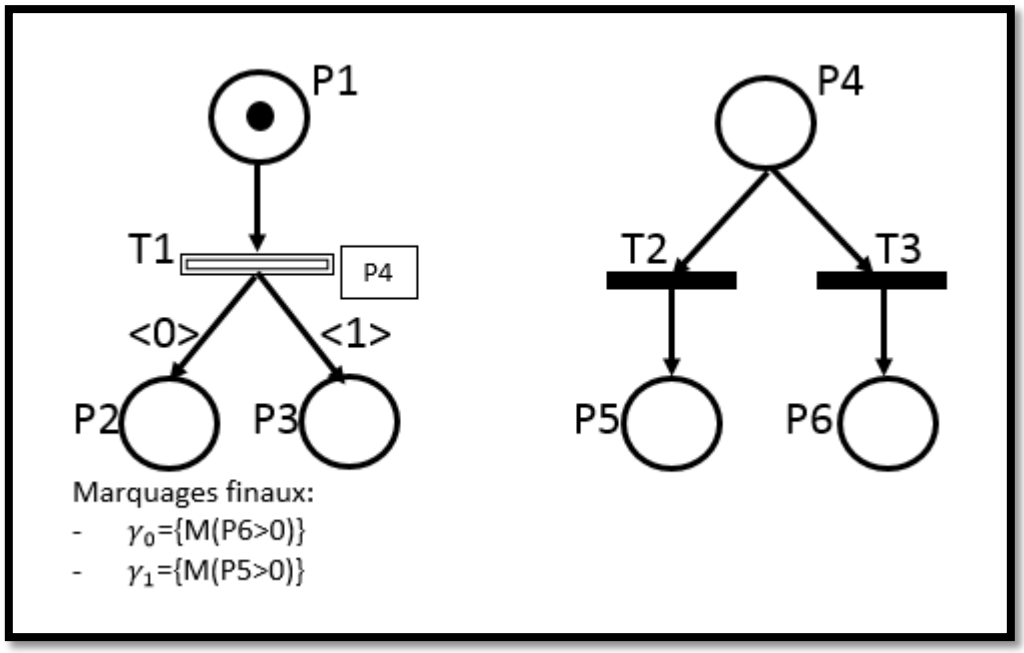


Figure 1.13 : Représentation graphique d'un RdPR

3.3. Marquage étendu

Le marquage étendu d'un RdPR est un arbre marqué $T_r = < V, M, E, A >$ où :

- V est l'ensemble fini des nœuds de l'arbre. Lorsque l'arbre T_r n'est pas vide, $V_0 \in$ V dénote la racine de l'arbre.
- M : V $\rightarrow \mathbb{N}^p$ est une application qui associe à chaque nœud un marquage ordinaire.
- $E \subseteq$ V x V est l'ensemble des arcs.
- A : E $\rightarrow T_{ab}$ est une application associant une transition abstraire à chaque arc de l'arbre.

La Figure 1.14 montre une séquence de franchissement de RdPR de la Figure 1.13.

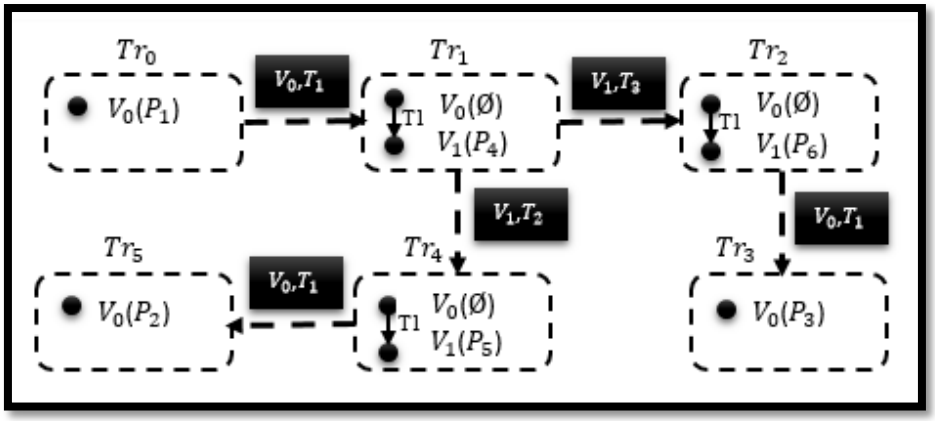


Figure 1.14 : Graphe d'accessibilité du RdPR de la Figure 1.13

3.4. Franchissabilité et franchissement

➢ Définition

- Le franchissement dans un RdPR marqué peut être soit un franchissement d'une transition élémentaire ou abstraite, soit l'exécution d'une étape de coupure (noté τ_i avec $i \in I$).

  Une transition T est franchissable à partir du nœud v ∈V ssi : M(v) ≥ W^- (.,T)

➢ Franchissement d'une Transition élémentaire

Le franchissement d'une transition élémentaire T_1 dans un nœud V_1 d'un marquage étendu $T_r =< V, M, E, A >$ conduit à un marquage étendu $T_{r'} =< V', M', E', A' >$ (noté $T_r \xrightarrow{t,v} T_{r'}$) tel que :

Pour $E'' = \{(v, v') \in E\}$ et $V'' = \{v' \in V \mid (v, v') \in E''\}$

- $\forall p \in P, M(v)(p) \geq W^-(p,t)$
- $V' = V \mid (U_{v' \in v''} Succ(v')), E' = E \cap (v' \times v')$
- $\forall e \in E', A'(e) = A(e)$
- $\forall v' \in V' \mid \{v\}, M'(v') = M(v')$
- $\forall p \in P \mid M'(v)(p) = M(v)(p) - W^-(p,t) + W^+(p,t))$

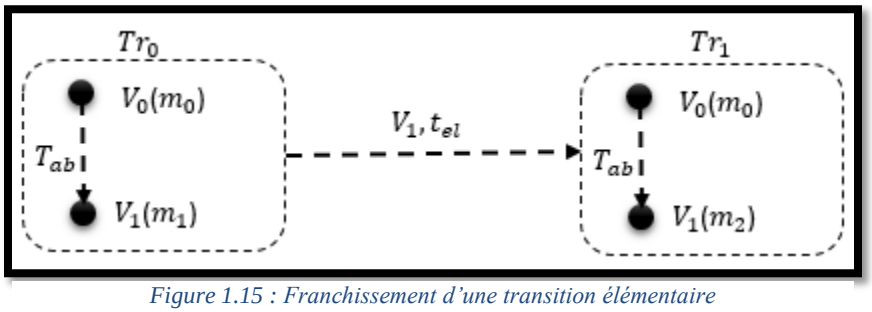


Figure 1.15 : Franchissement d'une transition élémentaire

➢ Franchissement d'une Transition abstraite

Le franchissement d'une transition abstraite T_1 dans un nœud V_1 d'un marquage étendu $T_r = < V, M, E, A >$ conduit au marquage étendu $T_{r'} = < V', M', E', A' >$ (noté $T_r \overset{t,v}{\rightarrow} T_{r'}$) si et seulement si :

- Soit v′ un nouvel identificateur dans l'arbre $T_{r'}$.
- $\forall p \in P, M(v)(p) \geq W^-(p,t)$.
- $V' = V \cup \{v'\}, E' = E \cup \{(v,v')\}, \forall e \in E, A'(e) = A(e), A'((v,v')) = t$.
- $\forall v'' \in V\{v\}, M'(v'') = M(v'')$.
- $\forall p \in P, M'(v)(p) = M(v)(p) - W^-(p,t)$.
- $M'(v') = \Omega(t)$ //le nouveau nœud est associé par le marquage initial de t.

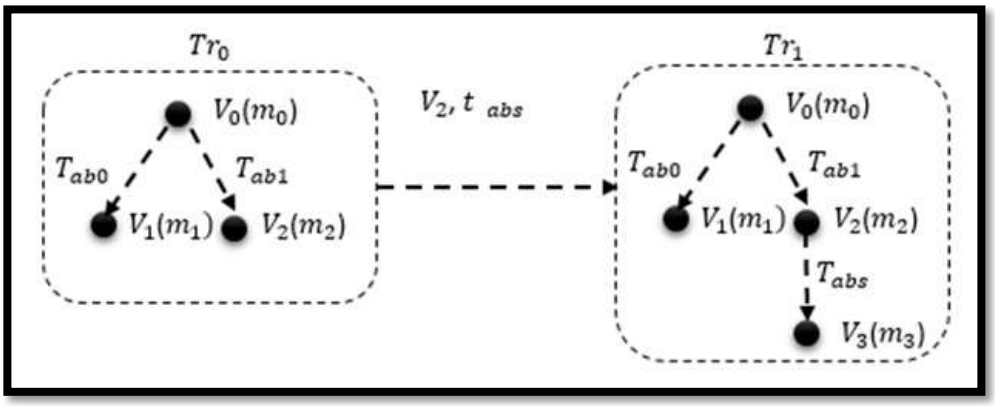


Figure 1.16 : Franchissement d'une transition abstraire

➢ Franchissement d'une étape de coupure

Un franchissement d'une étape de coupure d'un nœud de l'arbre élimine le sous arbre issu de ce nœud, et produit les jetons en sortie de la transition abstraite qui a généré ce nœud.

Formellement, le franchissement d'une étape de coupure τ_i par un nœud v d'un marquage étendu $T_r = < V, M, E, A >$survient quand M(v) ϵ Υ_i mène au marquage étendu $T_{r'} = < V', M', E', A' >$(dénoté par$T_r \xrightarrow{\tau_i, v} T_{r'}$) ssi :

- (v $= v_0$) alors $T_{r'} = \perp$
- (v $\neq v_0$) alors :
  - $V' = V \setminus Succ(v), E' = E \cap (V' \times V'), \forall e \in E', A'(e) = A(e)$
  - $\forall$ v′ ∈ V\{Préd(v)}, M'(v') = M(v')
  - $\forall p \in P, M'\big(Préd(v)\big)(p)\ M\big(Préd(v)\big)(p) + W^+(p, A(Préd(v), v, i).//$

    i représente l'indice de terminaison associé à ϒi.

*Succ(v) : l'ensemble de successeurs directs et indirects incluant pour chaque $v_i \in V$.

*Préd(v) : l'ensemble de prédécesseurs dans l'arbre quand v_i n'est pas la racine de l'arbre.

4. Conclusion

Dans ce chapitre nous avons définis les réseaux de Petri et les règles de franchissement ainsi que leur propriétés. Nous avons introduit le modèle des réseaux de Petri récursifs, son marquage étendu et le franchissement des transitions élémentaires et abstraites. Les réseaux de Petri récursifs sont définis en enrichissant les réseaux de Petri avec des mécanismes de création et suppression dynamiques de processus à travers l'introduction de concept de transition abstraite.

Chapitre 2 : Les ECATNets Récursifs

1. Introduction

Les ECATNets (Extended Concurrent Algebraic Term Nets) sont une catégorie des réseaux de Petri algébriques basés sur une combinaison saine des réseaux de Petri de haut niveau et des types algébriques abstraits. Les réseaux de Petri sont utilisés pour leur pouvoir d'exprimer graphiquement la concurrence et l'aspect dynamique. Les types abstraits algébriques sont utilisés pour leur puissance de décrire abstraitement les données. Leur association dans un seul cadre est motivée par le besoin de spécifier explicitement à la fois le comportement et les structures de données des processus.

Les ECATNets récursifs (en abrégé RECATNets) sont une catégorie des ECATNets ordinaires avec le concept de récursivité. Ils distinguent deux types de transitions comme les RdPR (élémentaire et abstraite). Chaque transition abstraite est associée à un marquage de départ représenté dans un cadre adjacent à cette transition [5].

2. Définition

Un RECATNet est une paire E = (Spec ,R) où :

- Spec = ($\sum$,E) : est une spécification algébrique telle que :
  - $\sum$ = (S ,Op) Est une signature où :

    - S : est l'ensemble de sortes (types)

    - Op : est l'ensemble d'opérations sur S
  - E : est l'ensemble de $\sum$-équations.
- R : est un uplet tel que R=(P, T, Ω, sort, Cap, IC, DT, CT, TC, I , γ , ICT)
  - P : ensemble fini de places
  - T = $T_{abs} \cup T_{elm}$: ensemble fini de transitions composé de transitions abstraits et transitions élémentaires (P ∩ T) = Ø
  - Ω : Fonction associant le marquage initiale pour chaque transition abstraite
  - *sort* : Fonction associant une sorte de Spec à chaque place du réseau.
  - Cap : C'est pour la capacité des places :

    La capacité associée à une place P précise le nombre de termes que peut contenir cette place pour chaque élément de la sorte de cette place

- IC : Est une fonction de condition d'entrée (Input Condition)
- DT : Fonction qui précise les jetons détruits (Destroyed Tokens)
- CT : Fonction qui précise les termes créés (Created Tokens)
- TC : Fonction qui associe à une transition un terme booléen (Transition Condition)
- I : Ensemble fini des indices appelé indices de terminaison
- γ : Une famille indexée de représentation effective d'ensembles semi-linéaires de marquages finaux.
- ICT : Est une fonction spécifiant les termes crée en places de sortes des transitions abstraites.

Chaque place dans un RECATNet est associée à une sorte et les jetons qui y résident sont des multi-ensembles de termes algébriques [5].

Les arcs reliant les places et les transitions d'un RECATNet sont étiquetés par des multi-ensembles de termes algébriques construits à partir de la signature multi-sorte et de l'ensemble des variables associées à ce réseau. Comme pour les RECATNets ordinaires, dans un RECATNet un arc entrant (P,T) reliant une place P à une transition T (élémentaire ou abstraite) est étiqueté par les deux inscriptions IC(P,T) (Input Conditions) et DT(P,T) (Destroyed Tokens). L'expression IC(P,T) précise les conditions sur le marquage de la place d'entrée P pour le franchissement de la transition T. L'expression DT(P,T) précise les termes détruits de la place P lorsque la transition T est franchie. Une transition T peut-être aussi étiquetée par une inscription de garde TC(T) (Transition Condition). Cependant à la différence des ECATNets ordinaires, les arcs sortant d'une transition dans un RECATNet sont étiquetés par des expressions différentes selon si cette transition est élémentaire ou abstraite.

Pour une transition élémentaire T,un arc sortant (T,P') la reliant à une place P' est étiquetée par l'inscription CT(P',T) (Created Tokens). Par contre, pour une transition abstraite T un arc sortant (T,P') la reliant à une place P' est étiqueté par une expression < i > ICT(P',T,i) (Indexed Created Tokens). Ces deux expressions précisent les jetons à créer dans la place de sortie P'de la transition$t$ [5].

La Figure 2.1 montre une représentation graphique d'une transition élémentaire d'un RECATNets.

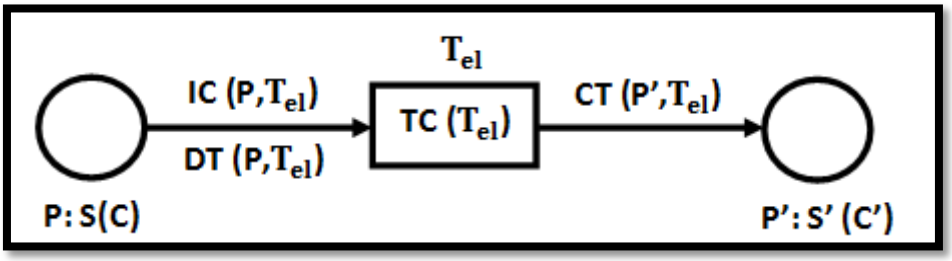


Figure 2.1: Représentation graphique d'une transition élémentaire de RECATNet

La Figure 2.2 montre une représentation graphique d'une transition abstraite d'un RECATNets.

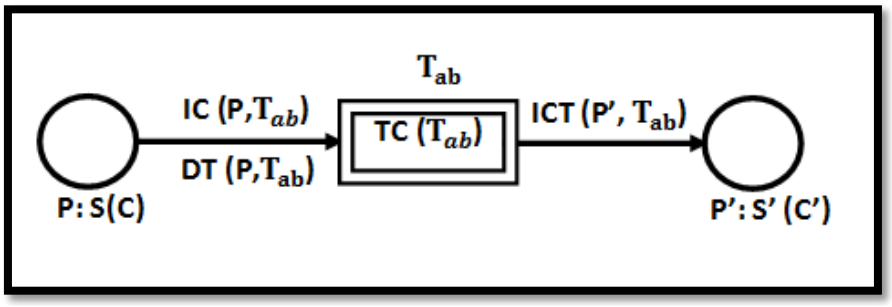


Figure 2.2: Représentation graphique d'une transition abstraite de RECATNet

3. Marquage étendu

Un marquage étendu Tr d'un RECATNet RN=(P , T , Ω , Cap , IC , DT , CT , TC , I , γ , ICT) est un arbre étiqueté Tr = <V, M, E, A >, tel que:

- V est l'ensemble fini des nœuds (processus) de l'arbre. Cette ensemble est vide si Tr = ⊥ (arbre vide).
- M : est l'application qui associe un marquage ordinaire à chaque processus de l'arbre.
- E ⊆ V × V est l'ensemble des arcs de cet arbre,
- A : E → T_{ab} est une application associant une transition abstraite à chaque arc de l'arbre.

Un RECATNet marqué (RN, Tr0) est un RECATNet RN associé à un marquage étendu Tr0 décrivant son état en cours. Pour un marquage étendu Tr, V(Tr) désigne l'ensemble de ses noeuds.

Particulièrement, lorsque l'arbre Tr n'est pas vide, v0(Tr) dénote la racine de l'arbre. Pour chaque noeud $v \in V$, Succ(v) représente l'ensemble de ses nœuds successeurs (directs et indirects) incluant v. De plus, lorsque v n'est pas la racine de l'arbre Tr, Pred(v) désigne son unique nœud prédécesseur (c'est-à-dire, son processus père) [5].

Exemple

Considérons le RECATNet de la figure 2.3 où le terme "ok" est une constante de type $Data$. On assigne une capacité finie à chaque place de ce réseau (soit $cap = 1$). Une séquence de franchissement de ce RECATNet est illustrée dans la figure 2.4. On constate que la transition abstraite t_{start} peut etre franchie indéfiniment créant aussi un nombre infini de processus en largeur dans les marquages étendus accessibles. Ainsi le nombre de marquages étendus accessibles différents est infini. Par conséquent, le RECATNet n'est pas fini bien que toutes les places du réseau soient bornées.

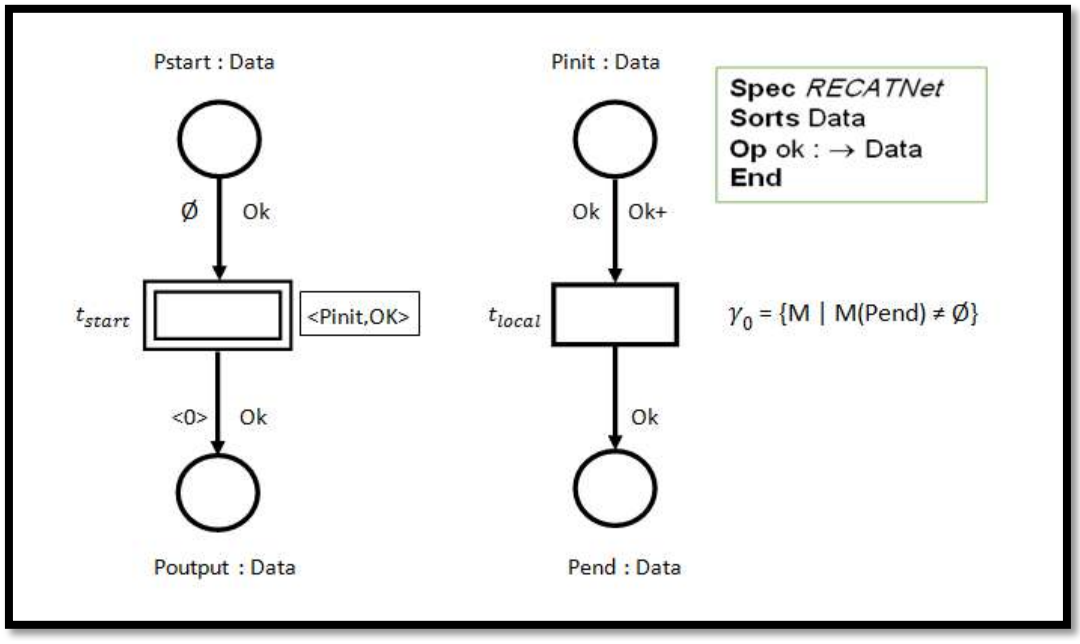


Figure 2.3: Exemple de RECATNet

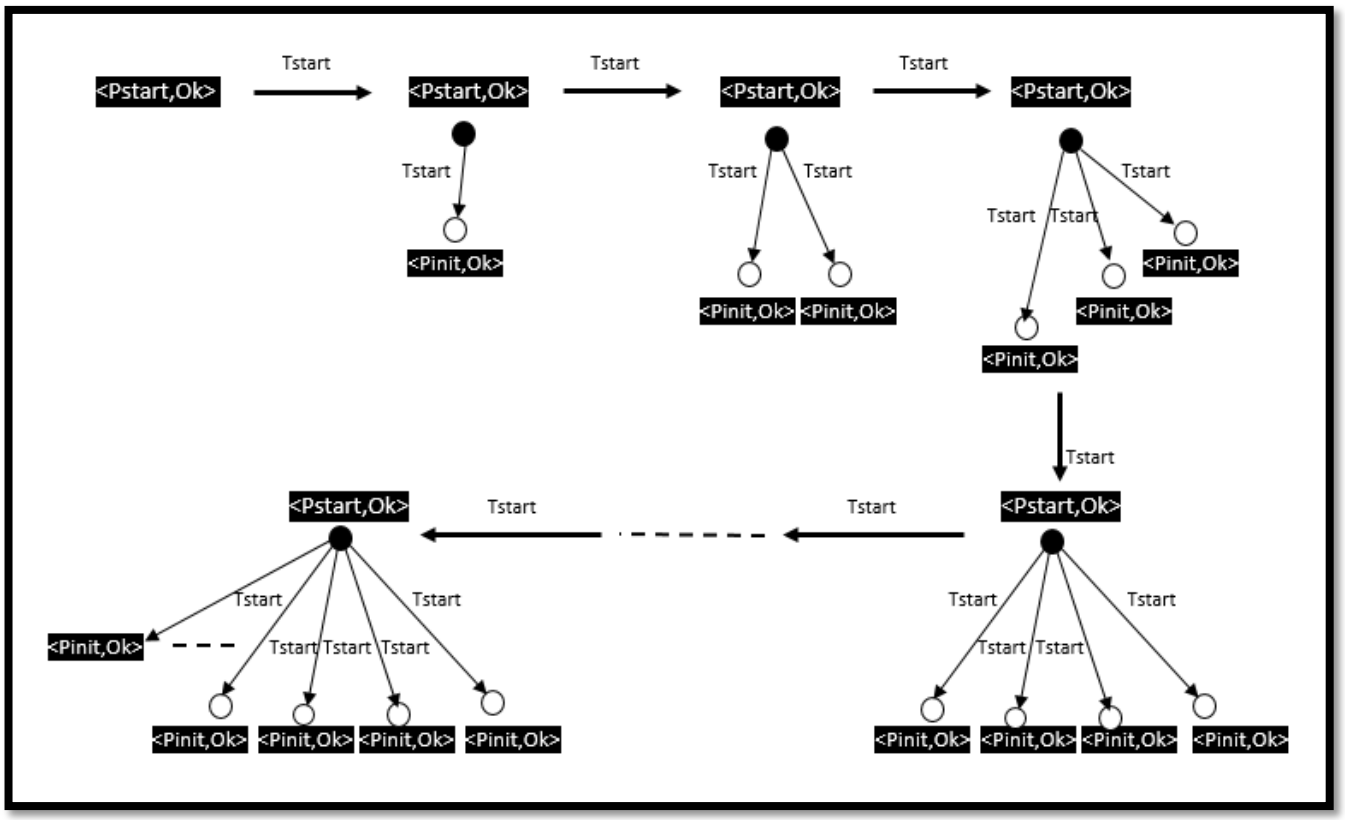


Figure 2.4: Séquence de franchissement de l'exemple précédent

4. Franchissabilité et franchissement

Un pas atomique dans un RECATNet peut être soit le franchissement d'une transition élémentaire ou abstraite ou l'exécution d'une étape de coupure [4].

- ***Franchissement d'une transition élémentaire dans un réseau***

  Le franchissement d'une transition élémentaire T_{el} dans un nœud v d'un marquage étendu Tr = <V, M, E, A > conduit au marquage étendu Tr′ = <V′, M′, E′, A′ > (noté tr —v, T_{el} → tr′) si et seulement si :

  Une transition T_{el} est franchissable dans un réseau v lorsque plusieurs conditions sont satisfaites simultanément: Chaque condition IC(p, T_{el}), pour chaque place d'entrée p de T_{el}, est vérifiée. La condition de transition TC(T_{el}) est vraie.

  - ✓ $V' = V \setminus (\cup v' \in V'' \ Succ(v'))$, $E' = E \cap V' \times V'$,
  - ✓ $\forall e \in E'$, $A'(e) = A(e)$,
  - ✓ $\forall v' \in V' \setminus \{v\}$, $M'(v') = M(v)$,
  - ✓ $\forall p \in P$, $M'(v)(p) = M(v)(p) - DT(p, T_{el}) \oplus CT(p, T_{el})$

- ***Franchissement d'une transition abstraite dans un réseau***

Le franchissement d'une transition abstraite T_{ab} dans un noeud v d'un marquage étendu Tr = <V, M, E, A > conduit au marquage étendu Tr' = <V', M', E', A' > (noté tr —v, T_{ab}→ tr′) si et seulement si :

Une transition abstraite est sensibilisée dans un réseau v lorsque plusieurs conditions sont satisfaites simultanément: Chaque condition IC(p, T_{ab}) est vérifiée pour chaque place d'entrée P de T_{ab}. La conditions de transition TC(T_{ab}) est vraie.

- ✓ Soit v′ un nouvel identifiant dans l'arbre Tr,
- ✓ V′ = V ∪ {v′}, E′ = E ∪ {(v, v′)},
- ✓ ∀ e ∈ E′, A′(e) = A(e), A′((v, v′)) = T_{ab}.
- ✓ ∀ p ∈ P , M ′(v)(p) = M(v)(p) − DT (p, T_{ab}),
- ✓ ∀ p ∈ P , M ′(v ′)(p) = CT(p, T_{ab}).

- ***Franchissement d'une étape de coupure dans un réseau***

Le franchissement d'une étape de coupure τi dans un noeud v d'un marquage étendu Tr = <V, M, E, A > conduit au marquage étendu Tr ' = <V', M', E', A' > (noté Tr—v,τi → Tr′) si et seulement si :

- ✓ M(v) satisfait la condition γ_i
- ✓ Si v est la racine de l'arbre Tr, alors Tr′ = ⊥, autrement :
- ✓ V′ = V \ Succ(v), E′ = E ∩ (V′× V′), ∀e ∈ E′, A′(e) = A(e),
- ✓ ∀ v′ ∈ V′ \ {Pred(v)}, M′(v′) = M(v′),
- ✓ p ∈ P, M′(Pred(v)) (p) = M(Pred(v)) (p) ⊕ ICT (p , A(Pred(v), v), i).

Notons qu'un RECATNet comprend ainsi la notion de franchissement concurrent de pas atomiques (dans les différents processus de son marquage étendu Tr). Au contraire d'un réseau de Petri récursif, qui raisonne en sémantique d'entrelacement, où un pas de franchissement est un pas dans l'un des processus de son marquage étendu [4].

5. Application

Pour illustrer la pertinence des RECATNets dans la modélisation de processus workflows collaboratifs et reconfigurables nous prenons l'exemple d'un processus simplifié de réservation en ligne de voyage (Figure 2.5). Une agence de voyage offre à ses clients la capacité de réserver en ligne des circuits de vacances complets: billets d'avion/train/autobus, hôtels, location de voitures, excursions, etc. Les prestataires de services (des compagnies aériennes, des sociétés d'autobus, des chaînes d'hôtel,…, etc.) fournissent des services Web pour vérifier les disponibilités de leurs offres et faire les réservations [5].

Le processus de réservation est initialisé lors de l'enregistrement d'une nouvelle demande d'un client représentée par un jeton (*Num, listBooking*) comprenant le numéro de la demande et la liste des réservations souhaitées (exp. Billets de trains, avions, hôtel...etc.). Notons que l'état initial de ce workflow est un arbre de processus réduit à un unique processus racine avec un jeton (*Num, listBooking*) dans la place d'entrée *Start*.

L'exécution de la tâche abstraite *BookingProcess* initialise le sous-processus de traitement de la demande de réservation en créant dynamiquement un nouveau processus fils dans l'arbre des processus du RECATNet avec l'état initial associé à cette transition abstraite. Dans ce sous-processus, l'exécution de la tâche abstraite *MakeBooking* pour chaque élément de la liste des réservations souhaitées (pour chaque couple (*Trip, Rq*) de la liste *listBooking*) crée, dynamiquement, un autre sous-processus. Chacun de ces sousprocessus nouvellement créés correspond à l'invocation du service mis en ligne par le prestataire de service (compagnie aérienne ou autre) pour vérifier la disponibilité de la demande (représentée par le couple (*Trip, Rq*)) et faire la réservation. Notons que le nombre d'instances en cours des services invoqués dépend du nombre des éléments de la liste des requêtes du client. Ce nombre n'est donc pas connu durant la phase de conception du workflow [5].

La synchronisation des instances crées est spécifiée par les marquages finaux¡. Par exemple, la terminaison d'un de ces services est indiquée par la présence d'un jeton dans la place *EndBooking* (voir les marquages finaux γ_2 et γ_3) [5].

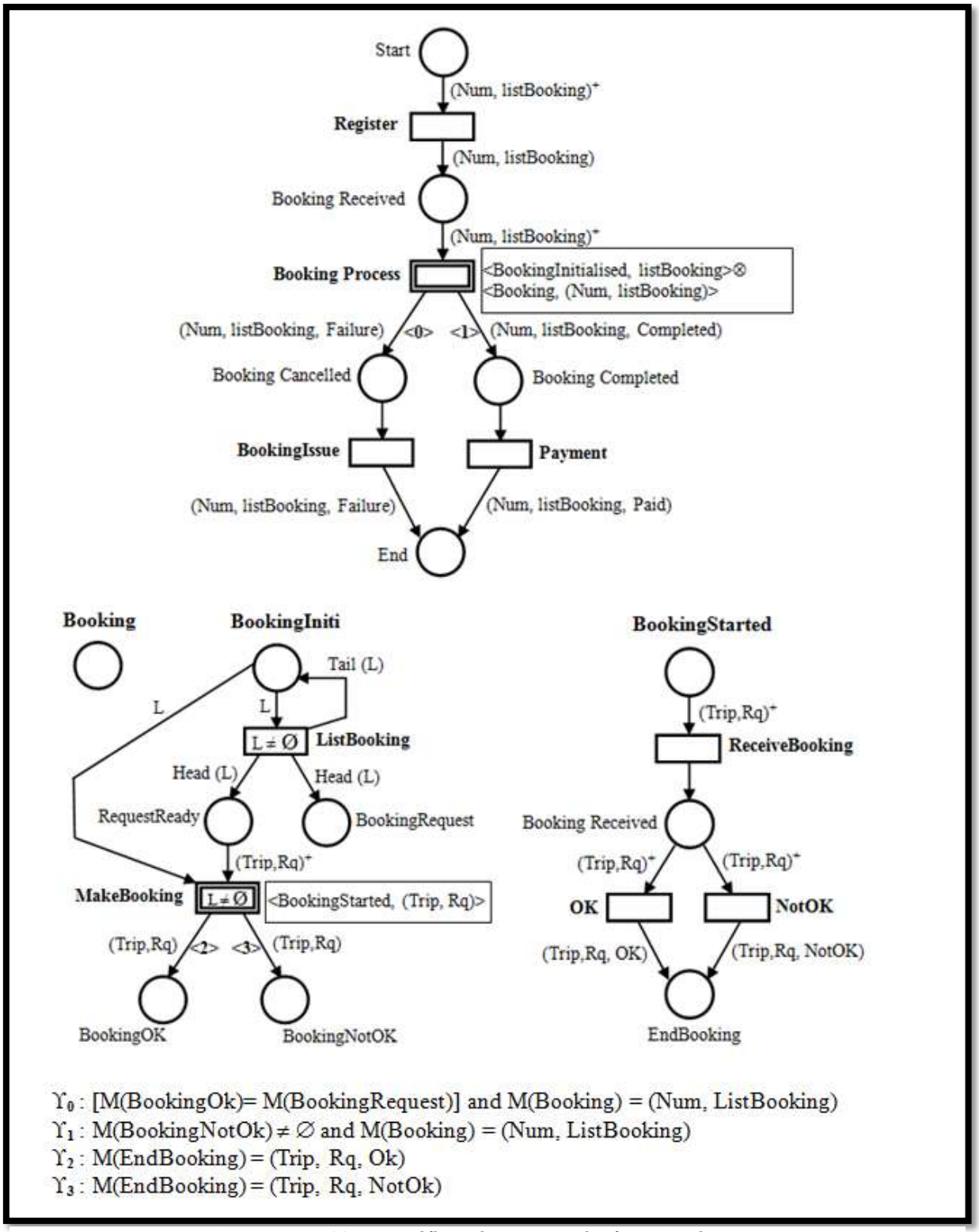


Figure 2.5 : RECATNet modélisant le Processus de réservation de voyage

6. Conclusion

Dans ce chapitre nous avons introduit un modèle qui s'appelle ECATNets Récursifs (RECATNets). Les RECATNets sont une classe de réseaux de Petri algébriques de haut niveau définis sur la base d'une combinaison saine des ECATNets classiques (Extended Concurrent Algebraic Term Nets) et des réseaux de Petri récursifs (RdPRs). Ils ont le pouvoir des spécifications algébriques et un mécanisme récursif pratique permettant une modélisation directe de la création et de la suppression dynamique de processus concurrents et leur synchronisation. De ce fait, ils conviennent à la modélisation compacte de tout système concurrent caractérisé par une structure de contrôle dynamique ainsi que la manipulation de structures de données complexes. Du fait de leur grand pouvoir de description, la propriété d'accessibilité n'est plus décidable pour les RECATNets.

Chapitre 3: Langage BPMN (Business Process Model and Notation)

1. Introduction

Business Process Modeling est une partie importante de la compréhension et de la restructuration des activités et des informations d'une entreprise typique utilise pour atteindre ses objectifs d'affaires. ***BPMN*** est la notation standard la plus adaptée pour modéliser, documenter et restructurer les processus et les flux d'informations. La spécification ***BPMN 2.0*** a été publiée par l'*OMG* (Object Management Group) [6].

2. Historique

À l'origine BPMN est un projet initié par Business Process Management Initiative (BPMI) qui fusionna en 2005 avec l'Object Management Group (*OMG*). L'*OMG* est une association américaine à but non lucratif créée en 1989, dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes. L'*OMG* est un organisme influant dans l'industrie de conception logiciel et est notamment à la base des standards *UML* (Unified Modelling Language), *MOF* (Meta-Object Facility) ou *CORBA* (Common Object Request Broker Architecture). Elle est aussi à l'origine de la recommandation *MDA* (Model Driven Architecture) développant les principes de l'ingénierie dirigée par les modèles (*IDM* que nous avons abordé au chapitre précédent) [7].

3. Définition

BPMN (Business Process Modeling Notation) est une notation graphique standardisée « Éléments graphiques et diagrammes » pour modéliser des processus d'entreprise. Son objectif est de fournir des notations compréhensibles directement par tous les utilisateurs au niveau d'une entreprise [6].

BPMN est un langage récent conçu pour être au cœur d'une approche de modélisation et d'implémentation utilisant les cadres conceptuels de l'ingénierie dirigée par les modèles (*IDM*), de l'architecture orientée services (*SOA19*) et de la gestion des processus métier (*BPM20*) [7].

La notation ***BPMN*** définit un ensemble d'objets graphiques qui permettent la description des processus. En effet, cette notation se basée sur : Événements, Activités, Branchements [6].

Les éléments présentés par la Figure 3.1 constituent la base de la modélisation avec BPMN. Chacun de ces symboles est ensuite déclinés en un certains nombres de variantes précisant leur sens.

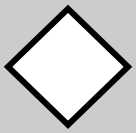
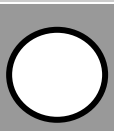
Nom	Symbole	Brève description
Activités		Réalisation d'une action par un utilisateur ou un système informatique
Branchements (Gateway)		Gestion des embranchements du flux d'activité. Associer, dissocier, dédoubler les chemins.
Évènement (event)		Quelque chose se produisant au cours du processus et ayant un impact sur lui (ex: Départ, arrêt, pause.)

Figure 3.1 : Les éléments d'un BPMN

4. les éléments de base de BPMN

➢ Événements

Un événement représente un fait se produisant durant l'exécution du processus, Un événement permet de marquer l'impact, sur le déroulement d'un processus, d'un phénomène interne ou externe au processus. On peut distinguer les événements de début, les événements d'émission de message qui provoquent l'interruption du processus, et les événements de fin [6].

Les évènements quant à eux sont toujours représentés de manière sphérique et sont déclinés de la même manière via des représentations graphiques additionnelles à l'intérieur du cercle. Cependant, on distingue plusieurs sortes d'événement qui sont présentés dans la Figure 3.2 [6].

Nom	Symbole	Brève description
Début		ce sont des évènements déclencheurs du processus. Ils peuvent se déclencher sans autre ou à réception d'un évènement particulier. Les évènements de départ reçoivent et déclenchent un processus mais n'émettent pas.
Intermédiaire		Permette d'indiquer l'attente d'un type d'évènement au milieu d'un flux. Par exemple, tant que l'évènement « réception d'un message » est en attente, le flux est stoppé. Les évènements intermédiaires peuvent recevoir ou émettre un évènement selon le cas.
Fin		Ils terminent un processus et peuvent émettre une information. Ils sont symbolisés par un trait extérieur épais. Les événements de fin émettent et terminent un processus mais ne reçoivent rien.
Timer		l'exécution du processus est retardé jusqu'à un certain point dans le temps est atteinte ou une durée particulière est terminée.
Message		Cet événement réagit à l'arrivée d'un message.
Erreur		Attrape une erreur nommée, qui a été jeté soit une portée intérieure. Cet événement doit être fixé à la limite d'une activité.

Figure 3.2 : les différents types d'évènements

Les éléments de la Figure 3.2 se sont de deux types :

- Les évènements de base (Début, Intermédiaire, Fin).
- Les évènements secondaires (Timer, Message, Erreur….)

➢ Les Branchements :

Un branchement est un élément de modélisation utilisé pour contrôler des enchaînements qui convergent ou qui divergent à l'intérieur d'un processus [7].

❖ Les branchements en sortie d'une étape :

En sortie d'une étape, un branchement représente un point de divergence des enchaînements en sortie d'un processus.

- Dans le cas d'un branchement Exclusif, seule une branche sortante peut être choisie parmi les différents possibles.
- Dans le cas d'un branchement Parallèle, l'ensemble des branches sortantes est traité simultanément.

La Figure 3.3 montre une représentation graphique des branchements en sortie d'une étape.

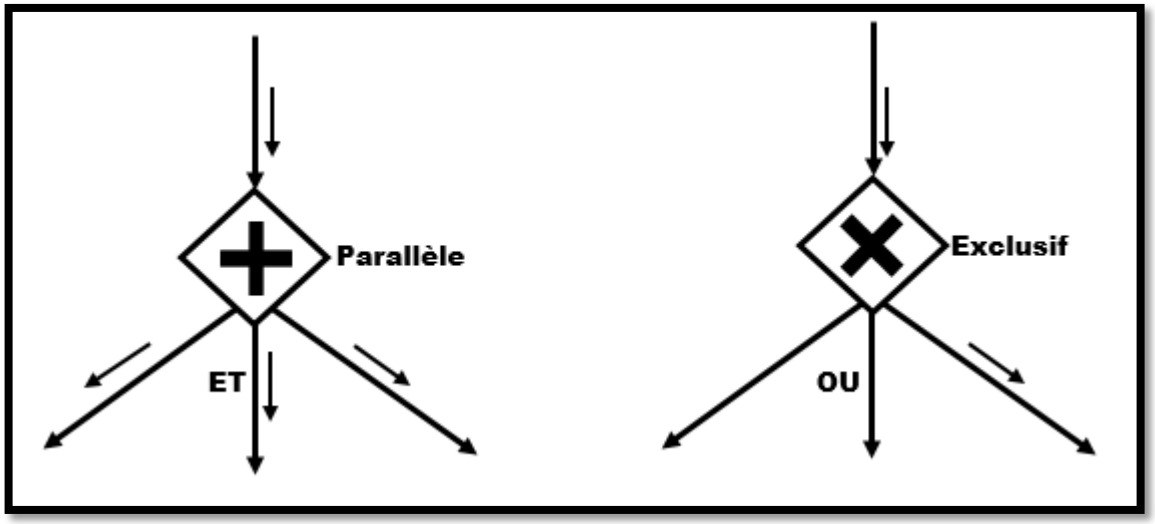


Figure 3.3 : les branchements en sortie d'une étape

❖ **Les branchements en entrée d'une étape :**

En entrée d'une étape, un branchement représente un point de convergence des enchaînements en entrée d'un processus.

- Dans le cas d'un branchement Exclusif, l'étape du processus est déclenchée dès qu'une des branches est active.
- Dans le cas d'un branchement Parallèle, l'ensemble des branches entrantes est traité simultanément.

La Figure 3.4 montre une représentation graphique des branchements en entrée d'une étape.

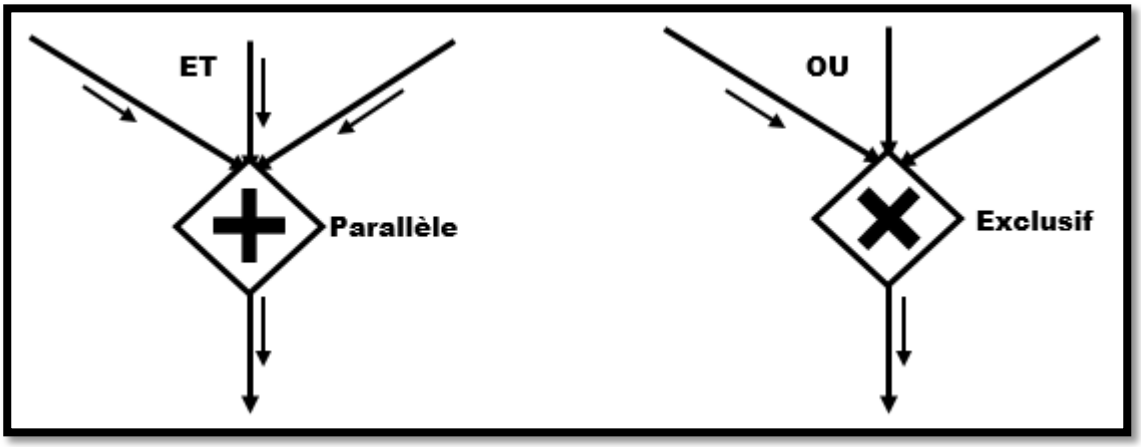


Figure 3.4 : les branchements en entrée d'une étape

➢ Les activités:

Une activité est un terme générique pour un travail réalisé par une compagnie, dans le cadre d'un processus métier particulier. Une activité peut être atomique ou non atomique (composée) [7].

Nom	Symbole	Brève description
Tâche		Une tâche est une étape élémentaire qui est inclus dans un processus. Une tâche est utilisée quand une étape d'un processus n'est plus décomposée en un niveau de description plus fin.
Sous-processus		Un sous-processus est composé d'un ensemble de tâches se réalisant indépendamment du processus principal. Un sous-processus est un mécanisme permettant de regrouper un ensemble de tâches ayant une certaine corrélation afin de simplifier la lecture d'un diagramme de processus.
Activité D'appel		Une activité D'appel est une tâche utilisée pour appeler un deuxième processus lors de l'exécution du processus en cours. Cette fonctionnalité permet, par exemple, de remplacer le processus appelé par un autre processus sans perturber la description du processus principal.

Figure 3.5 : Les différents types d'activités

➢ **Éléments de connexion :**

Nom	Symbole	Brève description
Flux de séquence (Sequence Flow)		Représente le cours du processus. Permet de lier les éléments du tableau précédents.
Flux de message (Message flow)		Représente le flux des communications. Élément lié à un message.
Association		Aucun lien avec le cours du processus. Ce symbole sert à lier des informations additionnelles facultatives à des éléments du processus.

Figure 3.6 : Les différents types de Flux

5. Définition formelle

Un processus BPMN de base est un tuple ***B = {O, F,Cond,Excp}***

- O est un ensemble d'objets qui peuvent être partitionné en ensembles disjoints d'activités ***A***, événements ***E*** et branchements ***G***

- ***A*** peut-être partitionné en ensembles disjoints de tâches (Tasks) ***T*** et des activités d'appel de sous-processus (subprocess invocation activities) ***S***.
- ***E*** peut être partitionné en ensembles disjoints des événements de début (Start Event) E^s, événements intermédiaires (intermediate events) E^i et les événements finaux (end events) E^e.
- E^i peut être partitionné en ensembles disjoints de messages d'événements intermédiaires (intermediate message events) E^i_M, les événements d'horloge intermédiaire (intermediate timer events) E^i_T et les événements d'erreur intermédiaires (intermediate error events) E^i_R.
- E^e peut être partitionné en ensembles disjoints d'événements non-erreur finaux (end non-error events) E^e_N et les événements d'erreur finaux (end error events) E^e_R.

- G peut être partitionné en ensembles disjoints de parallèle fork branchement (parallel fork gateways) G^F, parallèle join branchement (parallel join gateways) G^J, XOR décision branchement de base de données (data-based XOR decision gateways) G^D et XOR fusionnement des branchements (XOR merge gateways) G^M.

- $F \subseteq O \times O$ est le contrôle de relation de flux
- Cond : $F \cap (G^D \times O) \rightarrow C$ est une fonction qui mappe la séquence des flux émanant de XOR branchements sur la base de données à des conditions
- Excp : $E^i \rightarrow A$ est une fonction qui affecte un événement intermédiaire à une activité

6. Exemple 1 : Modèle BPMN pour une commande de pizza

La figure 3.7 montre un diagramme ***BPMN*** pour une commande de pizza par un client. Le client va choisir un plat de pizza puis il va le commander, si le plat n'est pas prêt alors il va attendre 20 min et demander son pizza, sinon, il va recevoir le plat, payer et en fin manger.

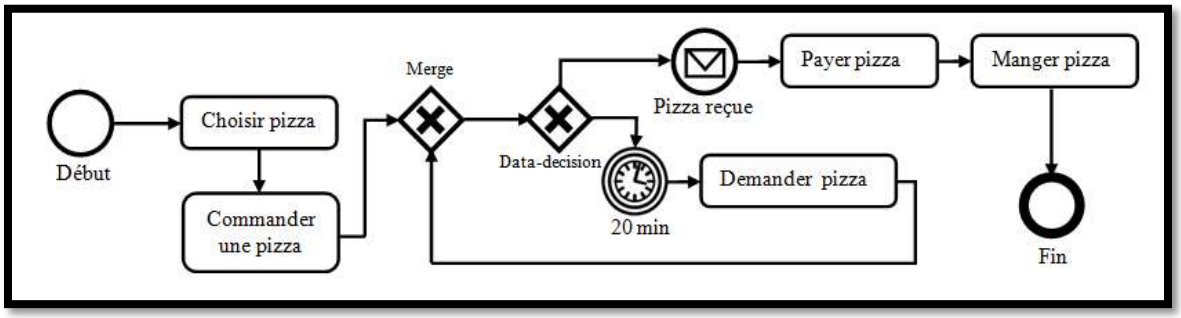


Figure 3.7 : Exemple d'un modèle BPMN pour une commande de pizza

7. Exemple 2 : Modèle BPMN pour une réservation de voyage

La figure 3.8 montre un diagramme ***BPMN*** pour les étapes de réservation d'un voyage, d'abord le client va inscrire dans le site de l'agence de voyage, ensuite il va réserver l'avion avec l'hôtel et choisir entre le bus et l'automobile, en fin il envoi les informations de la réservation. S'il y a une erreur dans la réservation d'avion ou bien d'hôtel alors le sous-processus va se terminer et exécute la tâche 'Erreur de Réservation' et termine.

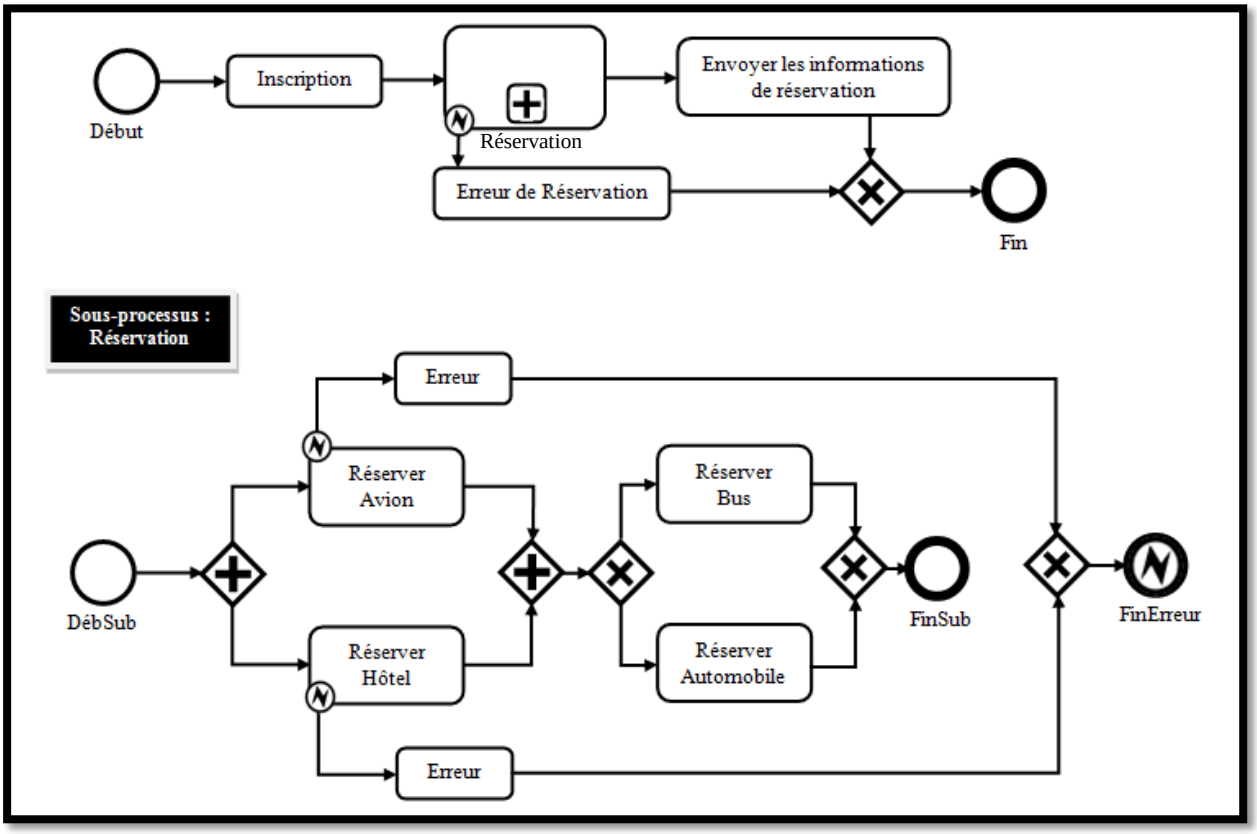


Figure 3.8 : Exemple d'un modèle BPMN pour la réservation de voyage

8. Exemple 3 : Modèle BPMN pour une réclamation d'assurance

La figure 3.9 représente un modèle BPMN pour les étapes de réclamation d'assurance d'un accident de voiture.

D'abord, il faut ramener les témoins qui ont été présents dans l'accident pour vérifier les témoignages. Ensuite obtenir les déclarations de ces derniers, si le nombre de déclarations obtenus deviennent 4 déclarations alors compléter l'obtention des déclarations. En fin on dépose la réclamation d'assurance.

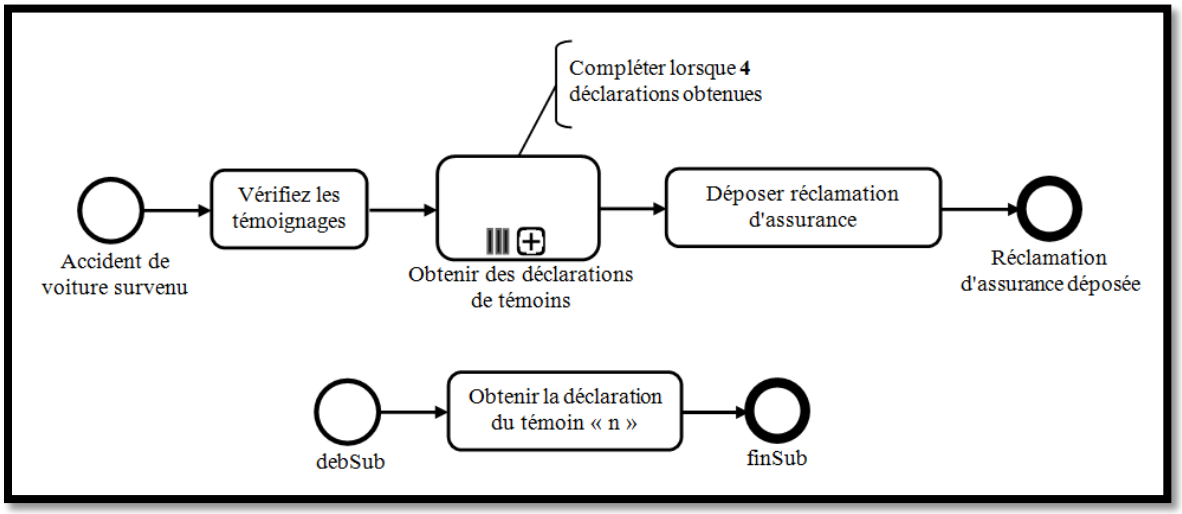


Figure 3.9 : Processus BPMN pour la réclamation d'assurance d'un accident de voiture

9. Outils utilisés pour la création de BPMN

Les outils utilisés pour la création des modèles BPMN sont :

(1) Eclipse BPMN2 Modeler : est un outil de modélisation graphique pour la création de processus métiers. Son objectif était de fournir un cadre d'édition de workflow graphique, qui peut être facilement personnalisé.

(2) Bizagi Modeler : est un processus puissant pour le modeleur BPMN compatible avec le standard BPMN 2.0, conçu pour transformer, modeler et schématiser tous les types de flux de travail.

10. Conclusion :

Dans ce chapitre nous avons présenté les concepts de bases sur ***BPMN*** qui permet de créer un mécanisme simple pour modéliser la complexité inhérente à un processus. ***BPMN*** fournit une notation commune de sorte que les personnes liées à des processus peuvent les exprimer graphiquement d'une manière claire, standardisée et complète.

Chapitre 4 : Transformation BPMN vers RECATNets

1. Introduction

Une transformation est une opération qui prend en entrée des modèles (source) et fournit en sortie des modèles (cibles), généralement un seul modèle source et un seul modèle cible. Ce chapitre présente la transformation de langage BPMN vers RECATNets.

2. Les Méta-modèles utilisés pour la transformation

Nous avons travaillé avec deux méta-modèles, le méta-modèle pour BPMN et l'autre pour RECATNet.

2.1 Méta-modèle BPMN :

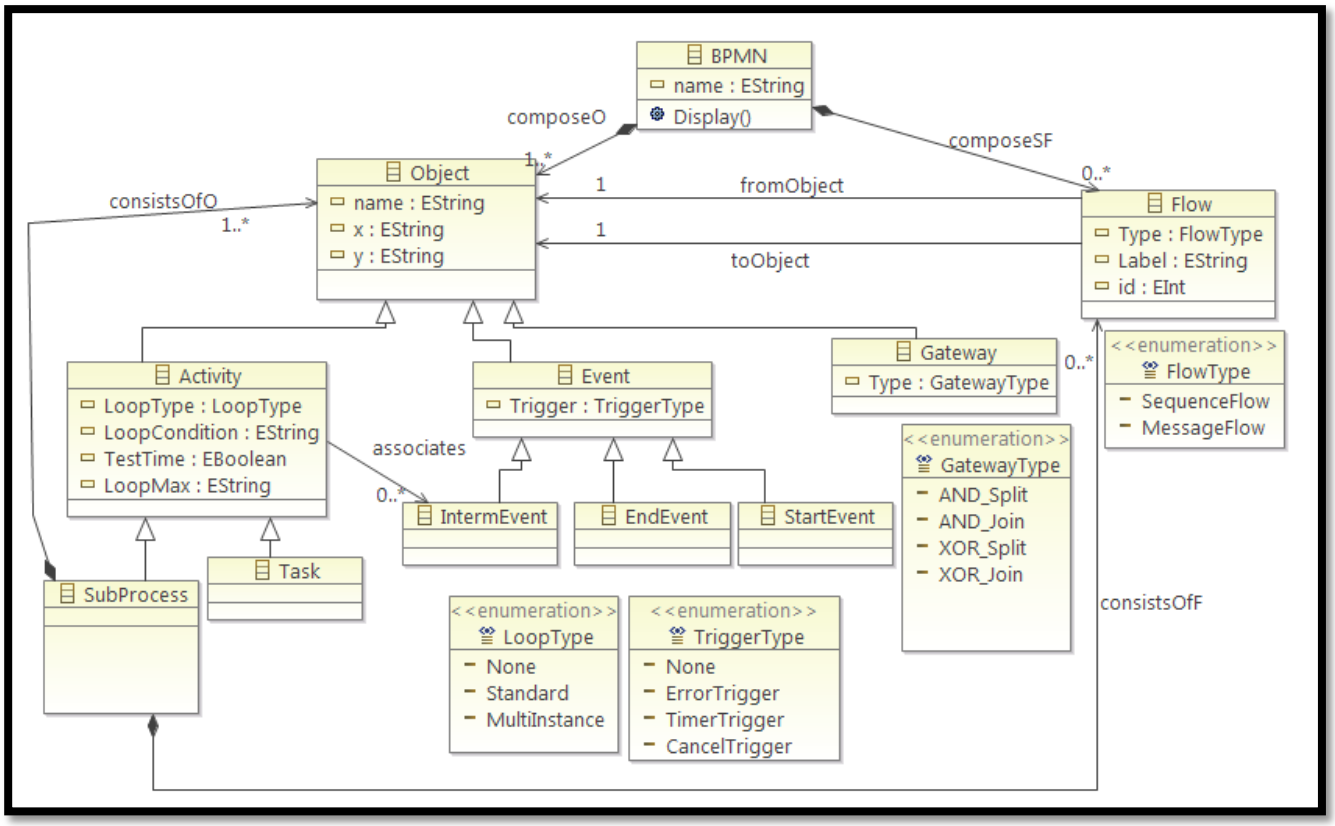


Figure 4.1: Méta-modèle de BPMN

Dans la figure 4.1, qui représente le méta-modèle de BPMN, nous avons défini tous les éléments nécessaires pour construire un modèle BPMN, le processus BPMN contient deux classes principales.

- La classe "**Object**" qui est hérité par trois classes :
  - **Gateway** (Branchements) qui peut être de type AND-Split, AND-Join, XOR-Split ou XOR-Join,
  - **Event** (Evènements) qui peut être soit StartEvent, IntermEvent ou bien EndEvent, cette classe a un attribut nommé *Trigger* qui veut dire le type de l'évènement (soit None (rien), ErrorTrigger, TimerTrigger ou CancelTrigger).
  - **Activity** (Activité) qui peut être une tâche (**task**) ou un sous-processus (**subProcess**), les attributs de cette classe sont :
    - **loopType :** C'est le type de la boucle (soit None (il n'y a aucune boucle), Standard ou multiInstance).
    - **loopCondition :** C'est la condition de la boucle.
    - **testTime :** Si testTime=true (before) alors on travail avec la boucle (while-do), sinon (after) on travail avec la boucle (do-until).
    - **loopMax :** C'est le nombre maximum d'instances de la boucle.

  La classe **subProcess** (sou-processus) est composé d'autre Flows et Objects.

- La classe "**Flow**" (c'est-à-dire flux) a deux relations avec les objets (fromObject et toObject), cette classe fait la relation entre les objets de BPMN (la relation de l'objet précédent « fromObject » avec l'objet suivant « toObject »). L'attribut **FlowType** veut dire qu'il y a deux types de flux, soit un flux de séquence ou un flux de message.

2.2 Méta-modèle RECATNet :

Dans la figure 4.2, qui représente le méta-modèle de RECATNet, nous avons défini tous les éléments nécessaires pour construire un modèle RECATNet.

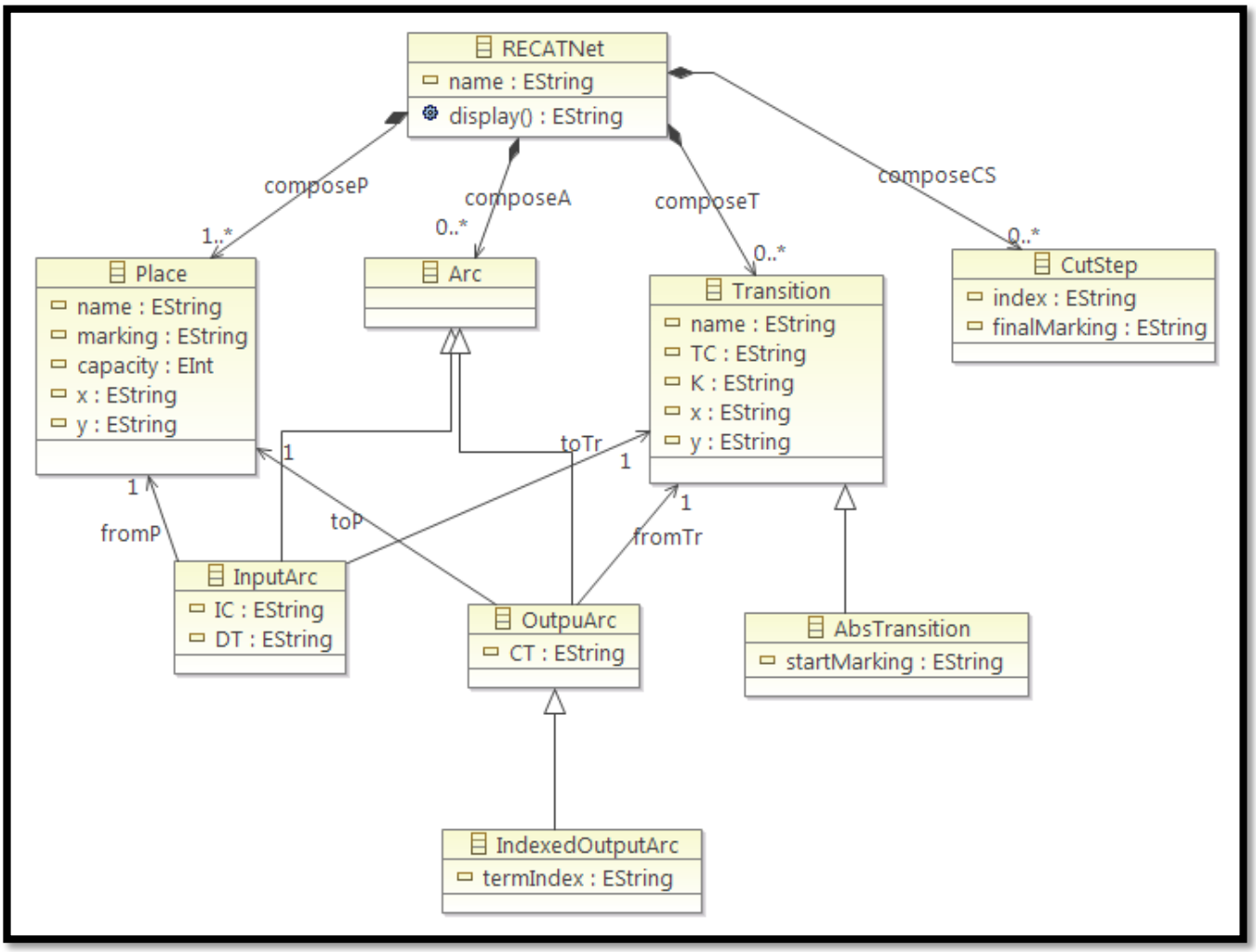


Figure 4.2 : Méta-modèle de RECATNet

Le processus RECATNet contient quatre classes principales.

- La classe "**Place**" : Représente les places. Les attributs *x* et *y* représentent les coordonnés de la place dans le graphe résultat de ce projet.

- La classe "**Transition**" : Représente les transitions, elle est héritée par une autre classe qui est **AbsTransition** qui représente les transitions abstraites et qui prend comme attribut (startMarking). Les attributs *x* et *y* représentent les coordonnés de la place dans le graphe résultat de ce projet.

- La classe "**Arc**" : Représente la relation entre les places et les transitions. Cette classe est spécialisée dans deux classes :
    - **InputArc :** Cet arc va sortir d'une place vers une transition, cette classe a deux relation : *fromP* avec la classe **Place** et *toTr* avec la classe **Transition**.
    - **OutpuArc :** Cet arc va sortir d'une transition vers une place, cette classe a deux relation : *fromTr* avec la classe **Transition** et *toP* avec la classe **Place**. La classe **IndexedOutputArc** fait la relation entre le **CutStep** et **OutpuArc.**

- La classe "**CutStep**" : C'est une coupure qui se fait à la fin d'un processus appelé par une transition abstraite (**AbsTransition)**, elle contient l'attribut *index* qui est de la forme γ_n telque *n* est le même avec l'attribut *termIndex* de la classe **IndexedOutputArc.** L'attribut *finalMarking* représente le marquage final qui est nécessaire pour réaliser la coupure.

3. Les règles de transformation :

Afin de simplifier la formalisation des règles de transformation, nous définissons les fonctions suivantes :

- ✓ in(x) : Cette fonction « in(x) = {y ∈ O | (y, x) ∈ F} » retourne les entrées des objets BPMN d'un objet x de BPMN.
- ✓ out(x) : Cette fonction « out(x) = {y ∈ O | (x, y) ∈ F} » retourne les soties des objets BPMN d'un objet x de BPMN.
- ✓ endSubProc(*S*) : Cette fonction retourne l'ensemble des événements de fin dans un sous-processus BPMN associée à l'activité d'invocation S.
- ✓ *startSubProc*(*S*) : Cette fonction retourne l'événement de début dans un sous-processus BPMN associée à l'activité d'invocation S.
- ✓ getTermIndex() : Cette fonction retourne un nombre naturel représentent l'indice de terminaison.
- ✓ Excp(excp) : Cette fonction retourne le nom du subprocess dont lequel l'évènement **excp** (boundary event) produisant une exception est associé.

Dans la suite, nous allons présenter nos règles de transformation des éléments du BPMN en module de RECATNet.

3.1. Les règles de transformation des éléments de base :

3.1.1. Règle N°1 : transformation de flux de séquence :

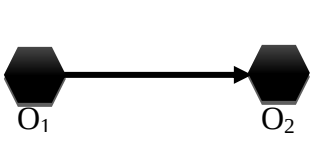
BPMN	RECATNet
	
$(O_1,O_2) \in F$	$P = P \cup \{P_{O1-O2}\}$

Figure 4.3: Transformation de flux de séquance

La figure 4.3 montre la transformation d'un flux de séquence de BPMN, qui est représenté par une flèche entre deux objets O_1 et O_2. Alors, la translation d'un flux de séquence en RECATNet est une place P_{O1-O2} ayant un nom, qui est la concaténation des noms des objets O_1 et O_2.

3.1.2. Règle N°2 : Les évènements (Events)

- **L'évènement de début (Start Event) :**

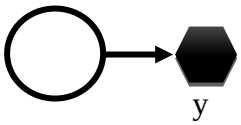
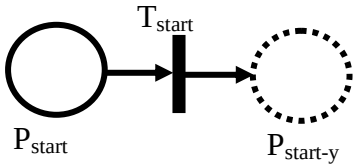
BPMN	RECATNet
	
Start event start $\in E^{st}$	$P = P \cup \{P_{start}\}$, $T_{elt} = T_{elt} \cup \{T_{start}\}$ $IC = IC \cup \{(P_{start}, T_{start}, token)\}$ $CT = CT \cup \{(T_{start}, P_{start-y}, token) \mid y \in out(start)\}$

Figure 4.4 : Transformation d'Evènement de début (Start Event)

La figure 4.4 montre un évènement de début « Start Event » indique qu'un processus particulier est en train de commencer, l'objet '*y*' est pour définir un objet quelconque pour présenter l'objet suivant de 'Start Event'. Alors on transforme cet objet vers une place (P_{start}) avec une transition élémentaire (T_{start}) et la place '$P_{start-y}$' est pour confirmer qu'il existe obligatoirement une place suivante qui présente le 'y' de BPMN.

- **L'évènement intermédiaire (Intermediate Event) :**

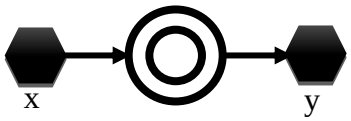
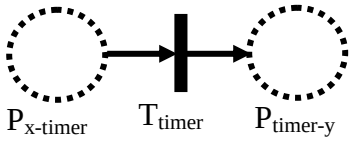
BPMN	RECATNet
	
Intermediate event timer ∈ E^i	$T_{elt} = T_{elt} \cup \{ T_{timer} \mid in(timer) \neq \emptyset \}$, $IC = \{(P_{x\text{-}timer}, T_{timer}, token) \mid x \in in(timer)\}$, $CT = CT \cup \{(T_{timer}, P_{timer\text{-}y}, token) \mid y \in out(timer)\}$

Figure 4.5 : Transformation d'Evènement intermédiaire (Intermediate Event)

La figure 4.5 montre un évènement intermédiaire « Intermediate Event », par exemple un 'timer', les objets O_1 et O_2 sont des objets quelconques de BPMN dans le but de présenter les objets précédant et suivant de cet évènement. Alors on transforme cet objet vers une transition élémentaire, le but des places ($P_{x\text{-}timer}$) et ($P_{timer\text{-}y}$) est de confirmer qu'il existe obligatoirement une place entrante et autre sortante.

- **L'évènement de fin (End Event) :**

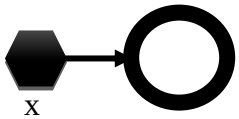
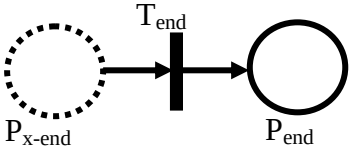
BPMN	RECATNet
	
End event end ∈ E^e	$P = P \cup \{P_{end}\}$, $T_{elt} = T_{elt} \cup \{T_{end}\}$, $IC = IC \cup \{(P_{x\text{-}end}, T_{end}, token) \mid x \in in(end)\}$, $CT = CT \cup \{(T_{end}, P_{end}, token)\}$, $\gamma = \gamma \cup \{(P_{end} > 0, getTermIndex()) \mid \exists s \in S : P_{end} \in endSubProc(s)\}$

Figure 4.6 : Transformation d'Evènement de fin (End Event)

La figure 4.6 montre un évènement de fin « End Event » qui indique la fin d'un flux du processus et le '*x*' indique qu'il existe un objet précédent. Alors elle est transformée vers une transition élémentaire (T_{end}) avec une seule place (P_{end}) et le but de la place '$P_{x\text{-}end}$' est de confirmer qu'il existe obligatoirement une place entrante.

3.1.3. Règle N°3 : Les branchements (Gateways)

- **Branchement Fork (AND-Splite):**

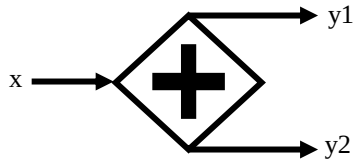
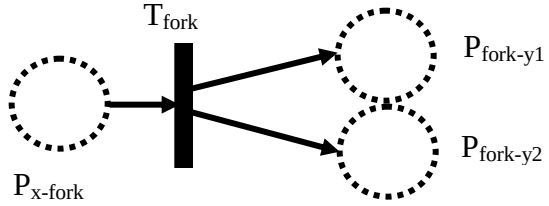
BPMN	RECATNet
	
fork ∈ G^P	$T_{elt} = T_{elt} \cup \{T_{fork}\}$, $IC = IC \cup \{(P_{x-fork}, T_{fork}, token) \mid x \in in(fork)\}$, $CT = CT \cup \{(T_{fork}, P_{fork-y}, token) \mid y \in out(fork)\}$

Figure 4.7 : Transformation de fork

La figure 4.7 montre un branchement fork parallèle connu sous le nom « AND-Split », permettant de diviser un flux de séquence '*x*' en deux ou plusieurs flux '*y*' qui peuvent être exécutées en parallèle dans le processus. Alors Il est transformé vers une transition élémentaire (T_{fork}), la place d'entrée (P_{x-fork}) est pour présenter le flux entrant et l'ensemble des places de sortie (P_{fork-y}) présentent l'ensemble des flux sortants.

- **Branchement Join (AND-Join):**

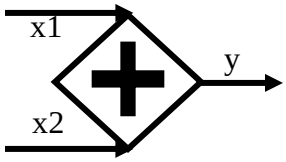
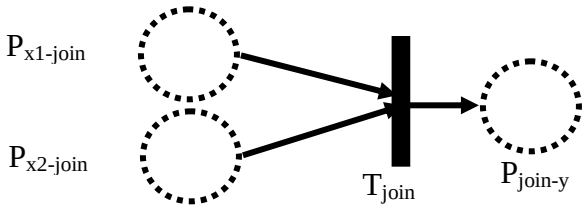
BPMN	RECATNet
	
Join ∈ G^J	$T_{elt} = T_{elt} \cup \{T_{join}\}$, $IC = IC \cup \{(P_{x-join}, T_{join}, token) \mid x \in in(join)\}$, $CT = CT \cup \{(T_{join}, P_{join-y}, token) \mid y \in out(join)\}$

Figure 4.8 : Transformation de Join

La figure 4.8 montre un branchement Join parallèle connu sous le nom « AND-Join », permettant de combiner deux ou plusieurs flux de séquence parallèles (*x1*, *x2*…) vers un seul flux de séquence (*y*).

Alors Il est transformé vers une transition élémentaire (T_{join}), l'ensemble des places d'entrée ($P_{x\text{-}join}$) sont pour présenter les flux entrants et la place de sortie ($P_{join\text{-}y}$) est pour présenter le flux sortant.

- **Branchement de décision (XOR-Split):**

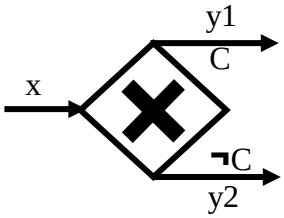
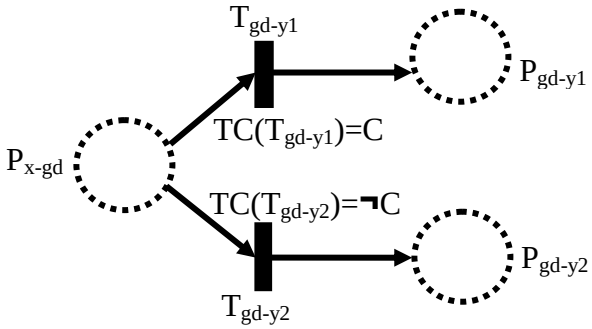
BPMN	RECATNet
	
data-decision gd ∈ G^D	$T_{elt} = T_{elt} \cup \{ T_{gd\text{-}y} \mid y \in out(gd)\}$, $TC = TC \cup \{(T_{gd\text{-}y}, Cond(gd,y)) \mid y \in out(gd)\}$, $IC = IC \cup (P_{x\text{-}gd}, T_{gd\text{-}y}, token) \mid y \in out(gd) \wedge x \in in(gd)\}$, $CT = CT \cup \{T_{gd\text{-}y}, P_{gd\text{-}y}, token) \mid y \in out(gd)\}$

Figure 4.9 : Transformation de "Data-decision"

La figure 4.9 montre un branchement de décision divergent (Data-decision) connu sous le nom « XOR-Split », il est utilisé pour créer des chemins alternatifs au sein d'un flux de processus. Le '*x*' présente le chemin entrant et les y (y1, y2..) présentent les chemins de sortie qui ont des conditions à vérifier pour utiliser le chemin, Alors Il est transformé vers un ensemble des transitions élémentaires ($T_{gd\text{-}y}$) et chaque transition a une place sortante avec une place d'entrée ($P_{x\text{-}gd}$) qui présente le flux entrant '*x*'. La place ($P_{gd\text{-}y}$) est pour présenter les flux conditionné. Chaque transition $T_{gd\text{-}y}$ on associe une condition de franchissement (TC).

- **Branchement Merge (XOR-Join):**

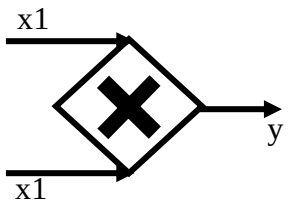
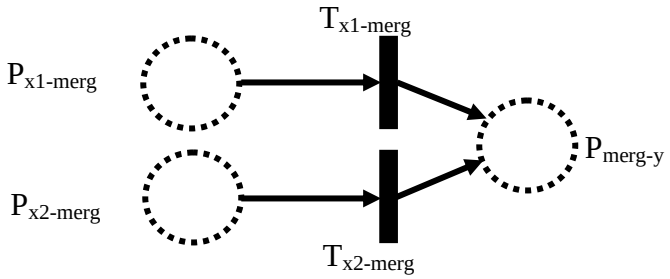
BPMN	RECATNet
	
merge ∈ G^M	$T_{elt} = T_{elt} \cup \{T_{x\text{-}merg} \mid x \in in(merge)\}$, $IC = IC \cup \{(P_{x\text{-}merg}, T_{x\text{-}merg}, token) \mid x \in in(merge)\}$, $CT = CT \cup \{T(_{x\text{-}merg}, P_{merg\text{-}y}, token) \mid x \in in(merge) \wedge y \in out(merge)\}$

Figure 4.10 : Transformation de fusion (Merge)

La figure 4.10 montre un branchement de fusion (Merge Gateway), connu sous le nom « XOR-Join », permettant de combiner deux ou plusieurs flux de séquence parallèles (*x1,x2..*) en un seul flux de séquence *y*. mais pas de synchronisation est nécessaire car aucune activité parallèle ne fonctionne au point de jonction (Join Point). Alors il est transformé vers un ensemble des transitions élémentaires ($T_{x\text{-}merge}$), avec une place de sortie ($P_{merge\text{-}y}$), chaque transition a une place entrante pour présenter les flux entrantes de BPMN.

3.1.4. Règle N°4 : Les taches (Tasks)

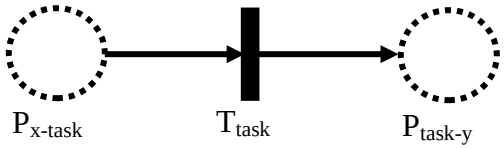
BPMN	RECATNet
	
task ∈ T	$T_{elt} = T_{elt} \cup \{T_{task}\}$, $IC = IC \cup \{(P_{x\text{-}task}, T_{task}, token) \mid x \in in(task)\}$, $CT = CT \cup \{(T_{task}, P_{task\text{-}y}, token) \mid y \in out(task)\}$

Figure 4.11 : Transformation de tâche (Task)

La figure 4.11 montre une tâche (Task) qui est transformée vers une transition élémentaire (T_{task}), avec une place d'entrée ($P_{x\text{-}task}$) et une place de sortie ($P_{task\text{-}y}$).

3.1.5. Exemple 1 : RECATNet pour une commande de pizza

La figure 4.12 représente la transformation de l'exemple 1 (modèle BPMN) du chapitre 3 (la Figure 3.7) vers un modèle RECATNet en appliquant les règles de transformation présentées précédemment.

D'abord, le client choisit une pizza et fait son commande, ensuite si elle est prête alors il la mange et paie, sinon il va attendre jusqu'à ce qu'elle est prête.

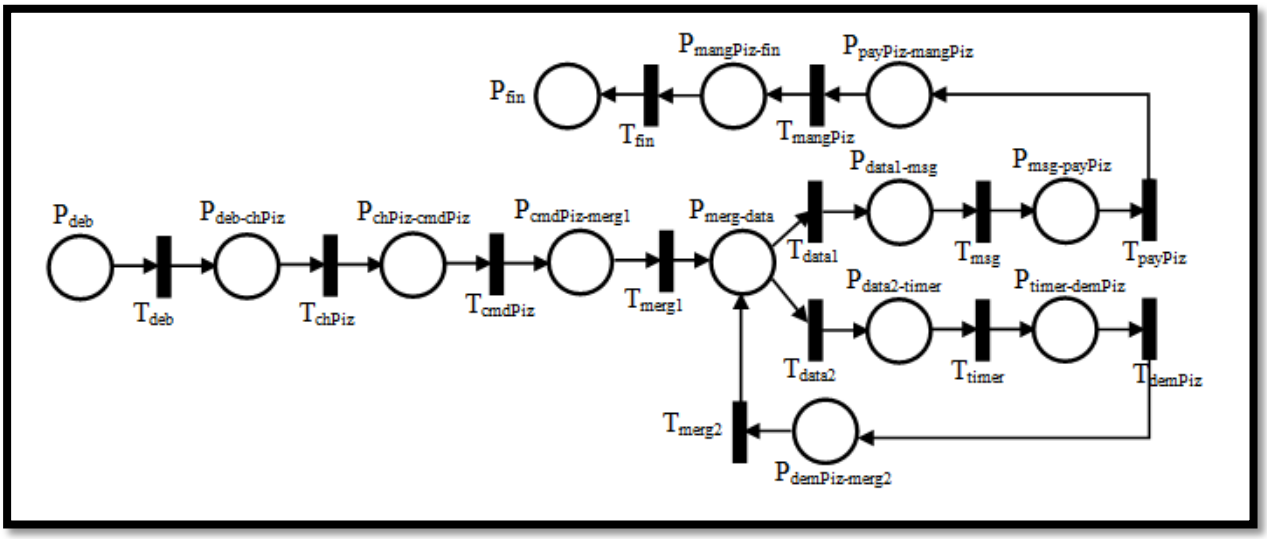


Figure 4.12 : RECATNet du processus de commande de pizza

3.2. Les règles de transformation des éléments complexes :

3.2.1. Règle N°5: Les sous-processus (SubProcess) :

Un sous-processus peut être considéré comme un processus BPMN indépendant. Dans la figure 4.13, nous décrivons l'application d'appeler un sous-processus via une activité d'invocation de sous-processus.

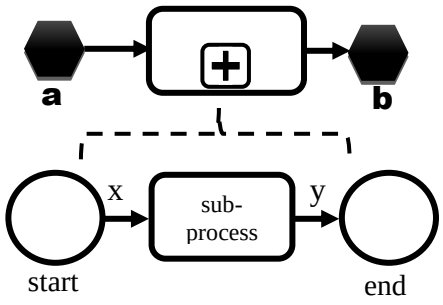
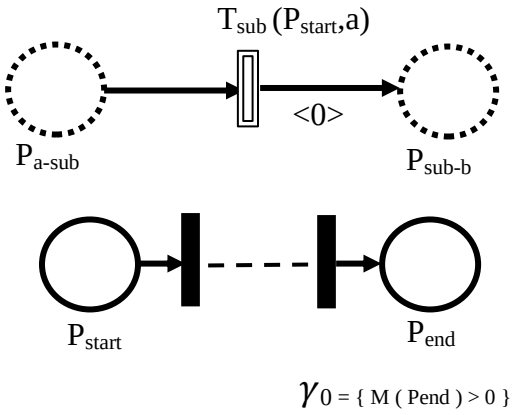
<table>
<tr><th>BPMN</th><th>RECATNet</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>subprocess sub ∈ S</td><td>$T_{abs} = T_{abs} \cup \{T_{sub}\}$,
$\Omega = \Omega \cup \{(T_{sub}, P_{start}, token) | start = startSubProc(sub)\}$,
$IC = IC \cup \{(P_{a\text{-}sub}, T_{sub}, token) \mid a \in in(Sub)\}$,
$ICT = ICT \cup \{(T_{sub}, P_{sub\text{-}b}, index, token) \mid b \in out(sub) \wedge$
$\exists\ cs\ (P_{end}, index) \in \gamma\ end \in endSubProc(sub) \wedge end \in E^{e}_{N}\}$</td></tr>
</table>

Figure 4.13 : Transformation de sous-processus (subProcess)

La place « P_{start} » représente l'évènement de début (Start Event), et la place « P_{end} » représente l'évènement de fin (End Event) du sous-processus invoqué. En outre, un ensemble semi-linéaire de marques finales est définie en indiquant les flux de fin d'un sous-processus. Par exemple, l'ensemble $\gamma_0 = \{M\ (_{Pend}) > 0\}$ indique qu'un sous-processus se termine lorsque la place Pend est marquée. Le sous-processus est transformé vers une transition abstraite (T_{sub}) où il commence le marquage $\Omega(T_{sub}) = (P_{start}, x)$. Le franchissement de la transition abstraite (T_{sub}) affine la transition par un nouveau sous-réseau (c'est-à-dire : création d'un nouveau thread, nommé son enfant) qui commence ses propres jetons x en place (P_{start}). Une fois un marquage final est atteint, selon l'ensemble des marquages finals semi-linéaires γ_0 défini ci-dessus, une étape de coupe ferme le sous-réseau correspondant, et produit des jetons dans la place de sortie approprié ($P_{sub\text{-}b}$) de la transition abstraite T_{sub}.

3.2.2. Règle N°6 : Les boucles (Loops) :

L'activité du bouclage contient les deux boucles « While-do » et « do-Until » en fonction de leur attribut 'TestTime'. En fait, si ***TestTime = Before*** respectivement ***TestTime = After*** signifie que la condition de la boucle est évaluée *au début* respectivement *à la fin* de l'itération de boucle.

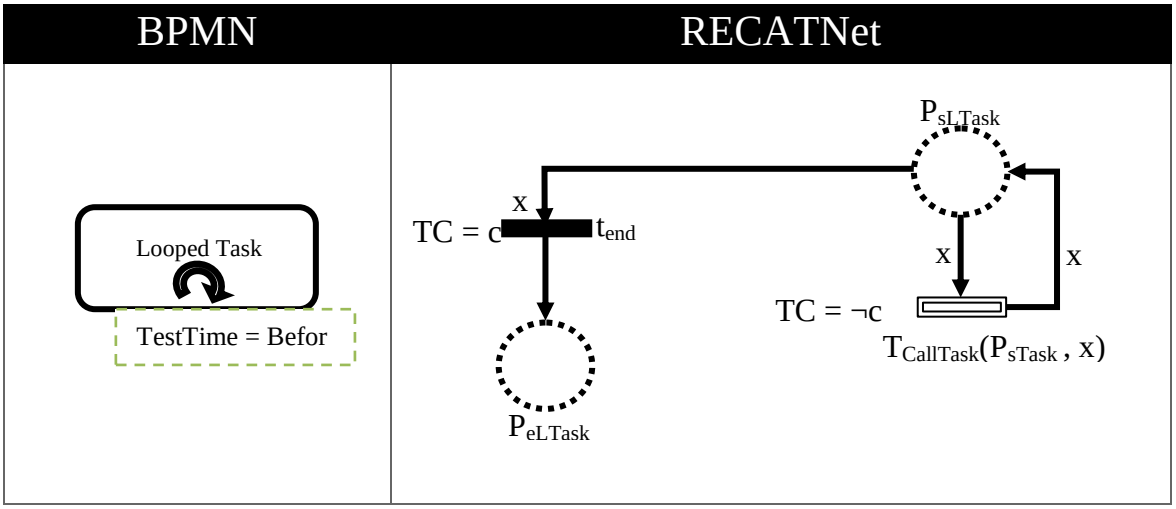


Figure 4.14 : Transformation de la boucle "while-do"

La figure 4.14 montre la transformation de l'activité de la boucle « While-do » dans 'RECATNet' tandis que la figure 4.15 montre la transformation de l'activité de la boucle « do-Until » dans 'RECATNet'. Dans les RECATNets obtenus, lorsque la transition abstraite « tCallTask » est tiré, elle appelle le sous-réseau associé à la tâche bouclé.

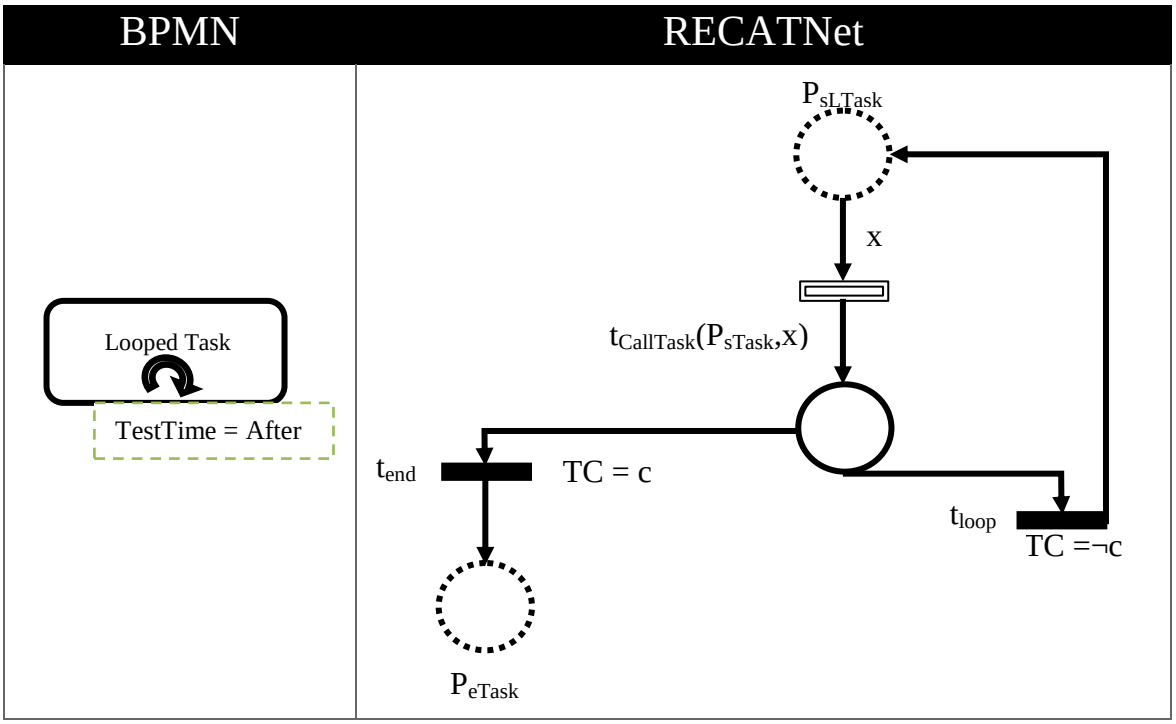


Figure 4.15 : Transformation de la boucle "do-until"

3.2.3. Règle N°7 : L'Activité multi-instances :

L'activité multi-instances parallèle permet de créer un ensemble d'instances des activités où ils sont exécutés en parallèle. Dans la figure 4.16, on montre une cartographie de l'activité multi-instance parallèle dans le cas statique. Dans ce cas on doit définir le nombre d'instances par le nombre naturel (n), on transforme cette activité vers une transition abstraite ($T_{MultiTask}$) qu'elle va appeler un autre réseau qui est responsable au démarrage des processus, et contient deux place ($P_{sCall-A}$), ($P_{eCall-A}$) et une transition abstraite (T_{Call-A}). La place '$P_{sCall-A}$' représente la place de début (Start Place) et contient "***n***" jetons. L'appel de la transition reste active jusqu'à la consommation de tous les jetons.

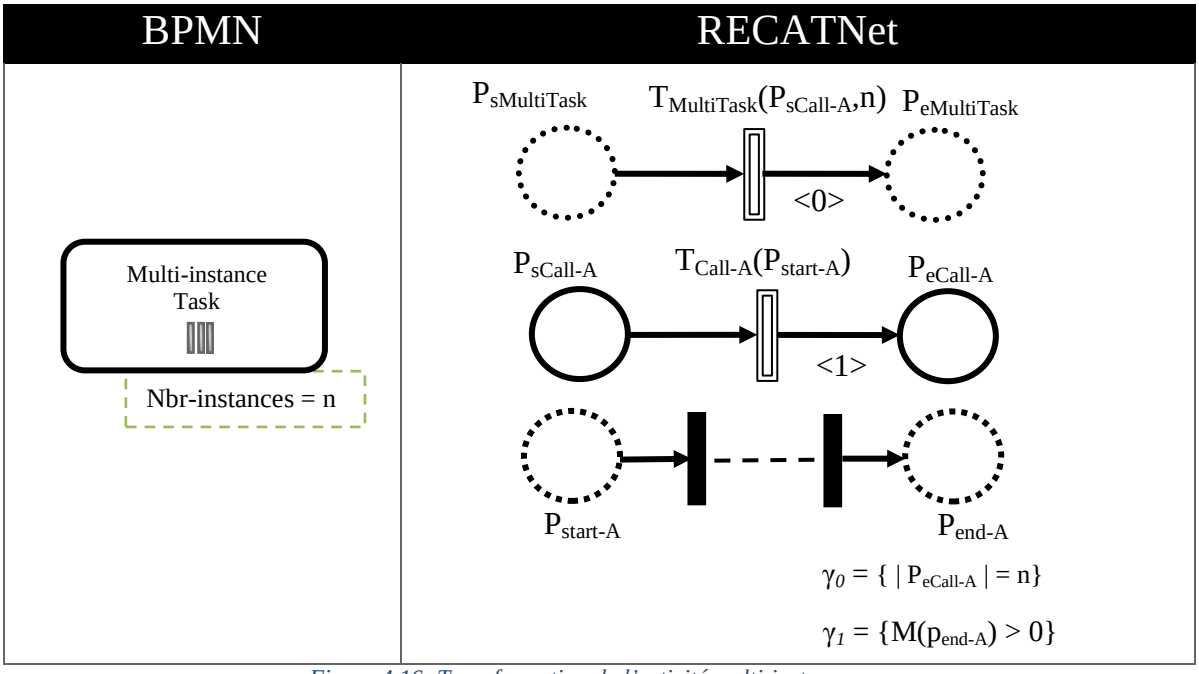


Figure 4.16: Transformation de l'activité multi-instance

3.2.4. Règle N°8 : Gestion des exceptions (Exception Handling) :

En BPMN, la gestion des exceptions est capturée par flux d'exception. Un flux d'exception provient d'un événement d'exception attachée à la limite d'une activité, qui est soit une tâche ou un sous-processus.

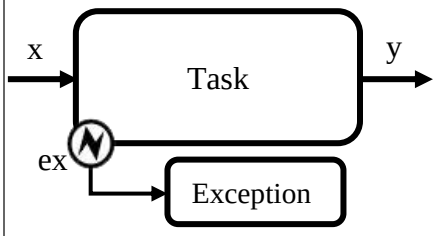
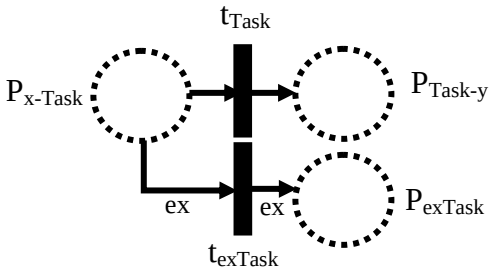
BPMN	RECATNet
	
Exception @ task excp ∈ E^i_R @ T	Telt=Telt∪{t_{exTask}|excp∈ E^i_R ∧ in(excp)=Ø ∧ ∃y∈T y=Excp(excp)}, IC = IC ∪{($p_{x\text{-}Task}$, t_{exTask}, exception)|y ∈ T ∧ x ∈ in(y) ∧ excp ∈ E^i_R ∧ y = Excp(excp)}, CT = CT ∪ {(t_{exTask}, p_{exTask}, exception)|excp ∈ E^i_R ∧ ∃x ∈ T x = Excp(excp) ∧ y ∈ out(excp)}

Figure 4.17 : Transformation de l'exception d'une tâche

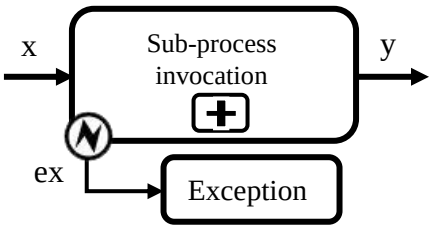
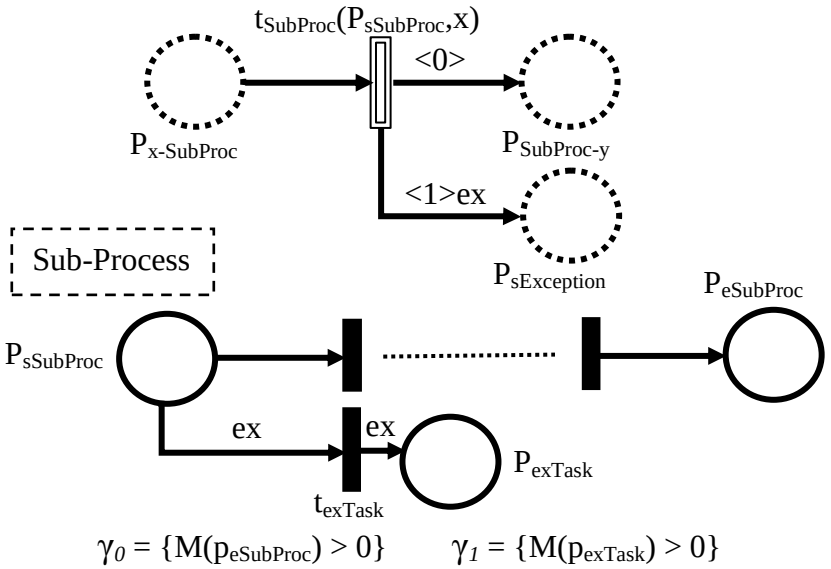
BPMN	RECATNet
	
Exception @subprocess excp ∈ E^i_R @S	ICT = ICT ∪ {($t_{subProc}$,$p_{sException}$, index, exception)|y∈S ∧ excp ∈ E^i_R ∧ y = Excp(excp) ∧ z ∈ out(excp) ∧ ∃ cs($p_{eSubProc}$,index)∈Y ∧ end ∈ endSubProc(excp)} ∧ end ∈ E^e_R }

Figure 4.18 : Transformation de l'exception d'un sous-processus

La figure 4.17 montre la cartographie d'une exception associée avec une tâche et la figure 4.18 montre la cartographie d'une exception associée avec une un sous-processus. Le RECATNet associé à un sous-processus montre la transformation d'une exception associée à une activité atomique (*Task*). L'apparition de l'exception ne peut interrompre que le flux normal au moment où il est prêt à exécuter la tâche. Par conséquent, la transformation d'une exception associée à une tâche est une transition élémentaire t_{exTask} où son franchissement évitera d'exécuter l'activité atomique '*Task*' et génère un jeton représente le type d'exception en place P_{exTask}.

Pour la manipulation d'exception associée à un sous-processus, l'occurrence de l'événement d'exception va interrompre l'exécution du flux normal à l'intérieur du sous-processus, lorsqu'elle est active (en cours d'exécution). Cette cartographie est simple dans RECATNet. Tout d'abord, un ensemble semi-linéaire de marquages finaux est défini, il indique l'occurrence d'une telle exception dans un sous-processus.

Par exemple, comme représenté dans la Figure 4.18, l'ensemble semi-linéaire de marques finaux $\gamma_1 = \{M\ (_{PexTask}) > 0\}$ indique l'occurrence d'une exception associée à l'activité atomique *Task*. Alors, le sous-processus peut se terminer par deux types de terminaisons. La première façon est lorsque le sous-processus se termine correctement en marquant la place $P_{eSubProc}$, qui doit définir l'ensemble des terminaisons γ_0.

La deuxième façon de terminaison est lorsque le sous-processus se termine sur l'occurrence d'une exception, c'est-à-dire marquer la place p_{exTask} qui doit définir l'ensemble de terminaison γ_1. Deuxièmement, nous avons besoin d'ajouter une autre place de sortie pour la transition abstraite, qui modélise l'invocation du sous-processus pour capturer l'exception. Par exemple, comme représenté dans la figure 4.18, la transition abstraite $t_{SubProc}$, qui peut invoquer le sous-processus, a une deuxième place de sortie $p_{sException}$, avec un indice de terminaison <1>. Cela signifie que, lorsque le sous-processus atteint un marquage final en γ_1, une étape de coupe interrompt le sous-processus et produit des jetons dans la place de sortie $p_{sException}$ associée à l'indice de terminaison <1>.

3.2.5. Exemple 2 : RECATNet pour une réservation de voyage

La figure 4.19 représente la transformation de l'exemple 2 (modèle BPMN) du chapitre 3 (la Figure 3.8) vers un modèle RECATNet en appliquant les règles de transformation présentées précédemment.

D'abord, le voyageur doit s'inscrire, ensuite fait la réservation. Dans la réservation nous avons utilisé une transition abstraite parce que nous avons travaillé avec un sous-processus dans le modèle BPMN. La transition abstraite « $T_{subReserv}$ » va appeler le sous-processus ci-dessus pour faire la réservation, le voyageur doit réserver l'avion avec l'hôtel et choisir entre la réservation du bus et de la voiture. S'il y a une erreur dans la réservation de l'avion ou bien l'hôtel alors le sous-processus se termine avec la place $P_{finErreur}$, sinon il va terminer normalement.

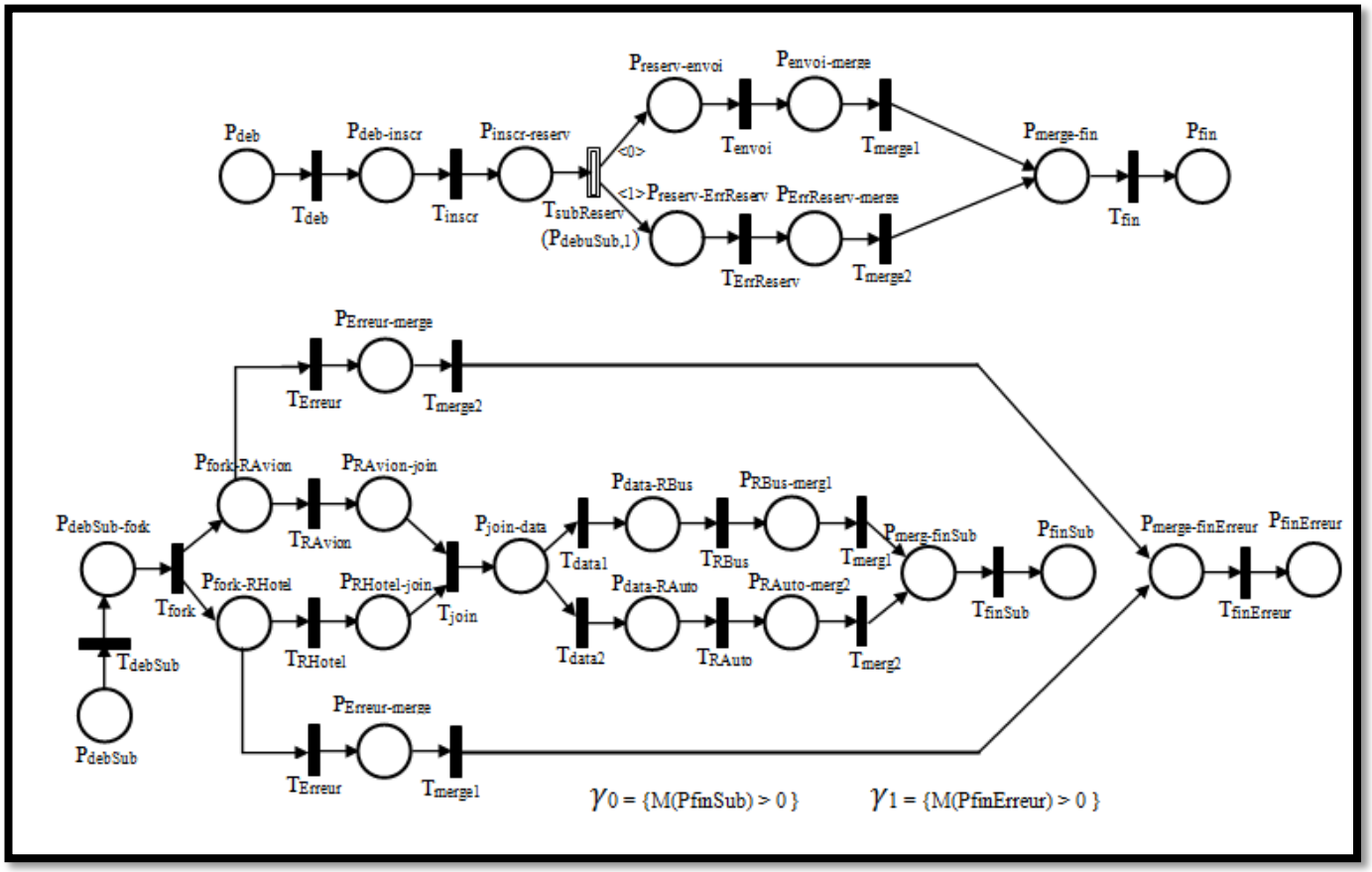


Figure 4.19 : RECATNet du processus de réservation de voyage.

3.2.6. Exemple 3 : RECATNet pour une réclamation d'assurance d'un accident

La figure 4.20 représente la transformation de l'exemple 3 (modèle BPMN) du chapitre 3 (la Figure 3.9) vers un modèle RECATNet en appliquant les règles de transformation présenté précédemment.

D'abord, vérifier les témoins, ensuite nous avons une transition abstraite pour l'obtention des déclarations des témoins, cette transition fait appel à un sous-processus qui à une autre transition abstraite qui va s'appeler 4 fois. Une fois le nombre d'instance devient 4 alors le sous-processus termine. En fin déposer la réclamation.

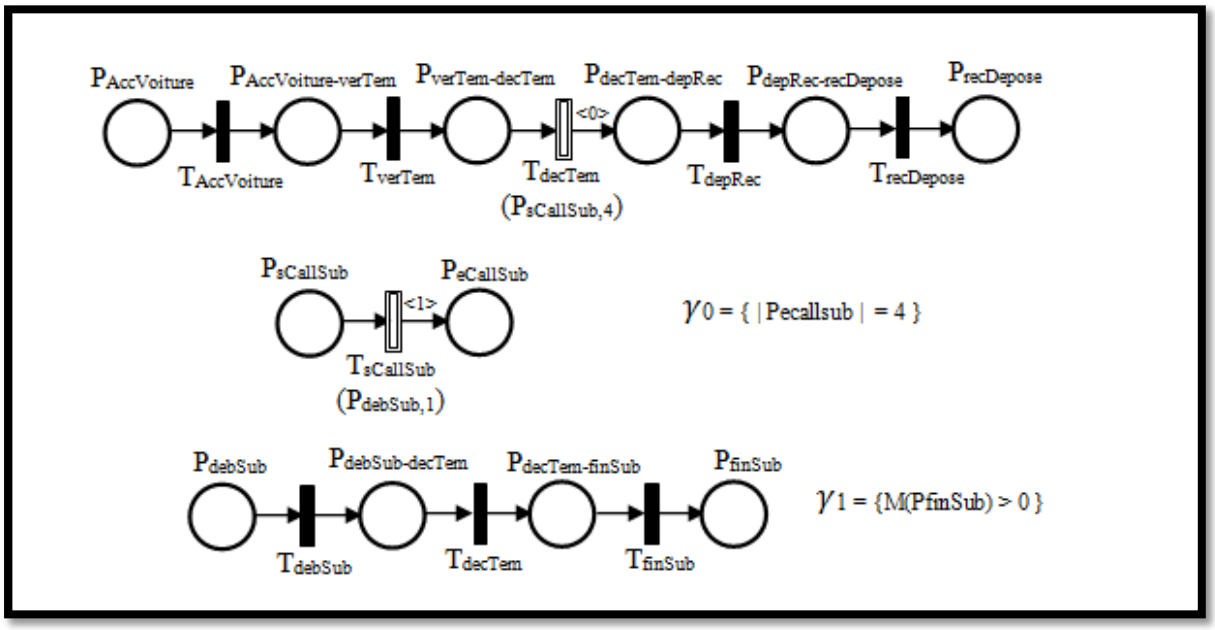


Figure 4.20 : RECATNet du processus de réclamation d'assurance d'un accident

4. Conclusion :

Dans ce chapitre, nous avons défini d'abord les méta-modèles utilisés pour la transformation (méta-modèle de BPMN et méta-modèle de RECATNet), ensuite nous avons défini une méthode permettant la transformation d'un sous-ensemble de BPMN vers RECATNets sur la base des règles de correspondance pour lesquelles une sémantique formelle est établie.

Avec cette application, nous pouvons couvrir un ensemble large de fonctionnalités de BPMN comme l'instanciation multiple du sous-processus et la manipulation d'exception tout en prenant en compte l'aspect de flux de données.

Chapitre 5 : Implémentation de l'approche de transformation

1. Introduction

Dans ce chapitre nous allons implémenter l'approche de la transformation de BPMN vers RECATNet en présentant un schéma globale pour la transformation avec un Parser JAVA ensuite avec ATL. Ensuite nous allons faire une implémentation pour les règles ATL programmé dans l'interface eclipse.

2. Schéma globale pour la transformation :

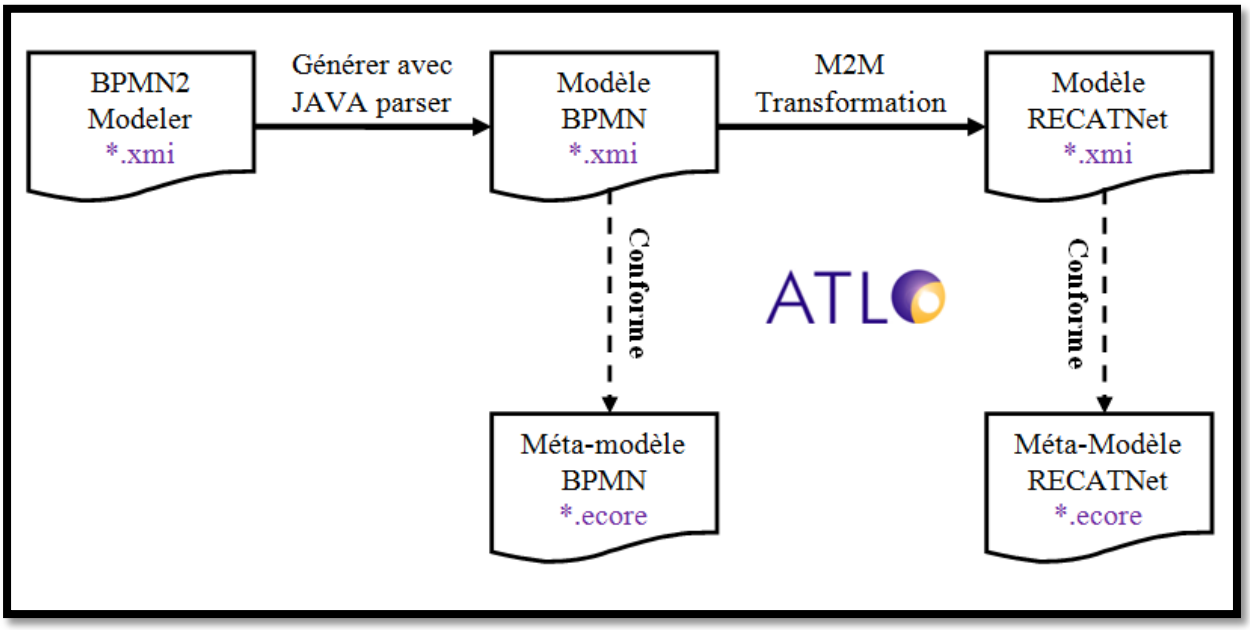


Figure 5.1: Schéma pour la transformation d'un modèle BPMN vers un modèle RECATNet

La figure 5.1 représente les différentes étapes de la transformation du modèle BPMN vers RECATNet.

Pour faire la transformation ATL, il faut d'abord utiliser un parser JAVA dans le but de transformer un fichier BPMN2modeler *.xmi vers un autre fichier *.xmi conforme à notre méta-modèle de BPMN. Ensuite, faire la transformation du fichier *.xmi qui est conforme à notre méta-modèle vers un autre fichier RECATNet *.xmi par ATL (Atlas Transformation Language).

3. Règles ATL :

Dans cette étape, nous allons implémenter les règles de transformation qui sont défini précédemment, nous allons les programmer dans le langage de Transformation ATL.

✓ Exemples de Helpers :

Les "helpers" ATL permettent aux développeurs de définir leurs propres méthodes dans les différentes unités d'ATL. Dans le contexte d'ATL, ces méthodes sont appelées "helpers". Comme toutes méthodes, ils permettent de factoriser du code ATL que l'on peut appeler à différents endroits du programme. Il existe deux type de "helpers" : fonctionnel et attribut. Tous les 2 sont définis dans le contexte d'une entité donnée. Un "attribut helper" est utilisé comme un attribut alors qu'un "fonctionnel helper" accepte des paramètres.

La figure 5.2 représente un exemple pour un helper « In ». Ce helper prends un Objet comme paramètre et retourne le nom du flux entrant de cette objet.

```
helper def : In(name : BPMNmm!Object) : BPMNmm!Flow =
    BPMNmm!Flow.allInstances()->select(f | f.toObject = name).first();
```

Figure 5.2 : Code source du helper "In"

La figure 5.3 représente un exemple pour un helper « Pin ». Ce helper prends un Objet comme paramètre et retourne la place qui représente le flux entrant de cette objet.

```
helper def : Pin(nam : BPMNmm!Object):RECATNetmm!Place =
    thisModule.getPlace('P'+thisModule.In(nam).fromObject.name+
                        '-'+thisModule.In(nam).toObject.name);
```

Figure 5.3 : Code source du helper "Pin"

✓ Transformation de flow vers place :

```
rule FlowToRECATNet{
    from
        sq:BPMNmm!Flow
    to
        plc :RECATNetmm!Place(
            name <- ('P'+sq.fromObject.name+'-'+sq.toObject.name),
            x <-(((sq.fromObject.x).toReal() + (sq.toObject.x).toReal())/ 2).toString() ,
            y <-(((sq.fromObject.y).toReal() + (sq.toObject.y).toReal())/ 2).toString(),
            capacity <- 1,
            marking <- ''
        )
}
```

Figure 5.4 : Transformation de flux de séquence

La figure 5.4 représente la transformation ATL d'un flux de séquence (sq) vers une place (plc). Cette place va prendre le nom de l'objet précédent de flux avec le nom de l'objet suivant.

✓ Transformation de StartEvent :

```
rule startToRECATNet{
    from
        start:BPMNmm!StartEvent
    to
        plc :RECATNetmm!Place(
            name <- 'P'+start.name,
            x <- start.x,
            y <- start.y,
            capacity <- 1,
            marking <- 'token'
        ),

        inarc :RECATNetmm!InputArc(
            fromP <- plc ,
            toTr <- tr,
            DT <- 'token' ,
            IC <- 'token'
        ),

        tr : RECATNetmm!Transition(
            name <- 'T'+start.name ,
            x <- (((start.x).toReal() + (thisModule.Pout(start).x).toReal() )/ 2).toString() ,
            y <- (((start.y).toReal() + (thisModule.Pout(start).y).toReal() )/ 2).toString(),
            K <- '',
            TC <- 'token'
        ),
        outarc :RECATNetmm!OutpuArc(
            fromTr <- tr ,
            toP <- thisModule.Pout(start),
            CT <- 'token'
        )
}
```

Figure 5.5 : Transformation de StartEvent

La figure 5.5 représente la transformation ATL d'un évènement de début vers :

1- Une place (plc) qui prend un marquage initial 'M1', prend la capacité = 1 (nombre de jetons maximum) et prend le nom de l'évènement StartEvent de BPMN.
2- Une transition (tr) qui prend le nom de l'évènement StartEvent de BPMN.
3- Un arc entrant (inarc) qui fait la relation entre la place précédente et la transition suivante.
4- Un arc sortant (outarc) qui fait la relation entre la transition précédente et la place suivante.

✓ Transformation de And-Split :

La figure 5.6 montre la transformation ATL du fork vers :

1- Une transition élémentaire (tr) qui prend le nom du gateway fork de BPMN.
2- Un inputArc (inarc) qui fait la relation entre la place précédente (avec la fonction *Pin()*) et la transition suivante.
3- Deux OutputArc (outarc1, outarc2) qui fait la relation entre la transition précédente et les deux places suivantes (avec la fonction *Pout()*).

```
rule andsplitToRECATNet{
    from
        aj :BPMNmm!Gateway(aj.typegatway()='AND-Split')
    to
        inarc :RECATNetmm!InputArc(
            fromP <-thisModule.Pin(aj) ,
            toTr <- tr,
            DT <- thisModule.Pin(aj).marking ,
            IC <- thisModule.Pin(aj).marking
        ),
        tr : RECATNetmm!Transition(
            name <- 'T'+aj.name ,
            x <- aj.x ,
            y <- aj.y ,
            K <- '',
            TC <- 'token'
        ),
        outarc1 :RECATNetmm!OutpuArc (
            fromTr <- tr ,
            toP <- thisModule.Pout(aj),
            CT <- 'token'
        ),
        outarc2 :RECATNetmm!OutpuArc (
            fromTr <- tr ,
            toP <- thisModule.Pouts(aj) ,
            CT <- 'token'
        )
}
```

Figure 5.6 : Transformation de fork gateway

✓ Transformation de sous-processus:

La figure 5.7 montre la transformation ATL du sous-processus vers :

1- Une transition ***abstraite*** (tr) qui prend le nom du subProcess de BPMN et le marquage initial (startMarking) par la fonction « startSubProc() ».

2- Un inputArc (inarc) qui fait la relation entre la place précédente (avec la fonction *Pin()*) et la transition abstraite suivante.

3- Un OutputArc (outarc) qui fait la relation entre la transition abstraite précédente et la place suivante (avec la fonction *Pout()*) plus le termIndex<0>.

4- Un CutStep (cut1) qui prend un index identique au termIndex de l'OutputArc plus le marquage final (avec la fonction endSubProc).

```
rule subProcToRECATNet{
    from
        sub :BPMNmm!SubProcess(sub.activitytype()='None')
    to
        inarc :RECATNetmm!InputArc(
            fromP <-thisModule.Pin(sub) ,
            toTr <- tr,
            DT <-'token' ,
            IC <-'token'
        ),
        tr : RECATNetmm!AbsTransition(
            name <- 'T'+sub.name ,
            startMarking <- '('+thisModule.startSubProc(sub).name+',1)',
            x <- sub.x ,
            y <- sub.y ,
            K <- '',
            TC <- 'token'
        ),
        outarc :RECATNetmm!IndexedOutputArc(
            fromTr <- tr ,
            toP <-thisModule.Pout(sub),
            CT <- 'token',
            termIndex <- '<'+thisModule.indice+'>'
        ),
        cut1 :RECATNetmm!CutStep(
            index <- 'Y'+thisModule.indice,
            finalMarking <-'M'+thisModule.endSubProc(sub).name +' > 0'
        )
        do {
        thisModule.indice <- thisModule.indice + 1 ;
        }
}
```

Figure 5.7 : Transformation de Sous-processus

4. Etude de cas

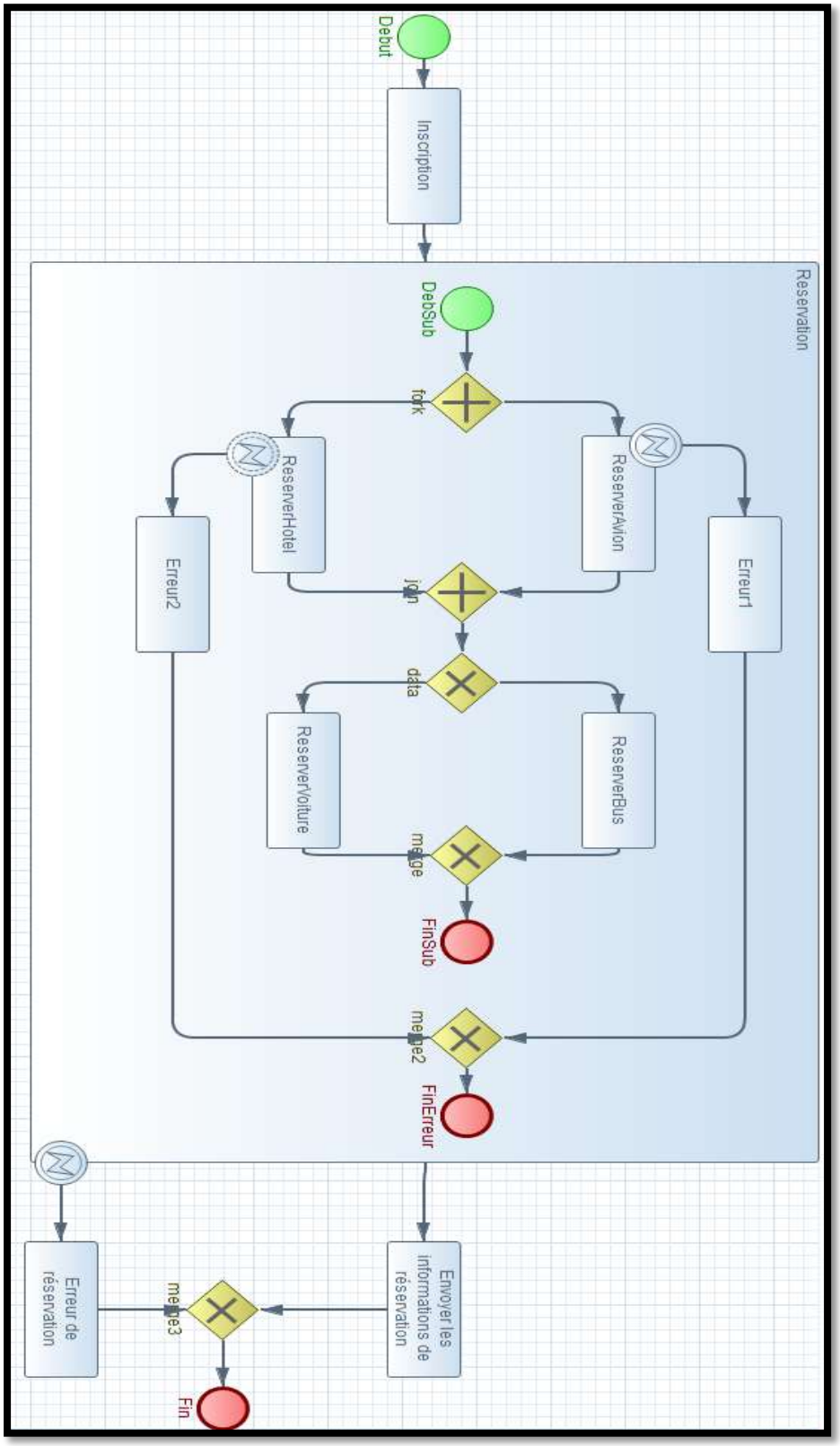


Figure 5.8 : Exemple de modèle BPMN dans BPMN2modeler

La figure 5.8 montre l'exemple de réservation de voyage (défini précédemment), nous allons appliquer cet exemple dans notre application. D'abord nous allons le dessiner dans l'interface BPMN2 Modeler avec l'éditeur Eclipse. Ensuite générer cet exemple avec le PARSER JAVA intégré dans notre application pour transformer ce modèle à un modèle conforme à notre méta-modèle.

La figure 5.9 représente la première l'interface de notre application qui permet d'importer (cliquer sur parcourir pour choisir le fichier) un fichier BPMN *.xmi pour le transformer au modèle conforme à notre méta-modèle de BPMN et de l'afficher dans cette interface. Et quand on clique sur le bouton « Générer Parser » alors le programme va nous envoyer à la deuxième interface.

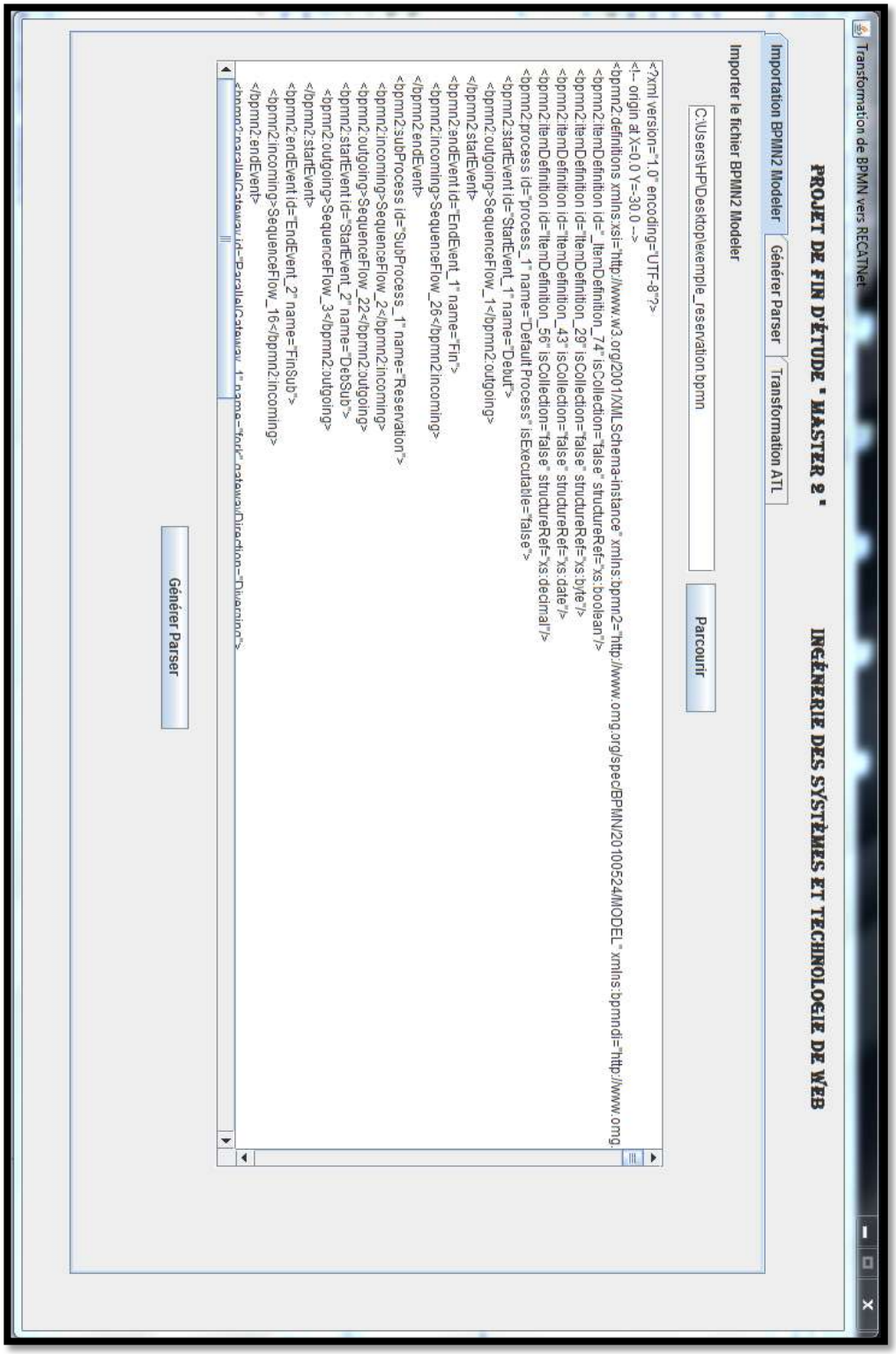


Figure 5.9 : Importation du fichier BPMN2 Modeler

La figure 5.10 montre la deuxième interface de notre application qui fait la transformation du fichier importé précédemment vers un fichier *.xmi conforme au méta-modèle BPMN et l'affiche dans un éditeur de texte.

Pour transformer ce modèle on doit cliquer sur le bouton ATL qui exécute une commande système (cmd.exe) pour faire appel à l'interface ATLStandalon pour exécuter notre fichier ATL qui fait la transformation de BPMN vers RECATNet.

La figure 5.10 nous envoi à la l'interface suivant « Transformation ATL ».

La figure 5.11 représente la troisième interface de notre application qui affiche le résultat de la transformation ATL, cette interface a un bouton « afficher Graphe » qui affiche le graphe de RECATNet résultat de la transformation.

Quand on clique sur le bouton en bas « Afficher le Graphe RECATNet », une autre interface va s'afficher pour le graphe dessiné.

La figure 5.12 représente le graphe résultat de notre projet généré par un programme JAVA. Les cercles représentent les places, les petits rectangles représentent les transitions élémentaires et les grands rectangles représentent les transitions abstraites.

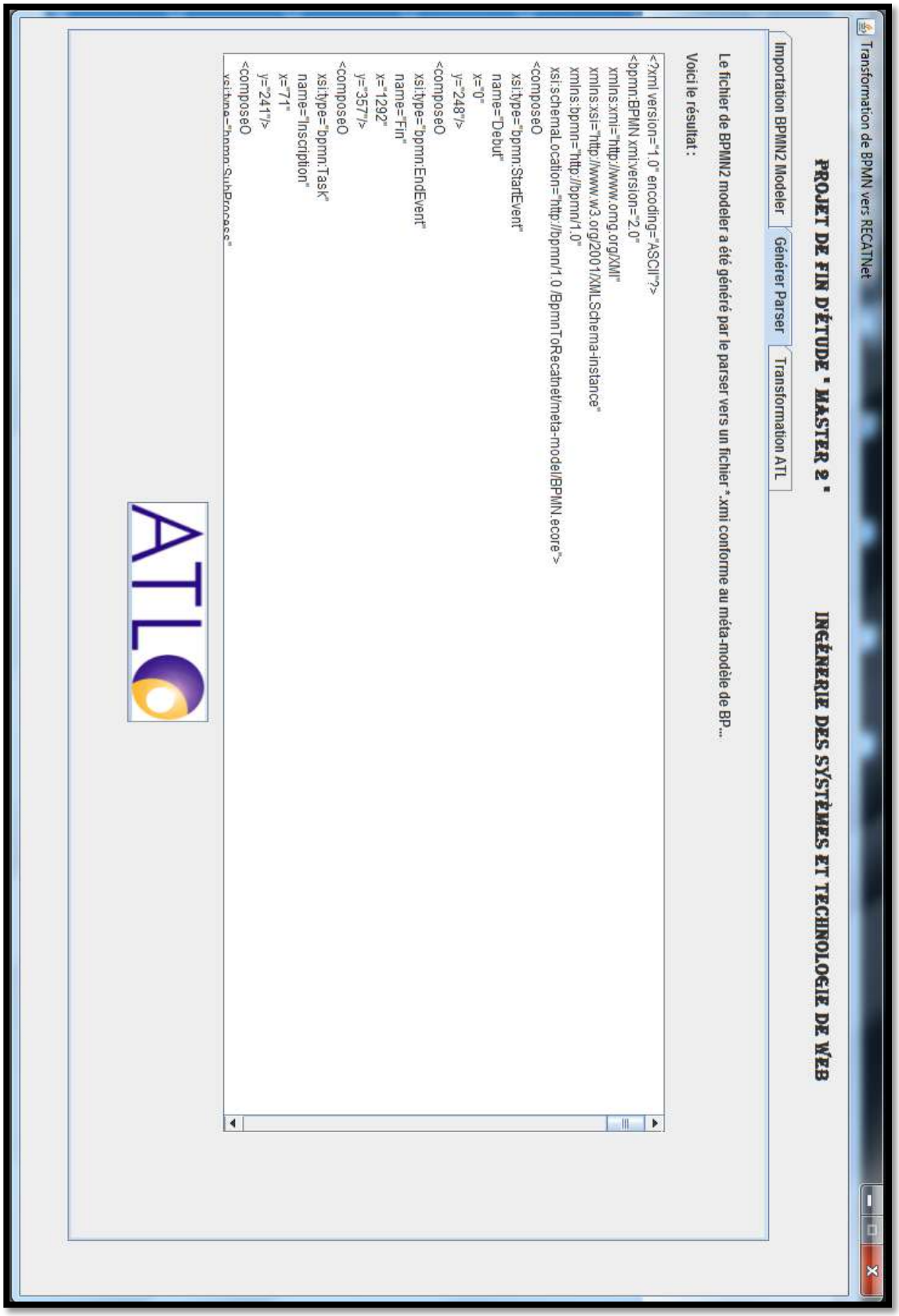


Figure 5.10 : Affichage du Parser et Transformation par ATL

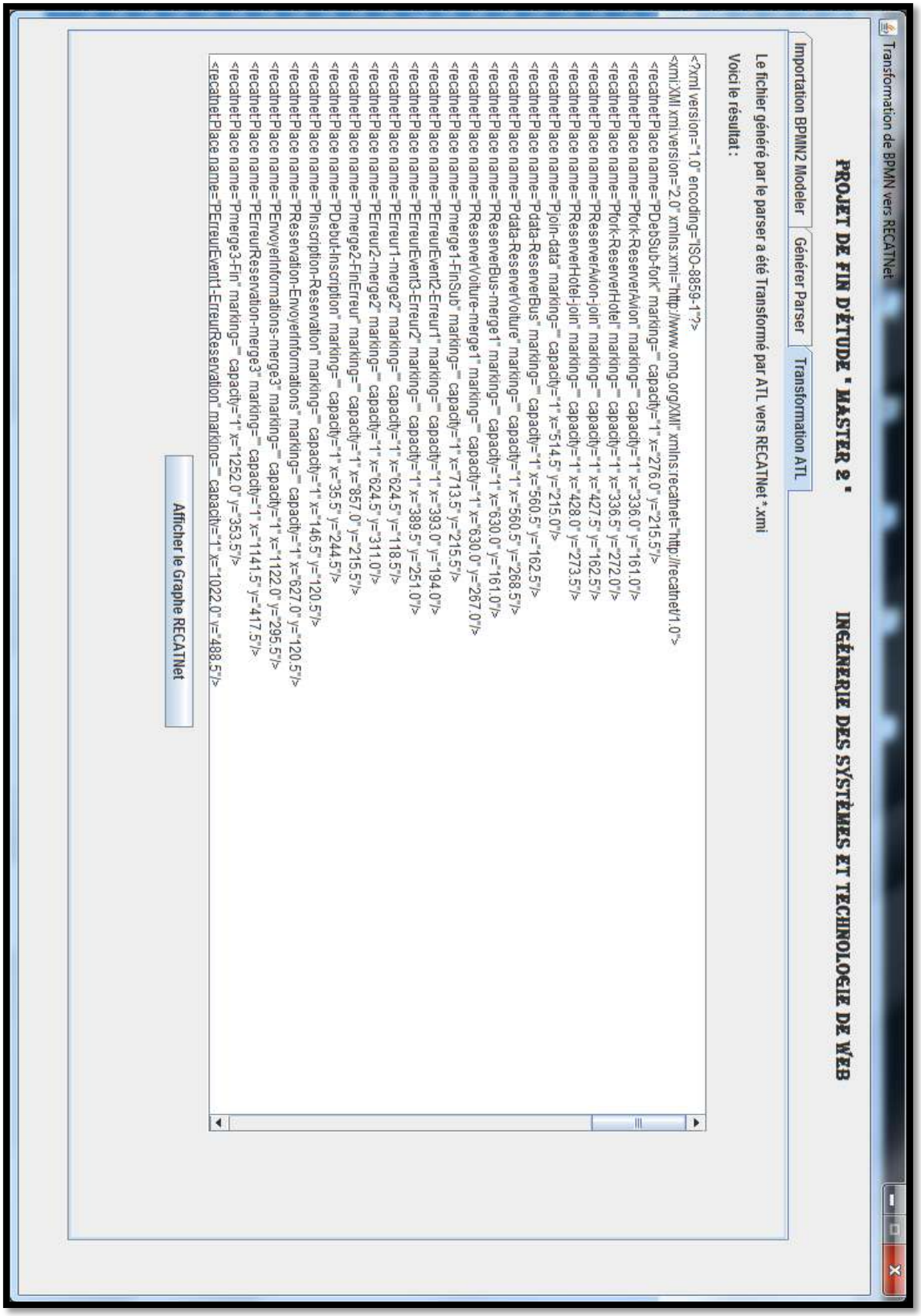


Figure 5.11 : Fichier RECATNet obtenu par la transformation

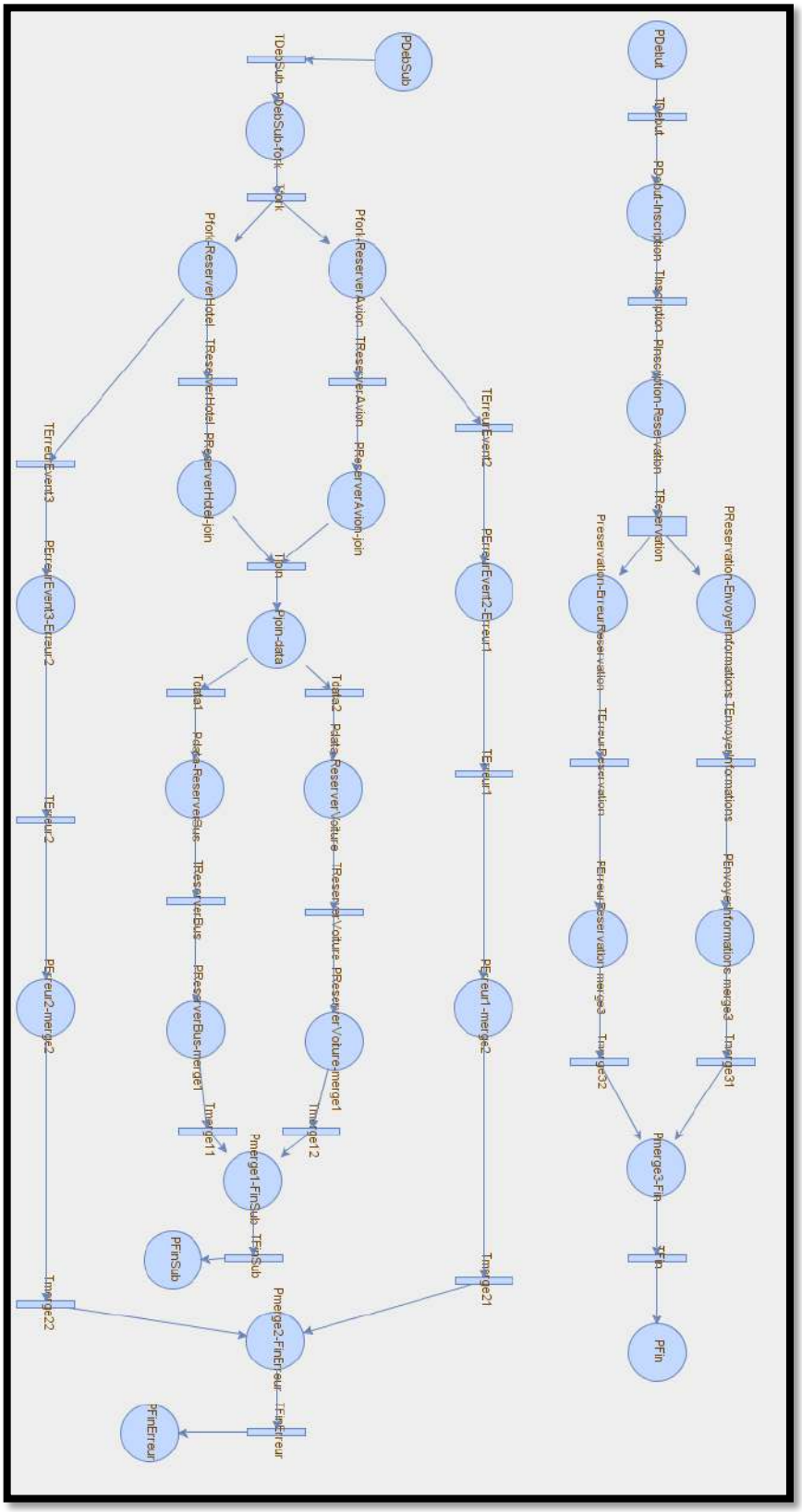


Figure 5.12 : le graphe résultat de RECATNet

5. Conclusion :

Dans ce chapitre nous avons présenté la réalisation de notre application, nous avons commencé par une petite explication sur l'approche proposé, en suite, nous avons présenté les deux méta-modèles de BPMN et RECATNet. Puis nous avons expliqué la partie de transformation « ATL » avec des exemples de transformation capturé à partir du code source. En fin nous avons présenté les étapes d'exécution du cas proposé et nous avons obtenu le RECATNet résultat sous forme d'un graphe.

Conclusion générale :

Dans ce mémoire, nous avons présenté, les RdP qui sont un outil graphique et mathématique qui s'applique à un grand nombre de domaine où les notions d'évènements et d'évolutions simultanées sont importantes. Ensuite nous avons présenté une catégorie de réseau de Pétri d'haut niveau appelée ECATNets récursifs qui introduisent la notion de récursivité à travers les transitions abstraites dans le but de modéliser la structure dynamique des systèmes. Puis nous avons présenté le langage BPMN qui permet de fournir des notations compréhensibles directement par tous les utilisateurs au niveau des entreprises. En effet, nous avons présenté les différents éléments de base de ce langage ainsi que les éléments avancés tels que les sous-processus, la multi-instance d'une activité et la gestion des exceptions.

Après l'état de l'art, nous avons présenté notre approche permettant la transformation d'un sous-ensemble de BPMN vers RECATNets sur la base des règles de correspondance pour laquelle une sémantique formelle est établie. En effet, notre approche permet de couvrir un ensemble large de fonctionnalités de BPMN comme l'instanciation multiple du sous-processus et la manipulation d'exception tout en prenant en compte l'aspect de flux de données.

En fin, nous avons implémenté l'approche proposée dans un outil JAVA qui fait la transformation automatique d'un modèle BPMN vers un modèle RECATNet. L'outil développé est constitué (1) d'un parser qui prend en entré un modèle BPMN créer par l'outil BPMN 2 Modeler et le transformer en un fichier xmi conforme à notre méta-modèle proposé pour BPMN, (2) d'un module de transformation des fichiers xmi générés par le parser vers des modèls de RECATNets utilisant le langage ATL (3) un module JAVA permet de visualiser les RECATNets générés.

Un des avantages d'utilisation des RECATNets est que leurs sémantiques peuvent être exprimées en termes de logique de réécriture conditionnelle. Par conséquent, nous pouvons utiliser le Maude LTL model-checker pour analyser sur plusieurs propriétés de comportement des processus métiers.

- **Perspectives**

Nous allons étendre le travail actuel en proposant des règles de transformation pour d'autres constructions BPMN telles que les messages échangés entre les processus, OR-JOIN, transaction, les activités de compensation ... etc.

Nous allons améliorer ce travail en tenant compte des contraintes temporelles et des ressources dans les modèles BPMN.

Références Bibliographiques :

[1] : Rachida HADIBY GHOUL, Les réseaux de Pétri : outil de Modélisation et de conduite des systèmes de production automatisés/ Livre, décembre 2012

[2] : C.Dutheillet, Modélisation des Systèmes Répartis (NI405)/Support de cours, Octobre 2012

[3] : Yann Morère, réseau de pétri/Support de cours, avril 2002

[4] : Awatef Hicheur, Les ECATNets récursifs pour la modélisation et l'analyse des processus workflows flexibles, 12 Mai 2006.

[5] : Awatef Hicheur and Kamel Barkaoui, thèse de doctorat : Modélisation et analyse des processus Workflows reconfigurables et distribués par les ECATNets récursifs, 2009.

[6] : Soualmi Yasmina et Madani Wassila, projet de synthèse pour l'obtention de diplôme de licence thème : Transformation de BPMN vers les réseaux de Pétri de haut niveau (RECATNETS), année 2013/2014.

[7] : Julien Da Costa, bpmn 2.0 pour la modélisation et l'implémentation de dispositifs pédagogiques orientés processus, février 2014.

[8] : Ahmed Kheldoun, Kamel Barkaoui, and Malika Ioualalen , Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach , 2015