C.D.T.A

C.D.T.A

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida

Faculté des sciences

Département d'informatique

Mémoire présenté par : Abdiche Mohand Amokrane & Fechit Abbas

En vue d'obtenir du diplôme de master

Domaine : Mathématique et informatique

Filière : Informatique

Spécialité : Informatique

Option : Ingénierie de logiciel

Sujet :

Automatisation de la transformation de modèle de processus métier BPMN en réseau de Petri

Soutenu le : 18/09/2017

Devant le jury :

Mme. Fareh	Présidente
Mme. Ghribi	Examinatrice
Mme. Arkam	Promotrice
Mme. Semar	Encadrante

Promotion

2016 / 2017

Dédicaces

A mes chers parents pour leur soutien moral, encouragements et sacrifices

A ma chère famille

Pour votre affection et tendresse

A mon binôme Abbas pour sa bonne collaboration

A tous mes amis

Au bonheur des plus chers

Je dédie cet humble travail

ABDICHE Mohand Amokrane

Dédicace

A mes très chers parents

À ma chère famille et tout ce qui m'aime

A tous mes amis

Au bonheur des plus chers

et qui aurait voulu partager ma joie ...

Je dédie cet humble travail

FECHIT Abbas

Remerciements

On tient à remercier dieu tout puissant de nous avoir permis de mener à bien notre mission

On remercie également notre encadreuse Mme. SEMAR Kahina pour l'orientation, la confiance et la patience qui ont constitué un apport considérable sans lequel ce travail n'aurait pas pu être mené à bon escient

Nos remerciements s'étendent également à Mme. ARKAME, notre promotrice pour ses bonnes explications qui nous ont éclairé le chemin de la recherche et sa collaboration avec nous dans l'accomplissement de ce modeste travail

Nous tenons à exprimer nos sincères remerciements à tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail

Pour conclure, Nous adressons nos remerciements les plus respectueux au jury qui ont accepté d'évaluer notre travail.

Résumé

De nos jours, les processus métier devient de plus en plus complexe Dans le cas où plusieurs processus ou sous processus interagissent entre eux, iis peuvent être contraint par des ressources partagées ou encore par des durées de traitements. La vérification de leur comportement global au moment de la conception, détermine à l'avance si un modèle de processus présente certains comportements souhaitables.

Dans le but de vérifier le comportement global d'un processus BPMN via les Models Checking (une méthode formelle de vérification de modèles). Nous avons effectué la transformation du modèle de processus BPMN en un modèle réseau de Petri afin de pouvoir vérifier son comportement global dans nos travaux futurs via les Models Checking.

La transformation de BPMN en réseau de Petri a était faite en deux parties. La première consiste à transformer de BPMN en XPDL en utilisant la méthode la plus adéquat. La deuxième partie consiste à transformer XPDL en réseau de Petri. Nous avons proposé une méthode de transformation ainsi que développer un outil qui permet de mettre en œuvre notre méthode. Cet outil a été réalisé en utilisant le langage de programmation Java EE ainsi que le langage de transformation de modèle ATL.

Mots-clefs : BPMN, réseau de Petri, XPDL, ingénierie dirigée par les modèles, transformation de modèles, ATL.

Abstract

Today, business processes become more and more complex. In the case where several processes or sub-processes interact with each other, they can be constrained by shared resources or by processing times. Verifying their overall behavior at design time determines in advance if a process model presents some desirable behaviors.

In order to check the overall behavior of a BPMN process via Models Checking (a formal method of model verification). We have carried out the transformation of the BPMN process model into a Petri network model in order to be able to check its overall behavior in our future work via the Models Checking.

The transformation of BPMN into Petri net was made in two parts. The first is to transform BPMN into XPDL using the most suitable method. The second part consists of transforming XPDL to Petri net. We proposed a method of transformation and created a tool that allows us to implement this transformation. This tool was made using the Java EE programming language as well as the ATL model transformation language.

Keywords: BPMN, Petri net, XPDL, model driven engineering, model transformation, ATL.

ملخص

في أيامنا، أصبحت العمليات المهنية أكثر تعقيدا، وفي حالة تفاعل العديد من العمليات أو العمليات الفرعية مع بعضها البعض، يمكن أن تقيدها الموارد المشتركة أو أوقات المعالجة. التحقق من سلوكهم العام في وقت التصميم يحدد مقدما ما إذا كان نموذج العملية معارض بعض السلوكيات المرغوبة.

من أجل التحقق من السلوك العام ل BPMN عبر Models Checking. لقد قمنا بتحويل نموذج العملية BPMN إلى نموذج شبكة بيتري من أجل أن نكون قادرين على التحقق من سلوكها العام في أعمالنا المستقبلية عن طريق Models Checking.

تم تحويل BPMN إلى شبكة بتري في مرحلتين. المرحلة الأولى هي تحويل BPMN إلى XPDL باختيار الطريقة الأنسب. تتمثل المرحلة الثانية في تحويل XPDL إلى شبكة بتري. قمنا باقترتح طريقة للقيام بهذا التحويل، فضلا عن تطوير أداة تسمح لنا بتجسيد هذه الطريقة. تم تطوير هذه الأداة باستخدام لغة برمجة Java EE بالإضافة إلى لغة تحويل النماذج ATL.

كلمات مفتاحية: BPMN, شبكة بتري, XPDL, الهندسة الموجهة بالنماذج, تحويل النماذج, ATL.

Table of Contents

Liste des figures

Liste des tableaux

Introduction générale

Introduction générale

Un processus est défini comme un enchaînement partiellement ordonné, d'exécution d'activités qui, à l'aide de moyens techniques et humains, transforme des éléments d'entrée en éléments de sortie en vue de réaliser un objectif dans le cadre d'une stratégie donnée.

On peut, donc, dire qu'un processus métier est un enchaînement d'activités qui prend un input (de n'importe quelle forme), lui rajoute de la valeur à l'aide de ressources pour fournir un output (produit /service) répondant aux objectifs de l'entreprise. Il est déclenché par des événements internes ou externes de l'entreprise. Il peut être décomposé en sous-processus et communiquer avec d'autres processus.

Dans le cas où plusieurs processus ou sous processus interagissent entre eux, ils peuvent être contraints par des ressources partagées ou encore par des durées de traitement. La vérification de leur comportement global au moment de la conception, détermine à l'avance si un modèle de processus présente certains comportements souhaitables ou non.

En effectuant cette vérification au moment de la conception, il est possible d'identifier des problèmes potentiels et, le cas échéant, le modèle peut être modifié avant son exécution. Une analyse des modèles de processus lors de la conception peut grandement améliorer la fiabilité des systèmes.

Le Models Checking est l'une des plus puissantes méthodes formelles de la vérification de modèles. Son principe est de générer tous les états possibles et de vérifier si les propriétés (comportementales et/ou structurelles) sont vérifiées pour chaque état mais cette méthode n'est pas applicable sur BPMN.

Dans le but de vérifier le comportement global d'un processus (BPMN) via le Models Checking. Le travail demandé en premier lieu est d'effectuer la transformation du modèle de processus BPMN en un modèle Petri net de ce dernier afin de pouvoir vérifier son comportement global dans nos travaux futurs via le Models Checking.

Cette étape de transformation (BPMN/Petri net) sera le sujet de notre projet de fin d'étude et qui est mené dans le cadre du projet SCIO-Web Social (Service de Collaboration Inter organisationnelle basée web social) du CDTA. Ce projet se devise en trois principales parties:

- Première partie : L'exploitation des réseaux sociaux professionnels pour la recherche et la sélection des meilleurs partenaires de collaboration ainsi que la modélisation des contrats de cette collaboration.
- Deuxième partie: Modélisation des processus de collaboration inter-organisationnelle, en utilisant l'ingénierie des connaissances.
- Troisième partie: La mise en œuvre de ce processus de collaboration sur le Cloud.

Notre travail se positionne dans la deuxième partie du projet. Le but de cette partie est d'exploiter les connaissances extraites du réseau et des partenaires de la collaboration afin de concevoir un modèle BPMN de processus collaboratif inter-organisationnel pour qu'il soit exécuté par la suite dans la plateforme Cloud du CDTA.

Dans le cadre de la mise en place d'une plate-forme de collaboration inter-organisationnelle sur le Cloud du CDTA, il est nécessaire de concevoir un module permettant d'assurer le passage automatique des modèles de processus collaboratifs inter-organisationnels (en BPMN) en des modèles XPDL puis Petri. Il existe un outil, BPMN transformer : BPMN TO PETRI, permettant la transformation de BPMN vers Petri, mais après l'avoir testé nous avons constaté que certains points de la transformation sont non-robuste et demande donc une vigilance (tel que : sous-processus avec exception, Porte OU, la porte-complexe,….) donc il ne peut être utilisé que dans des cas de processus BPMN simple. Dans notre cas, les modèles de processus inter-organisationnels sont très complexes, d'où la nécessité de concevoir notre propre outil de transformation automatique, d'autant plus de la nécessite de passer vers XPDL afin d'avoir des modèles qui peuvent être exécutés sur le Cloud du CDTA.

L'objectif de ce travail est de faire une étude comparative entre les différentes méthodes et techniques permettant la transformation automatique de processus métiers BPMN en XPDL puis en Pétri. Afin de concevoir et réaliser un outil permettant d'automatiser cette transformation pour n'importe quel processus métier modélisé avec BPMN.

Notre travail est basé sur l'approche MDA (Model Driven Architecture). C'est une approche de conception basée sur la transformation des modèles, supportant le développement de systèmes complexes et distribués. L'approche MDA est basée sur la transformation de modèles, de sorte que la construction du système soit un séquencement de modèles et de transformations entre ces modèles.

Ce mémoire est composé d'une introduction générale suivie de trois chapitres.

Le premier chapitre porte sur la modélisation des processus métier, une brève présentation de BPMN, XPDL et les réseaux de Petri ainsi qu'une introduction à l'architecture dirigée par modèles.

Le deuxième chapitre est dédié à la première phase de transformation (BPMN vers XPDL). Il comprend une étude comparative sur différentes méthodes et techniques de transformation de modèles BPMN en modèles XPDL ainsi que leurs principes de fonctionnement afin de choisir la méthode la plus adéquate ainsi qu'un outil qui permet la mise en œuvre de cette dernière.

Le dernier chapitre est consacré à la deuxième phase de transformation (XPDL vers réseau de Petri). Il contient aussi une étude comparative des différentes méthodes et techniques de transformation de modèles XPDL en réseaux de Petri. Suivi d'une conception de notre propre méthode de transformation. Ensuite, une présentation de l'outil que nous avons réalisé afin de mettre en œuvre la méthode de transformation proposée ainsi qu'un jeu de test.

Puis une conclusion générale viendra clôturer ce mémoire.

Chapitre I

Modélisation et transformation des processus métier

Chapitre I : Modélisation et transformation des processus métier

1. Introduction

Un processus métier, étant une unité fondamentale pour une organisation, a besoin d'être formellement défini d'une manière correcte, exposant ainsi le fonctionnement exact de l'organisation. La modélisation des processus métier peut effectivement être très utile dans différents domaines, notamment, dans le cadre d'une réorganisation ou amélioration du système ou, plus important encore, dans le cadre de vérification du fonctionnement correct des processus métier.

Une telle définition se fait à travers un langage (généralement standard) aisément compréhensible par tout type d'utilisateur. En ce qui suit, nous aborderons les différents modèles qu'on va utiliser au cours de notre travail.

2. Processus métier

2.1. Définition

Un processus métier est un ensemble de tâches liées les unes aux autres qui prenant fin à la livraison d'un service ou d'un produit à un client. Le processus métier a également été défini comme un ensemble d'activités et de tâches qui, une fois effectuées, rempliront l'un des objectifs d'un entreprise.

3. BPMN

3.1 Définition

BPMN est une représentation graphique de processus métier. Cette représentions peut être utilisé par les utilisateurs professionnels au lieu d'utiliser la représentation XML de processus métier qui est basé sur XPDL ou XML. BPMN traite la modélisation et pas l'échange de modèles entre outils pour cela BPMN a un ensemble riche d'éléments qui repends aux besoins de modélisation de processus métiers.

3.2. Ambition BPMN 2.0 et différence avec BPMN 1.0

- **BPMN 1.2**

Est une notation graphique non exécutable. Des solutions plus ou moins formelles qui sont développées pour implémenter les modèles BPMN, la conversion vers d'autres langages exécutables a reconnus des solutions qui ont donné des résultats variables. La majorité de ces solution consistent à passer par BPEL, cette étape est proposé par plusieurs outils commerciaux BPMN mais cette solution reste partielle et moins opérationnelle en outre certaines modèles

BPMN ne pas peuvent être converties vers BPEL car la définition des tâches humaines devient problématique.

❖ **BPMN 2.0**

Dans cette version BPMN a évolué vers un schéma d'échange standard basé sur XML permettant l'échange de modèles exécutables. BPMN 2.0 est devenue un langage de modélisation exécutable en remplacent BPEL.

« BPMN 2.0 étend la portée et les capacités de BPMN 1.2 dans plusieurs domaines : il officialise l'execution sémantique pour tous les elements BPMN, définit un mécanisme d'extensibilité pour les deux extensions du modèle de processus et d'extensionsgraphiques,affine la composition et la corrélation d'événements, étend la définition des intéractions humaines,définit la chorégraphie et les modèles de conversation (un pas en avant pour une meilleur modélisation des interactions), et résout également les bien connues incohérences et les ambiguités de BPMN 1.2 » (CHINOSI, 2012)

En résumé BPMN 2.0 ajoute par rapport à la version précédente les points suivants (Costa, 2012) :

- Un méta modèle normalisé et un format de sérialisation pour BPMN, qui permet aux utilisateurs d'échanger des modèles BPMN entre les outils de différents fournisseurs.
- Une sémantique d'exécutions normalisées pour BPMN, qui va permettre aux fournisseurs logiciels d'implémenter des moteurs d'exécution interopérables pour les processus métier.
- Un format d'échange graphique, permettant aux utilisateurs d'échanger les informations graphiques d'un diagramme de processus métiers
- Une notation étendue pour les interactions inter-organisationnelles (également connues sous le nom de *chorégraphies process*), qui permettra de créer de nouveaux cas d'utilisation pour les outils automatisés de soutien pour les processus qui impliquent plusieurs partenaires.
- Un processus de transfert détaillé de BPMN pour WS-BPEL, montrant l'alignement de BPMN avec les outils et les normes existants
- Certains éléments de modélisation supplémentaires pour des processus tels les événements et sous-processus non-interrompus.

3.3. Les éléments majors de BPMN

BPMN est constitué d'un ensemble d'éléments graphiques. Ces éléments permettent le développement d'un schéma simple. Les éléments ont été choisis pour se distinguer les uns des autres. Par exemple, les activités sont des rectangles, et les Gateways sont des diamants…etc. BPMN contient quatre catégories majors des éléments chaque catégorie contient plusieurs éléments comme mentionné dans l'organigramme suivant :

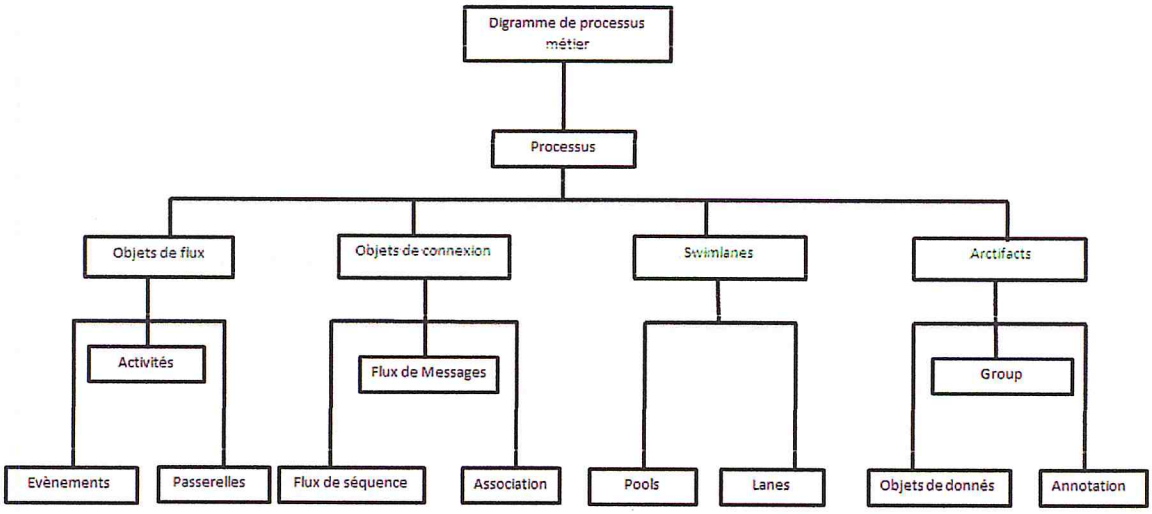


Figure 1: les éléments major de BPMN. (Jung, 2004)

3.3.1. Les objets de flux

Un objet de flux est un élément qui peut influencer dynamiquement le flux d'un processus, il existe trois classes des objets de flux : les évènements, les activités et les passerelles.

Lorsqu'un événement est déclenché, un déclencheur ou un impact évalue. Les activités sont la deuxième classe des objets de flux, une activité est une tâche qui sera exécuté lorsqu'elle sera activée par le jeton de processus. La dernière classe d'objets de flux est les passerelles. Ces derniers peuvent être utilisés pour représenter la divergence ou la convergence de contrôle tout au long d'un flux de processus (Hawajreh, 2014).

3.3.2. Les événements

Les événements sont les éléments BPMN qui déclenche les actions, un événement est représenté par un cercle. Il existe trois types d'événement : l'événement de début, l'événement intermédiaire et l'événement final. L'événement de début est l'élément qui peut déclencher une instance de processus. L'événement intermédiaire peut êtreretrouvé en tout lieu dans la

définition de processus entre l'événement de début et l'événement final, sa principale utilisation est de modifier l'état d'une instance de processus. Le dernier type d'évènements est l'événement final, ce dernier est utilisé pour mettre fin à une instance de processus. La ***Figure I.2*** représente les principaux événements dans BPMN.

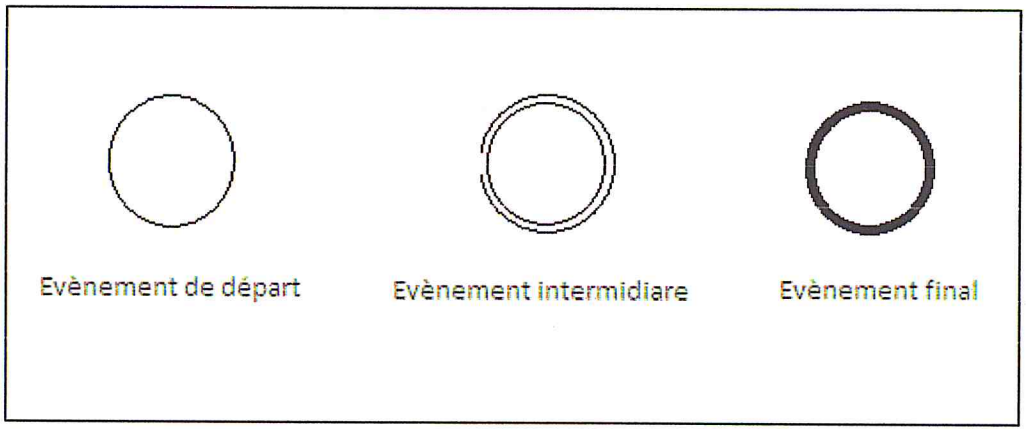


Figure 2: Les évènements en BPMN.

3.3.3. Les activités

L'activité est la deuxième classe d'objets de flux BPMN, il existe deux type d'activités selon la complexité : atomique et non atomique. Non-atomique est représenté par un sous-processus. Une activité atomique est une tâche qui ne peut pas être divisée en sous-processus. D'autre part, une activité de sous-processus peut contenir un sous-processus ou des processus imbriqués. La ***Figure I.3*** illustre les différentes notations d'activités dans BPMN.

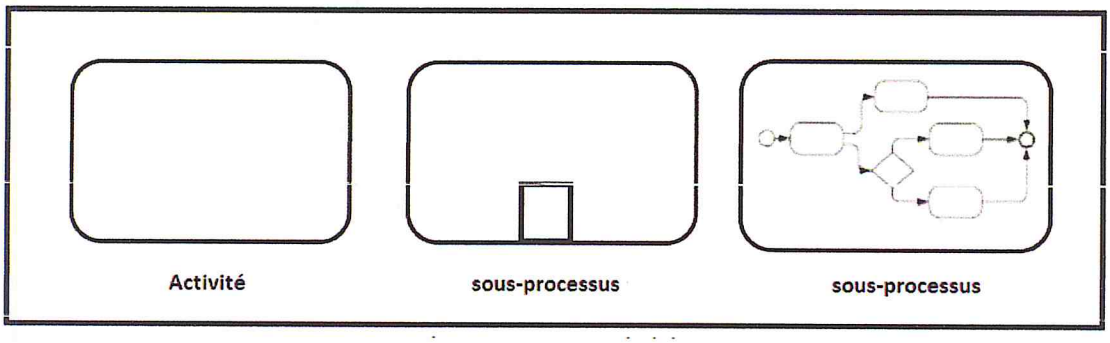


Figure 3: Les activités.

3.3.4. Les passerelles

Une passerelle est le composant BPMN qui modifie la convergence ou la divergence d'itinéraire. Sa forme graphique est un diamant. Il existe six types de passerelles, à savoir: exclusif, événementiel, basé sur des événements, inclusif, complexe et parallèle.

➢ **La passerelle exclusive**

La passerelle exclusive est utilisée lorsqu'il est nécessaire de parcourir un seul chemin multiple dans un flux séquentiel. Selon cette spécification, il doit y avoir des conditions à évaluer et selon ces conditions, un seul et même chemin est traversé. Cette condition est représentée par une expression conditionnelle attachée au connecteur de flux séquentiel. En outre, l'un de ces chemins peut être un chemin par défaut. La notation de la passerelle exclusive est exprimée par deux notations qui sont par défaut comme un diamant vide ou un diamant contient une marque X. La ***Figure I.4*** illustre comment différencier les deux notations de passerelle exclusives. Il existe deux types de passerelles exclusives, à savoir une passerelle exclusive divergente et une passerelle exclusive convergente. Dans une passerelle exclusive divergente, l'un des chemins parallèles est traversé. Le premier chemin évalué comme vrai peut être activé et les chemins restants sont désactivés. Dans la passerelle exclusive convergente, un chemin peut être activé pour passer à travers la passerelle exclusive à tout moment.

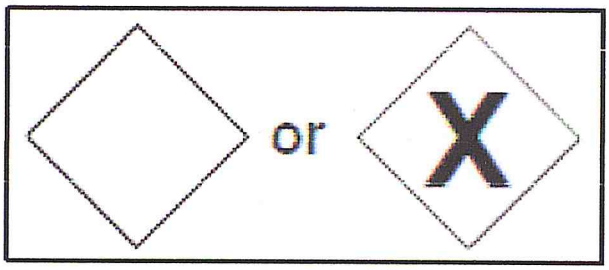


Figure 4: La passerelle exclusive.

➢ **La passerelle inclusive**

La passerelle inclusive donne la possibilité de parcourir plus d'un seul chemin au même temps, chaque chemin peut être parcouru si sa propre expression conditionnelle soit évaluée comme étant vraie. Il existe deux types de passerelle inclusive : convergente et divergente. Tous les arcs parallèles entrants à la passerelle convergente peuvent être actifs simultanément et pour la passerelle divergente c'est les arcs sortants qui peuvent être actif au même temps. La notation de la passerelle inclusive ou (ORgateway) est illustrée à la ***Figure I.5***

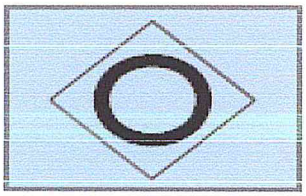


Figure 5: La passerelle inclusive.

- **La passerelle parallèle**

Contrairement aux passerelles précédentes, il n'y a pas d'expression conditionnelle correspondante aux arrêts dans la passerelle parallèle. Il existe aussi deux types de passerelles parallèles: divergente et convergente. Dans la passerelle divergente tous les arcs sortants sont activés d'une manière synchrone, d'autre part, la passerelle convergente sera activée seulement si tous ses les arcs entrants sont activés. Autrement dit, les arcs sortants de la passerelle parallèle ne peuvent jamais être actifs tant que les arcs entrants ne sont pas activés. La passerelle parallèle représente la passerelle AND qui est affichée dans la ***Figure I.6*** .

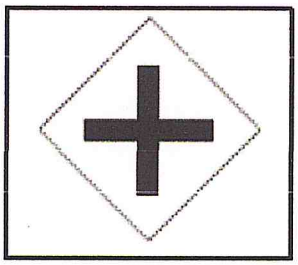


Figure 6: La passerelle parallèle

3.3.5. Les objets de connexion

Il existe quatre types principaux d'objets de connexion, flux de séquence, flux de message, association et association de donnés. Ces quatre types sont classés selon le flux de contrôle, les données, les messages et le flux anormal. Le flux de séquence est le lien qui contrôle le parcours des activités activés, le flux de message contrôle la circulation des messages entre deux participants qui proviennent de deux processus différents. Les associations sont utilisées pour corréler les artefacts. Le dernier type de liens est l'association de compensation qui montre le flux anormal, par exemple, lorsqu'une exception est lancée. Tous ces objets de connexion sont affichés dans la ***Figure I.7***.

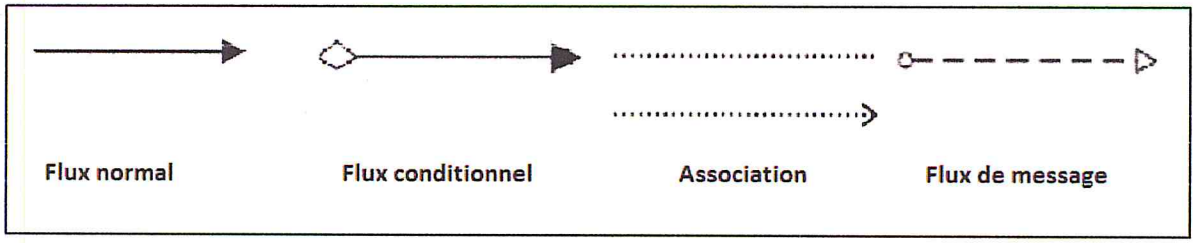


Figure 7: Les objets de connexion BPMN.

Des expressions conditionnelles et inconditionnelles peuvent être associées au flux de séquence. Le flux de séquentiel inconditionnel est représenté par une simple flèche entre deux activités sans toute évaluation d'expression. Cependant, le flux séquentiel conditionnel ne peut être transmis que si l'expression conditionnelle correspondante est évaluée. Certaines passerelles ont un flux de séquence spécial qui est par défaut. Ces passerelles ont plusieurs flux séquentiels conditionnels et peuvent éventuellement avoir un flux de séquence par défaut. Ce flux de séquence par défaut si tous les autres sont évalués comme faux, alors l'alternative n'est que le chemin par défaut.

3.3.6. Les swimlanes

Les swimlanes sont des boites rectangulaires utilisées pour représenter les participants d'un ensemble d'activités. Il existe deux types de swimlanes, Pool et Lane. Un Pool représente un groupe de rôles qui peut exécuter les activités pertinentes, un Pool peut être composé de plusieurs Lanes. Une Lane doit être contenu par un Pool, elle reflète un sous-niveau de catégorisation des activités contenues pour plus d'abstraction. La ***Figure I.8*** illustre les swimlanes.

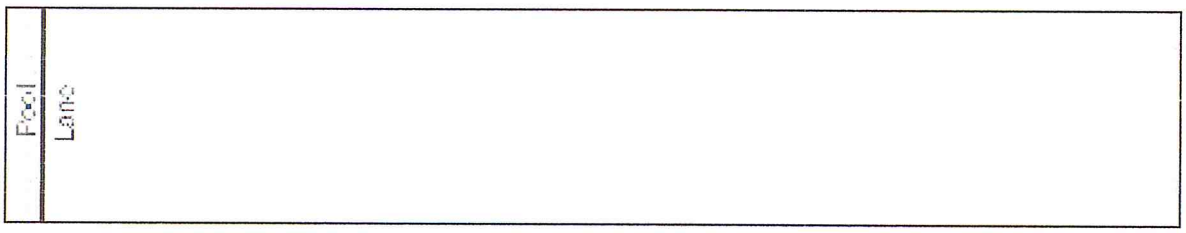


Figure 8: Swimlane.

3.3.7. Les artefacts

En BPMN, l'information est représentée par les artefacts. Les artefacts n'influence pas le flux dans un processus et ils sont utilisé seulement pour la documentation de diagramme. Il existe deux types d'artefacts, les groupes et les annotations de texte. Un groupe est utilisé pour

la description graphique d'un ensemble d'activités sous une seule catégorie. Les annotations de texte permettent la description textuelle d'une activité.

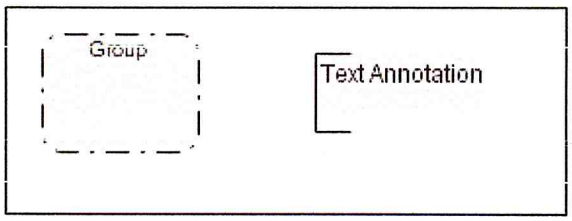


Figure 9: Les artefacts BPMN.

Dans la partie qui suit nous nous allons voir le format d'échange des modèles des processus métiers XPDL.

4. XPDL

4.1. Définition

XPDL est un langage de modélisation de processus basé sur XML. Il permet l'échange de modèles de processus entre les différentes applications, il est largement utilisé dans les moteurs de workflow pour automatiser le processus métiers.il est aussi un moyen de représentation de BPMN dans un format sérialisé. XPDL permet aussi de représenter un format exécutable d'une définition de processus qui est la référence de moteur de workflow et peut être le guide pour l'exécution de processus.

Le schéma XPDL version 2.2 possède un espace de noms différent et les outils souhaitant être compatibles avec XPDL version 1.0 doivent comprendre les espaces de noms XPDL 1.0 et XPDL 2.2.

Les éléments XPDL version 1.0 suivants ont été obsolètes dans la version 2.0 (WFMC, 2012):

- Elément automatique. Remplacé par les attributs StartMode et FinishMode de Activity.
- Attribut BlockId de l'élément BlockActivity. Remplacé par ActivitySetId.
- Élément DeadlineCondition. Remplacé avec DeadlineDuration.
- Index dans l'élément FormalParameter. Parce que FormalParameters doit correspondre à l'ordre dans la déclaration, et donc il n'est pas nécessaire d'indexer.
- Élément manuel. Remplacé par les attributs StartMode et FinishMode de Activity.
- L'élément Tool est obsolète.
- 4Élément Xpression dans l'élément Condition. Remplacé par Expression.

- L'ordre dans un processus de workflow a changé de DataFields, les participants et les applications pour être participants, applications et DataFields. Cela rend l'ordre en processus et package cohérent.

L'une des méthodes qui permettent la vérification de processus métiers est D'utiliser le réseau de Petri équivalent à ce dernier, dans ce qui suit nous allons parler de réseaux de Pétri.

5. Réseaux de Pétri

5.1. Aperçu sur les Réseaux de Pétri

Un réseau de Pétri est représenté par un type spécifique de graphes dirigés. Ce type de graphes est bipartite avec seulement deux types de nœuds, à savoir les boîtes (ou les barres) et les cercles. Les barres sont utilisées pour modéliser les transitions et les cercles sont utilisés pour modéliser les places. Les places peuvent contenir zéro ou plus de jetons. Ces jetons sont utilisés pour déclencher des transitions en fonction du nombre minimum de jetons pour déclencher une transition donnée. Ces nœuds sont connectés à des places utilisant des arcs dirigés et pondérés. Un poids d'arc est utilisé pour représenter le nombre requis de jetons pour déclencher une transition. Les Arcs peuvent connecter une transition vers une place ou une place vers une transition. Plus précisément, une place ne peut pas être directement connectée à une autre place, et une transition ne peut pas être directement reliée à une autre transition. Une transition peut être connectée à zéro ou à plusieurs places. En outre, une place peut être connectée à zéro ou plus de transitions. Pour refléter l'aspect dynamique d'un réseau de Pétri, on utilise des règles de marquage et de tir de transition pour représenter le changement d'état de Réseau de Pétri.

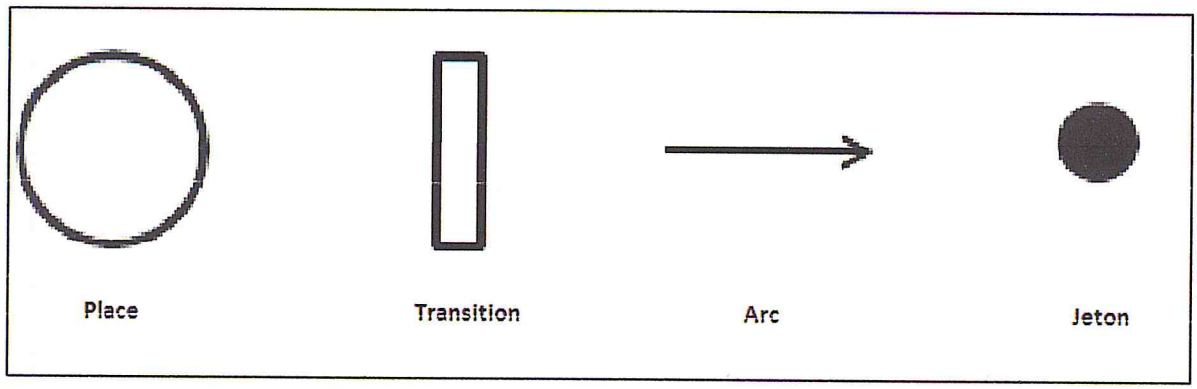


Figure 10: Les éléments de Réseau de Pétri.

6. Ingénierie Dirigée par les Modèles (IDM/MDE-Model Driven Engineering)

L'ingénierie dirigée par les modèles (IDM) est une approche spécifique du génie logiciel, elle s'appuie sur les modèles pour exprimer les préoccupations de manière plus abstraite lors de développement de systèmes complexes. Un système peut être décrit par différents modèles liés les uns aux autres. La réflexion sur l'IDM fait suite à la définition de l'approche Model Driven Architecture (MDA) (OMG, 2003) par l'OMG (OMG). L'IDM est une forme d'ingénierie générative qui aboutit au code du système décrit à haut niveau d'abstraction. Elle se focalise sur une modélisation abstraite du système réel en un modèle, et un passage automatique de ce modèle à un langage compréhensible par la machine.

6.1. Les concepts fondamentaux de l'IDM

6.1.1. Modèle

L'IDM s'appuie sur les modèles qui sont considérés comme les piliers de cette approche. considère le modèle comme étant une représentation (ou abstraction) d'un système, décrit dans une intention particulière. Définissent un modèle comme une simplification d'un système construit avec un objectif bien déterminé. Le modèle devrait être capable de répondre aux questions à la place du système actuel. Un modèle est une description ou une spécification d'un système et de son environnement dans un but bien déterminé. Un modèle est souvent présenté comme une combinaison de dessins et de textes. Le texte peut être dans un langage de modélisation ou dans une langue naturelle.

6.1.2. Méta Modèle

Un méta modèle est un modèle définissant le langage permettant d'exprimer un modèle. Un méta modèle est un moyen concret de définir un langage, ce n'est pas un langage. Dans le domaine de l'IDM, la méta-modélisation joue un rôle très important. En effet, elle est considérée comme une technique courante pour définir la syntaxe abstraite des Modèles et des interrelations entre les éléments du modèle. Si le modèle est une abstraction des éléments du monde réel, le méta-modèle représente encore une autre abstraction, définissant les propriétés du modèle lui-même. Un modèle est dit conforme à son méta modèle

6.1.3. La modélisation et la méta-modélisation

La modélisation des entreprises concerne la représentation et la spécification des différents aspects des opérations des entreprises. L'aspect fonctionnel décrit ce qu'on doit faire et dans quel ordre. L'aspect informationnel décrit quels sont les objets utilisés ou traités. L'aspect

ressource décrit qui "fait" les choses et selon quelle politique. Et enfin, l'aspect organisationnel décrit la structure organisationnelle dans laquelle les choses seront-elles faites.

La méta modélisation est défini par (Benoît, 2008)comme une activité consiste à définir le méta-modèle d'un langage de modélisation. Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser.

7. La transformation des modèles

7 .1 Définition

Une transformation est une génération automatique d'un ou plusieurs modèles cibles à partir d'un ou plusieurs modèles sources, en respectant la définition de la transformation. Une définition de transformation est un ensemble de règles de transformation qui décrivent la manière avec laquelle un modèle dans le langage source peut-être transformé en un modèle dans le langage cible. Une règle de transformation est une description de la façon avec laquelle une ou plusieurs constructions dans le langage source, peuvent être transformées en une ou plusieurs constructions dans le langage cible.

7.2. La transformation endogène

La transformation endogène est la transformation faite dans le même espace de technologique, dont le modèle source et le modèle cible sont conformes au même méta modèle, comme la transformation d'un modèle UML en un autre modèle UML.

7.3. La transformation exogène

C'est la transformation faite entre deux espaces technologique différents, dont le modèle source et le modèle cible sont conformes à des métas modèles différents. Par exemple la transformation d'un modèle UML en programme java et la transformation d'un fichier XML en schéma d'une BDD.

7.4. Mise en œuvre d'une transformation

Selon (Ramdane, 2016) La transformation des modèles sera réalisée en suivant les trois étapes suivantes :

1. La définition des règles de transformation.

2. Le choix d'un outil de transformation.

3. L'exécution des règles de transformation.

7.5. La définition des règles de transformation

Etant donné un modèle source dans un langage (par exemple BPMN), et un modèle cible dans un langage (par exemple XPDL), les règles de transformation sont établies entre le méta modèle source et le méta-modèle cible, c'est-à-dire entre l'ensemble des concepts du modèle source et celui du modèle cible. Le processus de transformation prend en entrée un modèle conforme au méta-modèle source et produit en sortie un (ou plusieurs) autre(s) modèle(s) conforme(s) au méta-modèle cible, en utilisant les règles préalablement établies. Dans notre travail, nous transformons un modèle XPDL de processus métier conforme au méta-modèle XPDL de processus métier en un modèle RdP de processus métier. Ce dernier sera conforme à notre méta-modèle RdP. On définit des règles de transformations ATL entre le méta-modèle source et notre méta-modèle RdP.

8. La transformation de BPMN en Réseau de Pétri

Plusieurs travaux existent dans la littérature ont ciblé la transformation de BPMN en Réseaux de Pétri, plusieurs travaux également ont opté de réalisé des outils qui permettent de faire cette transformation. Cette transformation a pour but de vérifier le comportement de processus métier avec des outils de vérification des réseaux de Petri ce qui permettre de savoir si le comportement de processus métier rencontre des problèmes tel que l'inter-blocage.

Nous ne intéressons pas de faire cette transformation mais plus tôt nous avons décidé de transformer BPMN en XPDL puis transformer XPDL en pétri, nous avons opté de suivre ce chemin pour garantir que le Réseaux de Pétri qui va être vérifié est équivalent au processus métier qui va être exécuté, parce qu'il faut transformer le processus métier BPMN en XPDL pour pouvoir l'échanger ou l'exécuter. La transformation de BPMN en XPDL peut engendre des pertes d'information ou d'élément et si on transforme BPMN en Petri directement puis transformer BPMN en XPDL nous somme pas sûr que le processus vérifié est équivalent au processus qui va être exécuté.

9. Conclusion

Ce premier chapitre énonce la modélisation des processus métier et la transformation de ces derniers. Nous avons présenté les concepts fondamentaux de BPMN, XPDL, MDA ainsi que ceux des réseaux de Petri. Ces derniers, sont un formalisme de modélisation des systèmes. Nous avons également décrit la transformation des modèles.

D'après l'étude faite, BPMN est une notation conçue pour la modélisation des processus métier, elle constitue l'une des notations les plus utilisées actuellement pour modéliser les processus

métier. Cependant, XPDL garantit l'échange, l'automatisation et l'exécution de processus métier.

Toutefois, pour une fin de vérification de ces derniers, la norme a besoin d'être accompagnée d'une méthode de modélisation. Nous avons constaté que les réseaux de Petri constituent un meilleur candidat. Ainsi, la transformation d'un modèle BPMN en XPDL puis XPDL en réseau de Petri nous permettra de bénéficier de la forte expressivité de BPMN et de la fiabilité du modèle offert par un réseau de Petri.

Afin d'entamer la transformation dont il s'agit, il est nécessaire d'étudier les différentes méthodes et techniques de transformation de modèle de processus métier ultérieurement.

Chapitre II

Transformation de BPMN en XPDL

Chapitre II : Transformation de BPMN en XPDL

I. Introduction

Dans le but d'exécuter et échanger les modèles de processus métier, il est nécéssaire de transformer BPMN en XPDL.

Dans ce chapitre nous allons introduire les travaux qui s'intéressent au mapping de BPMN vers XPDL et déduire les règles de mapping relatives à cette transformation. Par la suite nous allons parler des outils existants qui consistent à réaliser cette transformation automatiquement. Nous allons tester leur fiabilité et finalement choisir un outil qui garantira une transformation robuste de BPMN en XPDL tout en respectant les règles de mapping.

II. Etat de l'art de la transformation de BPMN vers XPDL

1. Introduction

Cette section consiste à faire une synthèse sur les travaux qui s'intéressent à la transformation de BPMN en XPDL.

2. Travaux existants

2.1. Travaux de WFMC

(WFMC, 2012) Est le document officiel de XPDL, ce dernier présente XPDL version 2.2 qui est compatible avec les versions antérieures de XPDL et peut être utilisé comme format de fichier pour BPMN 2.0 et BPMN 1.x. Ce travail prend en considération les éléments présents dans BPMN version 2.0 qui n'étaient pas présents dans les versions antérieures. Il fournit à chaque élément de BPMN, y compris ses attributs, un équivalent de XPDL afin d'assurer l'équivalence structurelle, comportementale et même les informations graphiques sont enregistrées.

2.2. Travaux de Jung et al

Dans les travaux présentés par Jung et al, le BPMN et le XPDL sont conçus comme une structure de graphique dirigée. Donc, le mapping de BPMN à XPDL proposé dans ce travail peut être décrit directement. Par exemple, une tâche et un sous-processus de BPMN sont transformés en une activité atomique et une activité de sous-flux de XPDL respectivement, et un flux séquentiel de BPMN est transformé en transition de XPDL. Comme BPMN est conçu pour modéliser et gérer des processus métier et XPDL est conçu pour exécuter des processus métier, il faut tenir compte de certaines différences entre BPMN et XPDL. Premièrement

BPMN comporte plusieurs éléments qui ne sont pas destinés à l'exécution de processus métier, mais seulement à la modélisation ou à la gestion de processus métier. Ces éléments BPMN n'ont pas besoin de mapper aux éléments XPDL. Deuxièmement, BPMN comporte plusieurs éléments pour l'exécution de processus métier mais ne se transforment pas directement en éléments XPDL. Les processus métier, qui sont représentés à l'aide de BPMN, y compris ces éléments, doivent être transformés dans le même sens que les processus métier XPDL.

3. Notre travail

(Jung, 2004) a était publié avant la sortie de XPDL 2.2, d'où il y a quelques éléments qui ne sont pas pris en considération, alors nous allons compléter cette insuffisance par (WFMC, 2012). D'où nous allons déduire notre table de mapping finale qui va être implémenté par la suite.

(Jung, 2004) Ne prend pas en considération les éléments qui ne sont pas liés à l'exécution du processus métier: les Swimlanes et les Artefacts. Dans notre cas nous voulons garder toutes les informations de BPMN, incluant les informations graphiques. D'où, nous avons opté à se baser sur le travail (WFMC, 2012) afin de compléter cette insuffisance. Nous avons pu compléter la table de mapping, que nous allons présenter par la suite.

III. Conception de la transformation de BPMN vers XPDL

1. Mapping BPMN vers XPDL

Dans cette section nous allons présenter chaque élément de BPMN avec son équivalent de XPDL en se basant sur les travaux mentionnés auparavant. Nous allons prendre en considération le mapping de toutes les éléments notamment le mapping structurel, simple et complexe.

2. Mapping structurel

Un processus métier est un flux de comportement composé par les éléments mentionnés dans le premier chapitre, la structure de processus métiers ne se diffère pas de BPMN à XPDL. Les figures suivantes illustrent la structure de processus métier de ces derniers. (Jung, 2004)

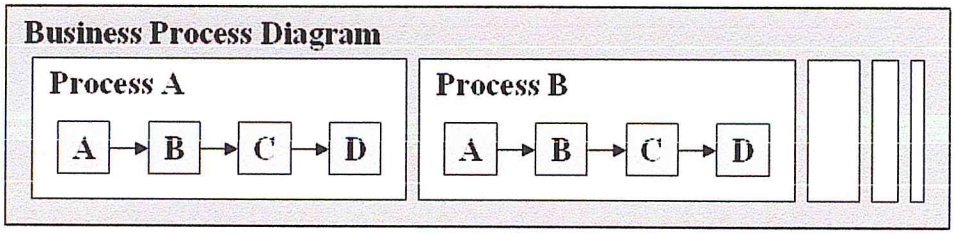


Figure II. 1 : ***Structure d'un processus métier de XPDL*** **(Jung, 2004).**

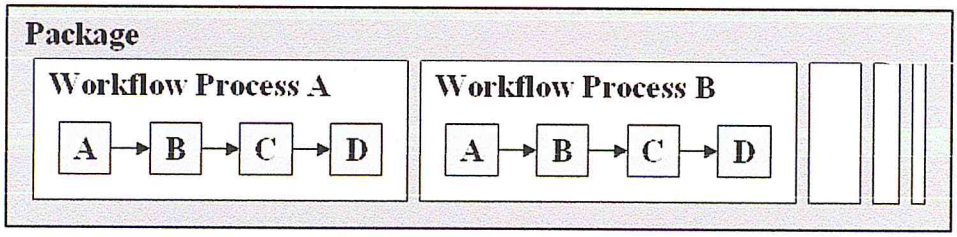


Figure II. 2: Structure d'un processus métier de XPDL (Jung, 2004).

Comme le montre les précédentes. Un processus BPMN est structuré par les objets de flux et de connexion qui forment un graphe orienté.en outre le processus de workflow définit par XPDL est composé par des activités et des transitions dans un format de graphe orienté, d'où nous obtenons la table de mapping structurel suivante

Element BPMN	**Element XPDL**
Business Process Diagram	Package
Process	Workflow Process
Flow Object	Activity
Connecting Object	Transition

Tableau II. 1: Structure du mapping (Jung, 2004).

Comme le montre la table précédente, Le package, le processus de workflow, l'activité et la transition en XPDL sont équivalents au diagramme de processus métier, processus, objet de flux et objets de connexion respectivement.

Les acteurs, les services et les données qui sont les éléments des processus métier sont représentés par acteur, Implémentation, Propriété dans BPMN et acteur, Tool, Data Field dans XPDL comme le montre la table suivante. (Jung, 2004)

Element BPMN	**Element XPDL**
Performer (withinActivity)	Performer (withinActivity)
Implementation (withinActivity)	Tool (withinActivity)
Property (within Process and Activity)	Data Field (within Workflow Process)

Tableau II.2 *Mapping des éléments du processus métier* (Jung, 2004).

3. Mapping simple

Dans la partie précédente, nous avons décrit que les objets de flux de BPMN et les objets de connexion et leurs équivalents dans XPDL. Mais les objets de flux de BPMN et les objets de connexion sont classés avec les événements, les activités, les passerelles et les flux de séquence, et ils sont classés plus détaillés en fonction de la valeur de l'attribut lié au type. Cette section décrit le mapping de ces éléments BPMN détaillés aux éléments XPDL qui se transforment directement.

3.1. Mapping des événements

L'événement BPMN agit comme un événement qui se produit ou réagit contre un autre événement. (Par exemple, annulation d'une commande, modification d'une commande et traitement de ces événements) et classé en fonction de sa position (début, intermédiaire, fin) et le type de déclencheur (message, minuterie, erreur, annulation, compensation, règle, Lien, Complexe, Terminé). (WFMC, 2012)

Les évènement s en BPMN sont des éléments qui produisent ou réagissent contre autre évènements. (Par exemple, annulation d'une commande, modification d'une commande…etc.), ils sont classés selon leurs position dans le diagramme de processus métier (début, intermédiaire, fin) et ainsi selon le type de déclencheurs (message, erreur…etc.).

Cependant, XPDL se base sur le contrôle basé sur les données. Par conséquent, il existe certains éléments dans ce dernier qui fonctionnent comme des évènements BPMN et beaucoup d'autres ne sont pas pris en charge. La table suivante présente les évènements BPMN et leurs équivalents en XPDL.

Element BPMN	**Element XPDL**
None Start Event	Route Activity
None End Event	Route Activity
Message Start Event	Route Activity and Formal Parameters of Workflow Process
Message End Event	Atomic Activity and Route Activity
AcyclicTimer Start Event	Route Activity and No Implementation Atomic Activity including Deadline
Acyclic Timer Intermediate Event (Normal Flow)	No Implementation Atomic Activity including Deadline
Acyclic Timer Intermediate Event (Exception Flow)	Deadline of AtomicActivity

Tableau II.3 *Mapping des événements de BPMN vers XPDL* (Jung, 2004).

3.2. Mapping des activités

Une activité de BPMN est une tache ou un ensemble des taches qui se déroulent dans un processus métier. L'activité est classée en sous-processus et tâche selon sa complexité, et ces classifications sont détaillées selon la méthode utilisée. Les différents types de sous processus BPMN et leurs équivalents en XPDL sont présentées dans table de mapping suivante.

Element BPMN	Element XPDL
Embedded Sub-Process	Block Activity
Independent Sub-Process	Subflow Activity
Reference Sub-Process	Block Activity or Subflow Activity equivalent to the Activity referenced

***Tableau II.4** Mapping des sous-processus de BPMN vers XPDL* (Jung, 2004).

La table de mapping suivante montre les différents types d'activité BPMN est leurs équivalents en XPDL.

Element BPMN	Element XPDL
Service Task	Atomic Activity
Receive Task	Atomic Activity
Send Task	Atomic Activity
User Task	AtomicActivityincluding Performer
Script Task	Atomic Activity including Extended Attribute which can contain the script
ManualTask	Manual Mode AtomicActivity
Reference Task	Activity equivalent to the Activity Referenced

***Tableau II.5** Mapping des tâches de BPMN vers XPDL* (Jung, 2004).

3.3. Mapping des passerelles

La passerelle BPMN contrôle l'écoulement du flux séquentiel divergent et convergent. Et les types de passerelle sont classés selon la division / fusion et la fonction. Dans XPDL, l'activité Route implique des transitions de partage ou de jointure, et elle fonctionne comme passerelles BPMN selon l'élément de restriction de transition dans l'activité Route. (Jung, 2004)

Element BPMN	Element XPDL
Exclusive Decision (XOR) – Data-Based	XOR Split Route Activity
Exclusive Merge (XOR) – Data-Based	XOR Join Route Activity
Inclusive Decision (OR)	AND Split Route Activity
Inclusive Merge (OR)	AND Join Route Activity
ParallelFork (AND)	AND Split Route Activity
ParallelJoin (AND)	AND Join Route Activity
ComplexDecision / Merge	Combination with several Route Activities and Transitions

***Tableau II.6** Mapping des passerelles de BPMN vers XPDL* (Jung, 2004).

La table de mapping ci-dessus représente les différents types de passerelles BPMN et leurs équivalents en XPDL.

3.4. Mapping de flux de séquence

Le flux de séquence de BPMN montre l'ordre des objets de flux. Et les types de flux de séquence sont classés en fonction de la valeur de l'attribut Condition.

Element BPMN	Element XPDL
Sequence Flow	Transition
Conditional Sequence Flow	Transition including CONDITION Type Condition
Default Flow	Transition including OTHERWISE Type Condition

***Tableau II.7** Mapping du flux de séquence vers XPDL* (Jung, 2004).

3.5. Mapping des Swimlanes

Les pools et les lanes vont tout simplement être respectivement mappés en Pool et Lane.

Element BPMN	Element XPDL
Pool	Pool
Lane	Lane dans Pool

***Tableau II.8** Mappings des Swimlanes de BPMN vers XPDL* **(WFMC, 2012).**

3.6. Mapping des artefacts

Element BPMN	Element XPDL
Data Object	Data Object
Group	Group
Annotation	Annotation

***Tableau II.9** Mapping des artefacts de BPMN vers XPDL* **(WFMC, 2012).**

4. Mapping complexe

Selon (Jung, 2004) le mapping complexe est le mapping des éléments BPMN complexes qui n'ont pas d'équivalents en XPDL avec les mêmes fonctions, ce type de mapping est atteint par la recomposition d'un processus métiers qui contient des éléments complexes d'une manière à obtenir un processus métiers sans éléments complexes. Le travail propose de suivre trois mécanismes pour ce mapping que nous allons présenter dans ce qui suit.

Mécanisme 1. Boucle

Le mécanisme de transformation Boucle est applicable sur les éléments BPMN qui fonctionnent d'une manière répétitive, la ***Figure II.3*** représente une deux exemples ce mécanisme. (Jung, 2004)

***Figure II.3** Mécanisme de transformation Boucle.* (Jung, 2004)

Mécanisme 2. Discriminateur

Le Discriminateur signifie qu'il ne réalise qu'un travail de tous les autres travaux qui sont mis en œuvre simultanément. Nous pouvons appliquer ce mécanisme à l'élément complexe, si

l'élément a la sémantique de "il finit B ou C". La ***Figure II.4*** représente un exemple du mécanisme de transformation Discriminateur. (Jung, 2004)

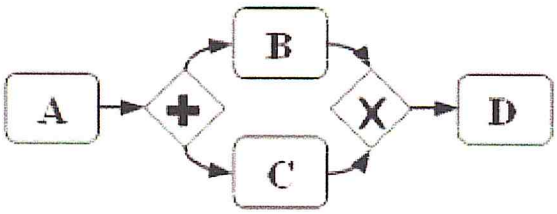


Figure II.4 *Mécanisme de transformation Discriminateur.* (Jung, 2004)

Mécanisme 3. Sérialisation

La sérialisation signifie qu'il transforme quelque chose, qui est sérialisé implicitement, vers une autre chose sérialisée explicitement. Nous pouvons appliquer ce mécanisme à l'élément complexe, si l'élément a la sémantique de "il fonctionne dans l'ordre d'une base". (Jung, 2004)

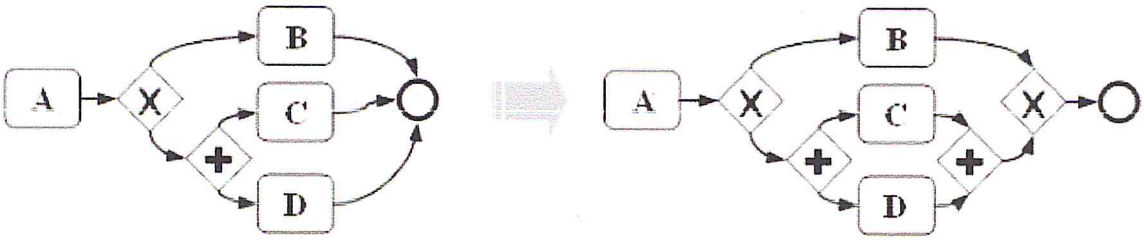


Figure II.4 *Mécanisme de transformation Sérialisation.* (Jung, 2004)

Tout d'abord, événement de la minuterie cyclique, activité de boucle standard, activité de boucle séquentielle à multi-instance et StartQuantity (Quantité de départ) l'attribut d'Activité s'applique au mécanisme de transformation Boucle; un événement cyclique à minuterie représente le travail qui est implémenté dans un cycle. La boucle standard et l'activité de boucle multi-instance séquentielle représentent le travail qui est implémenté à plusieurs reprises selon la condition d'activité. StartQuantity l'attribut d'Activité représente le travail qui attend jusqu'à ce que le nombre de jetons soit satisfait. Et la ***Figure II.5*** montre l'application du mécanisme de transformation Boucle à ces éléments BPMN. (Jung, 2004)

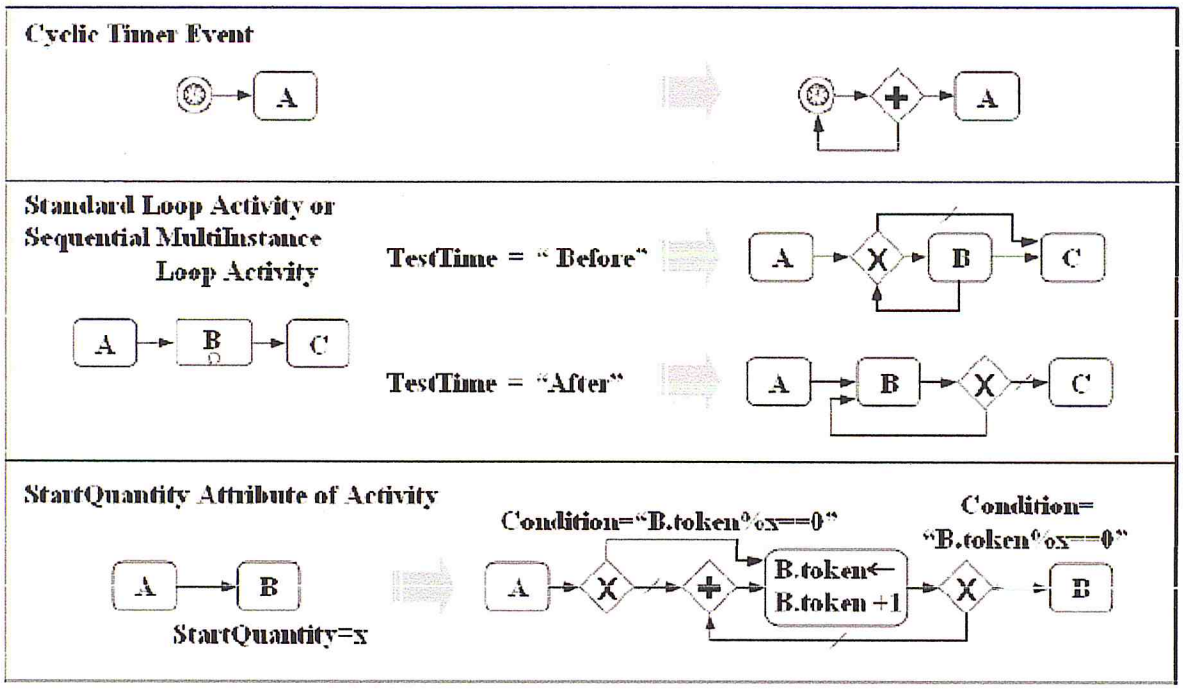


Figure II.5 *Classe de transformation de boucles.* (Jung, 2004)

Deuxièmement, événement intermédiaire de message (flux d'exception), événement d'erreur, événement multiple, événement de terminaison et Décision exclusive basée sur événement s'appliquent au mécanisme de transformation Discriminateur; L'événement intermédiaire du message (flux d'exception) et l'événement d'erreur signifient que cela entraîne un flux normal ou un flux d'exception. L'événement multiple signifie que si l'un des événements, qui est affecté à lui-même, se produit, il sera déclenché. Evénement de terminaison signifie qu'il interrompt tous les travaux, qui fonctionnent dans le processus, et met fin au processus à la fois au lieu de la fin normale. La décision exclusive basée sur événement signifie que seul un des événements, qui sont en ordre, peut être déclenché. Et la ***Figure II.6*** montre l'application du mécanisme de transformation Discriminateur à ces éléments BPMN. (Jung, 2004)

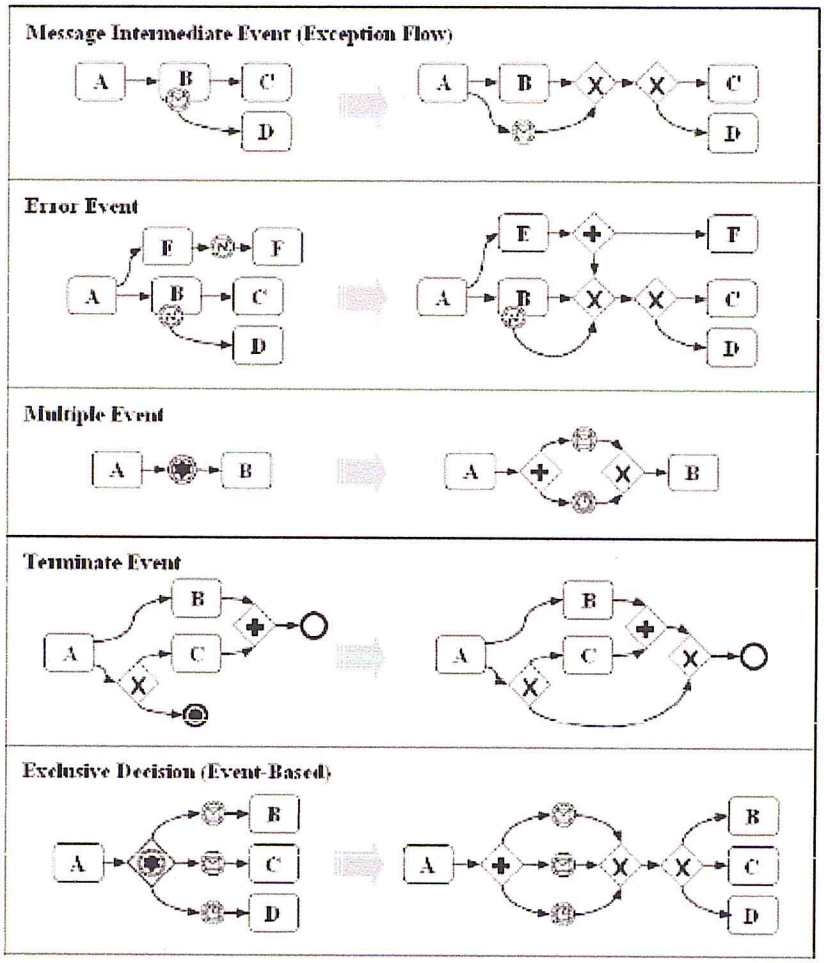


Figure II.6 *Classe de transformation des discriminateurs.* (Jung, 2004)

Troisièmement, événement de liaison et activité de boucle parallèle multi instance s'applique au mécanisme de transformation Sérialisation; Evénement de liaison représente le travail qui doit être connecté à un autre événement de liaison qui a le même LinkId (attribut qui représente l'identificateur). Activité de boucle parallèle multi instance signifie qu'il fait le même travail en parallèle. ***Figure II.7*** montre l'application du mécanisme de transformation Sérialisation à ces éléments BPMN. (Jung, 2004)

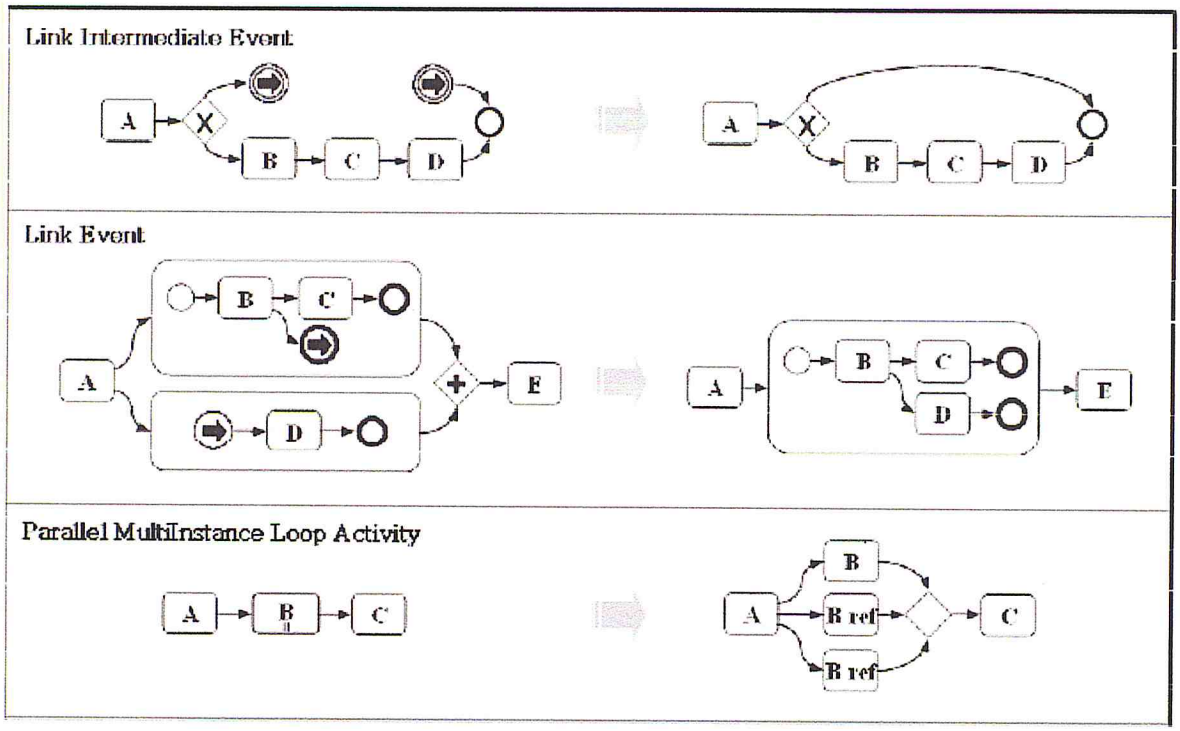


Figure II.7 *Classe de transformation de sérialisation.* (Jung, 2004)

Enfin, le sous-processus Ad-Hoc est le processus que l'ordre des travaux et la fin du processus sont décidés par les acteurs au moment de l'exécution. Donc, cela fonctionne en commençant par le travail, qui est sélectionné par un Acteur, à plusieurs reprises jusqu'à ce que la condition complète du processus Ad-Hoc soit satisfaite. Par conséquent, Processus Ad-Hoc s'applique à la fois aux mécanismes de transformations Boucle et Discriminateur. La ***Figure II.8*** montre l'application des mécanismes de transformation au sous-processus Ad-Hoc. (Jung, 2004)

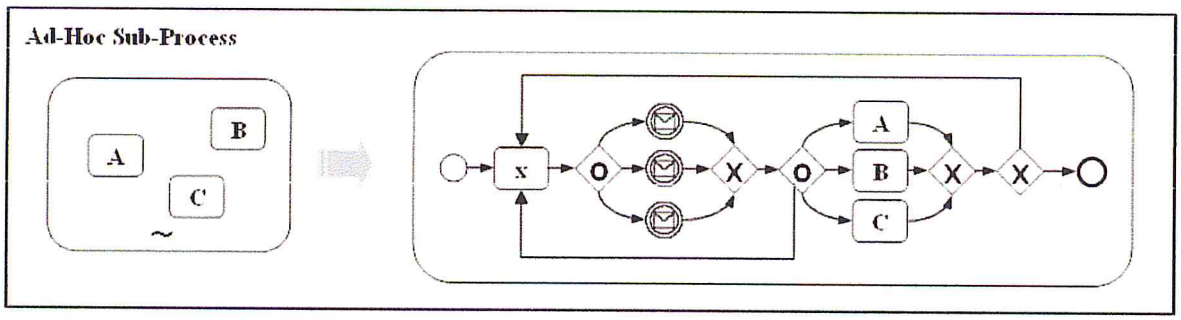


Figure II.8 *Transformation du sous-processus ad-Hoc* (Jung, 2004)

5. Conclusion

Dans cette section nous avons parlé de l'équivalence entre BPMN et XPDL et la relation entre leurs éléments. Nous avons présenté par la suite des tables de mapping donnant à chaque élément de BPMN son équivalent au niveau de XPDL. Cette partie portait sur un problème qui résultait d'une différence de signification entre BPMN et XPDL. Nous avons analysé les éléments qui sont mappés de BPMN à XPDL, puis nous avons proposé trois mécanismes de transformation (boucle, discriminateur et sérialisation) qui ont donné la possibilité de transformer les éléments complexes de BPMN tout en leurs transformant en des éléments de BPMN atomiques et simples.

Par la suite nous allons parler de l'implémentation des règles mentionnées dessus en mentionnant les outils qui nous aiderons à la réaliser.

IV. Implémentation de la transformation de BPMN vers XPDL

1. Introduction

Le but final de ce travail est l'automatisation de la transformation de BPMN vers XPDL, d'où le mapping mentionné dans la section précédente doit être automatisé. Dans cette partie nous allons voir et tester les différents outils qui permettent la transformation automatique de BPMN vers XPDL et choisir le plus pratique entre eux.

2. Outils de transformation disponibles

Nous avons constaté qu'ils existent plusieurs outils payant ou non qui font le passage de BPMN vers XPDL. Toute fois nous nous sommes intéressés seulement aux produits open source ou freeware. Dans cette section nous allons citer les outils plus utilisés par la communauté scientifique, à savoir :

2.1. Bizagi Process Modeler

Bizagi Process Modeler est une application freeware pour graphe, documenter et simuler des processus dans le format standard BPMN. Bizagi permet aussi de partager et à communiquer les modèles des processus métiers dans l'ensemble de l'organisation.

Bizagi BMPN Modeler supporte les versions les plus récentes de BPMN et XPDL c'est à dire BPMN 2.0 et XPDL 2.2.

2.2. ADONIS Community Edition

ADONIS Community Edition est une version gratuite de la trousse d'outils de gestion des connaissances, ADONIS a été créé par BOC Group. Ses fonctionnalités comprennent la modélisation, l'analyse et la simulation de modèles, ainsi que la documentation.

Ce dernier offre plusieurs types de modèles y compris BPMN 2.0. Il permet d'importer les modèles de BPMN en XPDL 2.2.ADONIS est disponible en téléchargement gratuit. Il peut être utilisé à des fins commerciales. (BOC, 2017)

2.3. ARIS

ARIS expresse est un outil de modélisation gratuit pour l'analyse et la gestion des processus métier. Il prend en charge différentes notations de modélisation incluant BPMN 2.0. Il donne aussi la possibilité d'exporter BPMN en XPDL.

2.4. Signavio

(Signavio Process Manager) est une application pour dédié à la modélisation et la simulation des processus métier. Il a des capacités de collaboration pour permettre aux parties

prenantes de partager et de commenter les modèles de processus et les diagrammes. Il stocke des objets dans un référentiel et peut partager des données avec plusieurs plates-formes d'automatisation de processus. Il est disponible pour l'installation sur site ou comme un service d'abonnement basé sur le Cloud.

3. Choix d'outil

Parmi les outils vus dessus, nous avons testé les outils 'Signavio' et 'Bizagi' parce qu'ils ont des outils très bien documentés et freeware et conseillé par la communauté scientifique. Dans ce qui suit nous présenter les résultats des tests de chacun.

- ✓ **Signavio**

Dans le but de tester la fiabilité de signavio nous avons créé plusieurs modèles de processus métier BPMN sur signavio en prenant en considération tous les éléments BPMN, et voir si ces élément BPMN vont être transformé correctement selon les mapping vu dans la section précédente et si il y a pas de perte.

Nous avons remarqué que signavio ne génère pas la dernière version de XPDL (2.2) c'est ce que nous empêche de réaliser un outil standard aux versions existante de XPDL. D'autre part, après avoir utilisé signavio pendant deux mois, nous avons constat é que Signavio ne transforme pas un des éléments BPMN relatif au comportement de processus métier qui est la passerelle basé sur évènement ce qui est considéré comme une anomalie. La figure suivante (***Figure II.9***) représente la passerelle basée sur évènement.

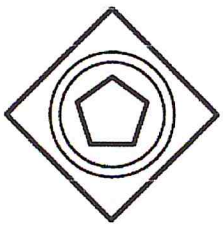


Figure II.9 *la passerelle basée sur évènement.*

Nous avons remarqué un autre problème dans signavio ou les sous processus ne sont pas pris en considération lors de l'exportation du modèle du processus principal en XPDL. Le sous processus va être considéré comme une activité atomique sans génération de modèle XPDL de ce dernier. Ce qui est considéré un manque d'information.

- ✓ **Bizagi**

Bizagi est l'un des outils de transformation les plus intéressants, il génère la dernière version de XPDL (XPDL 2.2). Il propose aussi un ensemble de modèle sur la page guide d'utilisateur ce que nous a permet de tester la transformation de tout élément BPMN notamment la passerelle basée sur évènement, cette dernière est transformé correctement en XPDL ce qui nous a confirmé que l'absence de cette dernière au modèle XDPL généré par Signavio est un défaut. Contrairement à Signavio, Bizagi génère les modèles des sous processus XPDL automatiquement avec la génération de modèle XPDL de processus principal, ce qui est considéré comme un point fort par rapport à signavio. Bizagi garantisse la transformation de chaque élément BPMN en un élément XPDL équivalent.

4. Synthèse

Après avoir testé plusieurs outils de transformation de modèle BPMN en XPDL notamment Signavio et Bizagi, et vu que nous avons trouvé des inconvénients dans les autres outils tel que la génération d'un modèle XPDL de versions antérieur…etc. Nous avons choisi Bizagi vu qu'il ne contient aucun défaut qui peut empêcher la validation de notre application.

5. Présentation de Bizagi Process modeler

Bizagi est un outil freeware permet de modéliser et simuler les processus BPMN, il permet aussi de générer les modèles XPDL en version 2.2. En utilisant Bizagi Modeler, il aussi possible de publier les processus sur Word, PDF, Wiki, Web ou SharePoint, ou exportés vers Visio et les formats d'image (png, bpm, svg ou jpg).la figure suivante (***Figure II.10***) représente l'interface principale de Bizagi.

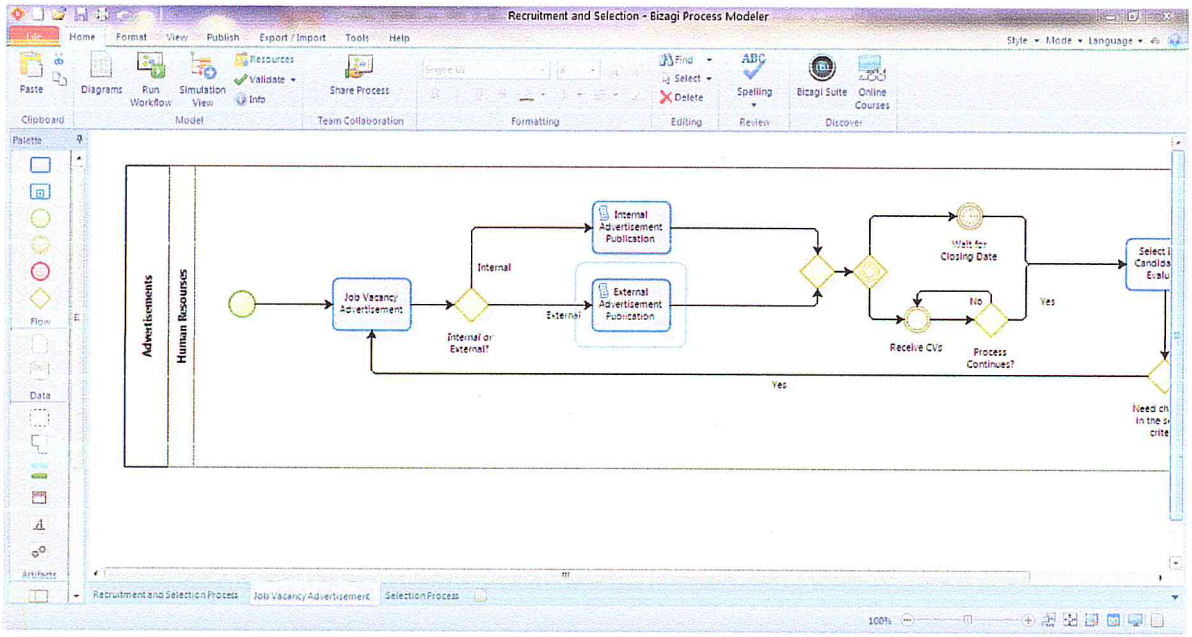


Figure II.10 l'interface principale de Bizagi.

6. Jeu de test

Dans le but de valider la fiabilité de Bizagi en ce qui concerne la transformation de modèle BPMN en modèle XPDL, nous allons appliquer dans ce qui suite un exemple de cas de processus métier BPMN « processus de sélection ». La figure suivante (***Figure II.11***) représente le processus de sélection XPDL.

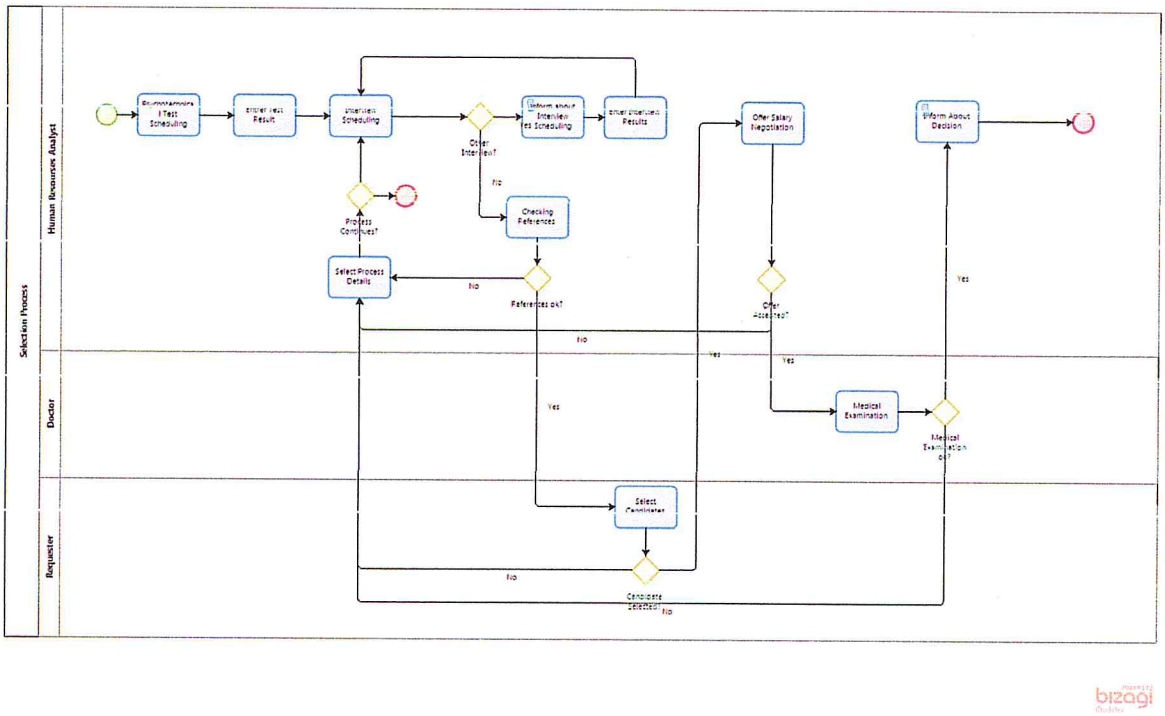


Figure II.11 l'interface principale de Bizagi.

Nous allons exporter le modèle « processus de sélection » en cliquant sur le bouton XPDL comme montré dans la figure suivante :

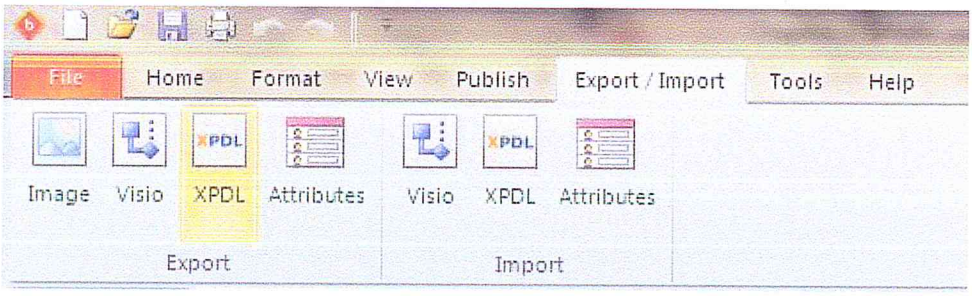


Figure II.12 Menu import/export de Bizagi.

Nous obtenons par la suite le modèle XPDL de processus de sélection.la figure suivante représente le modèle XPDL obtenue.

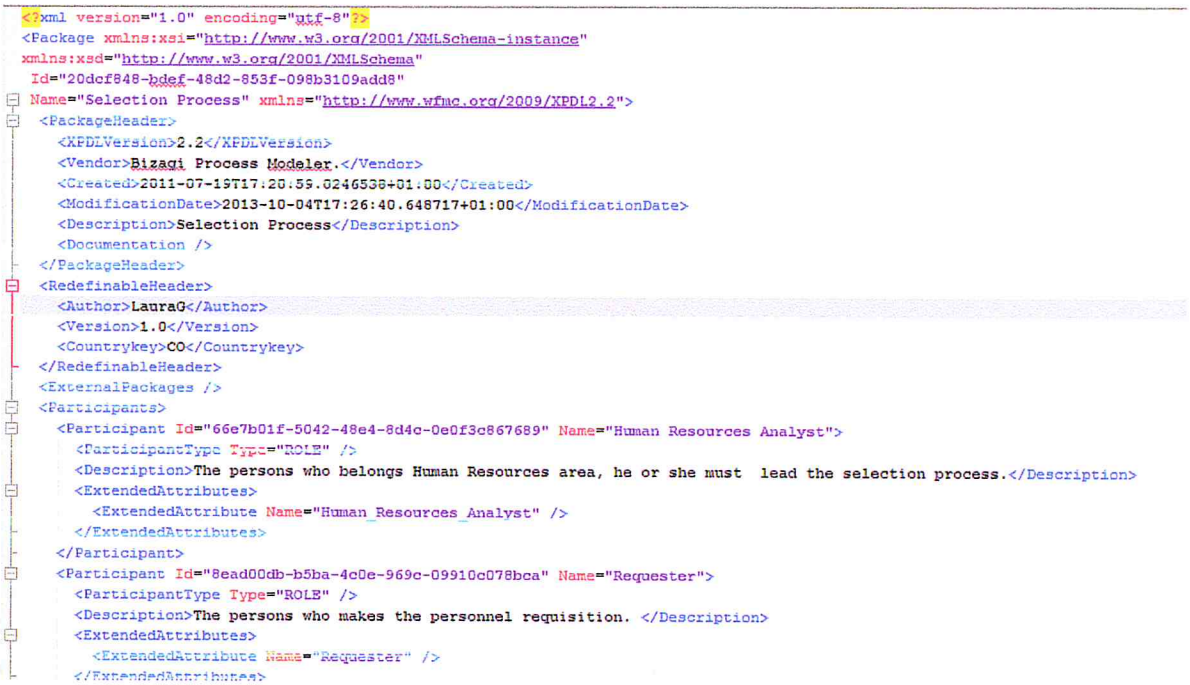


Figure II.13 *Le processus de sélection XPDL.*

A ce niveau nous avons obtenu le modèle XPDL souhaité sans perte de moindre d'informations ce qui nous a confirmé la fiabilité de Bizagi en transformation de modèle BPMN en XPDL.

V. Conclusion

Dans ce chapitre nous avons présenté les travaux qui s'intéressent à la transformation de modèle de processus métier BPMN en modèle XPDL, après nous avons fait une synthèse sur ces travaux, puis nous avons introduit les outils qui implémentent la transformation et choisit l'outil le plus fiable. Finalement nous avons fait un de test pour valider la fiabilité de l'outil choisit.

En ce qui suit, nous passerons vers la dernière et plus importante étape du travail à savoir le mapping XPDL en réseau de Petri.

Chapitre III

Transformation de XPDL en réseau de Petri

Chapitre III: XPDL vers Réseau de Petri

I. Introduction

Après avoir assuré le passage de BPMN vers XPDL, nous allons maintenant passer à la deuxième partie qui est le passage de XPDL vers les réseaux de Petri. Dans ce chapitre qui est diviser en trois sections intitulées état de l'art, conception et implémentation de la transformation de XPDL en réseau de Petri, nous allons concevoir et réaliser un outil qui permet la transformation de XPDL en réseau de Petri.

Nous aborderons différents algorithmes et méthodes qui permettent la transformation de XPDL vers le réseau de Petri pour sortir avec notre propre méthode. Nous allons par la suite faire une étude comparative entre les langages et les approches de transformation de modèles existants afin de choisir le plus convenable et finalement implémenter notre méthode.

II. Etat de l'art de la transformation de XPDL en réseau de Petri

1. Introduction

Dire qu'une méthode ou un algorithme de transformation de modèle est correct ou pas dépend du choix de la notion d'équivalence. Dans notre cas, la transformation de XPDL vers le réseau de Petri va être faite pour que le processus métier puisse être vérifié par la suite. D'où la notion d'équivalence adopter est bien l'équivalence structurelle et comportementale.

Dans cette section nous allons aborder différents méthodes et algorithmes de transformation de XPDL en réseau de Pétri qui sont intéressés à préserver l'équivalence structurel et comportementale entre ces deux derniers.

2. Travaux existants

2.1. Travaux de 'Haiping Zha et al'

Travail **(Haiping, 2007)** propose une méthode de transformation de processus de workflow en réseaux de Petri, cette méthode est applicable sur XPDL et plusieurs autres modèles de processus workflow. Le réseau de Pétri produit de cette transformation est nécessairement un WF-nets., un réseau de petri PN = (P, T, F) est WF-net si et seulement si:
- PN n'a qu'une seule place source i(aucun arc en entré et au moins un arc en sortie).
- PN n'a qu'une seule place puits o (au moins un arc en entré et aucun arc en sortie).
- Si on ajoute une transition t à PN qui connecte la place o avec la place i (où i est la place source et o est la place puits), le réseau de Petri résultant sera fortement connecté.

Cette méthode de transformation consiste à ne transformer que les éléments du processus de workflow qui ont une influence sur la structure ou le comportement de ce dernier: les Activités, les Transitions et les Structures de routage. Une activité de XPDL va devenir des transitions de Réseau de Petri et une transition de XPDL va devenir deux arcs qui sont directement reliés par une place comme montré dans la figure 1(a). Il ajoute aussi que les fonctionnalité de routage ont de différentes formes dans des différents langages de modélisation, donc il a donné cinq modèles qui représente ces derniers quel que soit le langage de modélisation utilisé, ces modèle sont comme suit:

✓ **Modèle 1. Séquence:** Une activité dans un processus de workflow est activée après l'achèvement d'une autre activité dans le même processus. **La figure III.1** (a) illustre le modèle de séquence en utilisant le formalisme du réseau de Petri, où l'activité 'b' sera activée après que l'activité 'a' a été déclenchée.

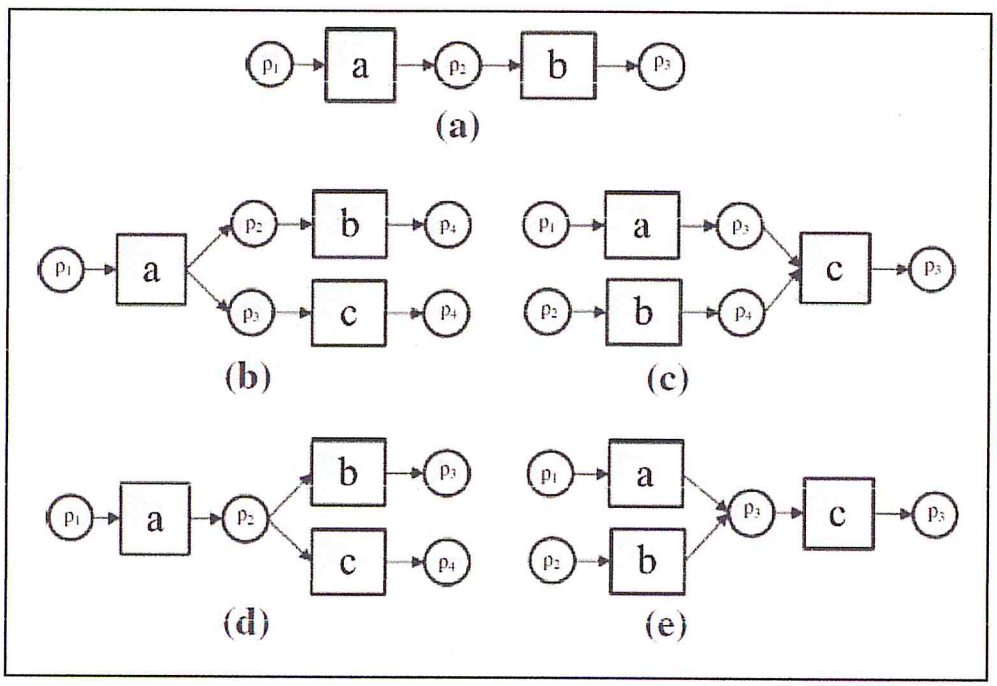


Figure III.1 : Sémantique formelle des modèles de control de base dans les réseaux de Petri.

✓ **Modèle 2. Division parallèle:** Une seule branche se divise en plusieurs branches parallèles, permettant ainsi d'exécuter des activités simultanément ou dans n'importe quel ordre. **La figure III.1** (b) montre la mise en œuvre du modèle division parallèle dans un réseau Petri. Une fois l'activité est déclenchée, les deux Place p2 et Place p3 seront marquées. Ensuite, les activités 'b' et 'c' sont activées et peuvent être déclenchées de manière indépendante.

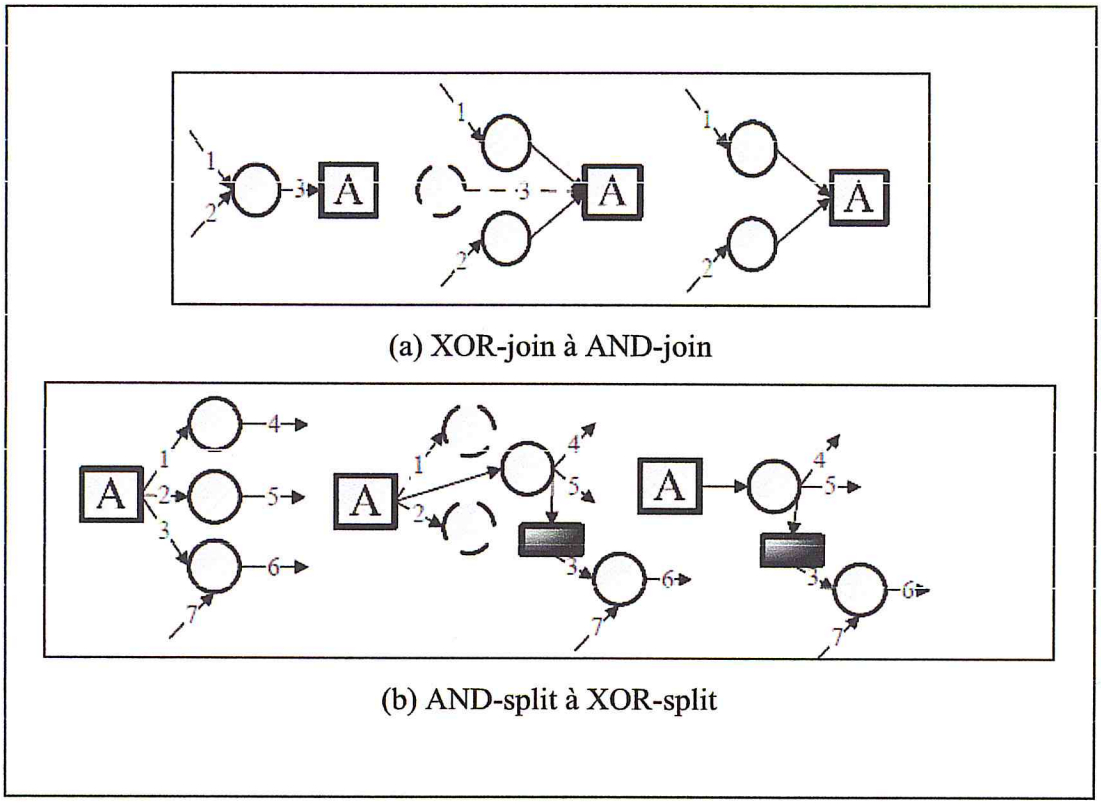
✓ **Modèle 3. Synchronisation:** Plusieurs activités parallèles convergent en une seule branche pour synchroniser plusieurs branches.La **figure III.1**(c) explique la mise en œuvre du modèle de synchronisation.

✓ **Modèle 4. Choix exclusif:** Une parmi plusieurs branches est choisie pour être exécutée en fonction d'une décision ou d'une donnée de contrôle de workflow. La **figure III.1** (d) illustre la mise en œuvre du modèle de choix exclusif.

✓ **Modèle 5. Fusion simple:** Deux ou plusieurs branches alternatives se réunissent sans synchronisation.La **figure III.1** (e) illustre la mise en œuvre du modèle de fusion simple.

2.2. Travaux de 'Yun Yang et al'

Le travail **(Yang,2007)** présente une méthode de transformation de modèle de processus XPDL en réseau de Petri. Ce dernier se concentre sur l'équivalence structurelle et comportementale seulement. D'où la transformation se fera que sur trois types de d'éléments de XPDL: les Activités, les Transitions et les Structures de routage. L'algorithme transformation consiste à transformer chaque activité de XPDL à une place et une transition de réseau de Petri relié par un arc. Les transitions de XPDL vont devenir des arcs de réseau de Petri sortant de la transition de l'activité source partant vers la place de l'activité de destination. Les structures de routage vont être traitées comme des activités. Mais, afin de préserver la cohérence sémantique et syntaxique, les résultats de la transformation des structures de routage AND-join (fusion inclusive) et XOR-split (division exclusive) en réseau de Pétri vont être encore transformées une deuxième fois, parce que toutes les structures de routages résultantes de la première transformation sont soit des XOR-join ou bien des AND-split. La transformation de ces deux derniers est montrée dans la **figure III.2**.



(a) XOR-join à AND-join

(b) AND-split à XOR-split

Figure III.2: La connexion des structures de routag.

En conclusion il propose l'algorithme de mapping d'un modèle XPDL vers un réseau de Petri suivant:

1. Planifiez chaque activité dans XPDL à une transition dans des réseaux de Petri, ainsi que leurs attributs nécessaires.
2. Ajoutez une place de saisie et un arc d'entrée pour chaque transition, à l'exception de celui qui n'a pas de liens d'entrée dans XPDL.
3. Mappez chaque lien vers un arc, reliant la transition source et la place d'entrée de la transition cible.
4. Corriger la structure AND-join, voir ***Figure III.2*** (a).
5. Correction de la structure XOR-split, voir ***Figure III.2*** (b).
6. Ajoutez une place source et une place puits.

Cet algorithme est suit par un théorème ainsi que sa preuve.

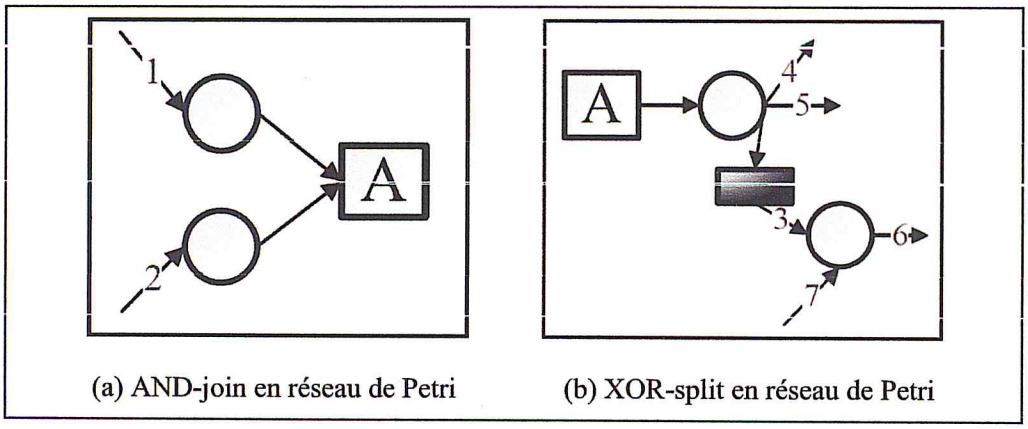
Théorème: Le réseau de Petri est équivalent avec le modèle XPDL en structure et les comportements observés, c'est-à-dire que l'algorithme de transformation est correct (Haiping,2007).

Preuve:

1. L'étape 1 garantit que les deux modèles ont le même ensemble d'activités.
2. L'étape 2 et l'étape 3 font que toutes les relations de commande d'activité soient les mêmes dans les deux modèles.
3. Puisque l'algorithme traite toute structure de jointure comme XOR-join et toute structure division comme AND-split, les structures originales XOR-join et AND-split restent inchangées après la transformation.
4. Dans les étapes précédentes, les structures AND-join sont traitées comme des structures XOR-join et des structures XOR-split en tant que structures split AND de manière incorrecte. L'étape 4 et l'étape 5 traitent ces défauts respectivement. Par conséquent, l'algorithme de transformation garantit la même structure des deux modèles. Par conséquent, il garantit les mêmes comportements observés.

2.3. Travaux de 'Ding Xiao et Qianqian Zhang'

Le travail **(Xiao, 2010)** propose un algorithme de mapping de XPDL vers le réseau de Petri. Cet algorithme se concentre seulement sur l'équivalence de la structure et du comportement du processus métier, d'où la transformation s'appliquera seulement sur les activités, les transitions et les structures de routage. Il propose de mapper chaque activité de XPDL à une transition de réseau de Petri puis la précéder par une place si l'activité n'est pas initiale. Les transitions de XPDL deviennent des arcs de réseau de Petri reliant la transition résultante de l'activité source avec la place ajouté à l'activité de destination. Concernant les structures de routage, XOR-join et AND-split vont être traitées comme des activités. Par contre AND-join et XOR-split vont être transformé comme des activités mais avec quelques changements:
AND-join: sera transformer en une transition précédée par un nombre de places égale au nombre d'élément qui la précède; chaque antécédent sera relié à une de ces places, voir **figure III.3** (a).
XOR-split: sera transformer juste comme une activité où tous les éléments qui le suivent aurons une seule place partagée entre eux au lieu d'une place chacune, voir **figure III.3** (b).



(a) AND-join en réseau de Petri (b) XOR-split en réseau de Petri

Figure III.3: *Transformation des structures de routage AND-join et XOR-split.*

Finalement, il ajoute une place source qui va avoir un arc sortant vers chaque transition résultante d'une activité de début. Il finit par une place puits qui va être la destination de toutes les activités de fin.

2.4. Travaux de 'Al Hawjreh'

Le document **(Al Hawjreh, 2014)** propose une méthode de transformation BPMN/XPDL en réseau de Petri. Cette méthode consiste mapper les éléments de BPMN/XPDL qui ont un effet sur le control de flux, c'est à dire les éléments qui définissent la structure et le comportement du processus de workflow. Les éléments de XPDL qui vont être mappés sont donc les activités, les transitions, les structures de routage. Il propose de mapper les activités en des transitions de réseau de Petri puis ajouter une place qui précède cette dernière. Les transitions de XPDL vont être mappées en des arcs. Les structures de routages XOR-join et AND-split sont mappées juste comme les activités, AND-join et XOR-split sont mappé autrement. Chaque élément va vers une structure AND-join doit passer par sa propre place avant d'arriver la transition de AND-join. Chaque élément sortant d'une structure XOR-split va être précédée par une place qui elle-même va être précéder par la structure XOR-split.
Finalement il indique qu'une place source et une place puits doivent être ajoutée. La place source sera la seule place marquée au début. Toute transition de réseau de Petri qui n'a pas de destination va être liée à la place puits.

2.5. Synthèse

Les travaux mentionnés dessus se mettent d'accord que lors de la transformation de XPDL en réseau de Petri pour qu'il soit vérifié par la suite, seule l'équivalence de la structure et du comportement doivent être assurée. D'où, les éléments de XPDL qui ont une influence sur la

structure ou le comportement du processus de workflow sont les seules êtres mappés, ces éléments sont: les activités, les transitions et les structures de routage.

Ces travaux adoptent le même mapping pour tout élément transformable de XPDL, mais chacun utilise son propre algorithme de transformation. (Haiping,2007) a opter pour une représentation générale des fonctionnalité de routage existantes dans XPDL en les classifiant en cinq modèles, puis il a indiqué comment transformer chacun d'eux en réseau de Petri. (Yang,2007) a opter pour un algorithme itératif où la première itération consiste à transformer les activités et les transitions de XPDL, la deuxième faire la correction des structures de routage AND-join et la dernière à corriger les structures de routage XOR-split. (Xiao, 2010) a opté pour un algorithme basé sur la théorie des ensembles, où les éléments de XPDL et de réseau de Petri vont être représentés par des ensembles lors de l'exécution de la transformation. (Both,2009)compte a appliquer ses règles de mapping directement, en parcourant les éléments du modèle XPDL en entrée un par un et transformant chacun d'eux à son équivalent dans le mapping proposé.

Dans le cas d'un processus métier complexe, l'algorithme itératif de (Yang,2007) et l'algorithme basé sur la théorie des ensemble de (Xiao, 2010) seront moins efficace qu'une transformation directe basé sur un ensemble de règles de mapping comme dans (Haiping,2007) et (Al Hawjreh,2014). Une transformation directe par contre nécessite l'utilisation d'autres outils en plus qui servent à implémenter les règles de mapping.

3. Conclusion

Dans cette section nous avons présenté quatre travaux intéressés à la transformation de XPDL en réseau de Petri. Nous avons souligné les points en commun et les différences entre ses travaux et nous les ont comparés.

Les modèles XPDL que nous allons transformer représente des processus métier complexe. Après l'analyse des travaux mentionnés dessus nous avons constaté que l'algorithme itératif et l'algorithme basé sur la théorie des ensembles seront moins efficaces que les deux autres à cause de la complexité des modèles à transformer. D'où notre conception, qui viendra par la suite, sera basée sur les travaux (Haiping,2007) et (Al Hawjreh, 2014) ainsi que le mapping commun adopté dans tous les travaux.

III. Conception de la transformation de XPDL en réseau de Petri

1. Introduction

Maintenant que nous avons analysé et comparer les différents algorithmes et méthodes de transformation de XPDL en réseau de Petri, nous allons finalement introduire notre propre méthode.

Dans cette section nous allons présenter les règles de mapping de chaque élément transformable de XPDL. Nous allons aussi mettre l'accent sur quelques cas spéciaux qui n'était pas traités dans les méthodes mentionnées dessus pour les compléter.

Remarque : **Afin d'éviter de confondre les transitions de XPDL avec les *transitions* du réseau de Petri, nous allons nommer les transitions de XPDL des *liens*.**

2. Méthode adoptée

Notre objectif est de traduire un modèle de processus XPDL en un réseau de Petri*. Cependant, nous ne poursuivons pas une sémantique complète de Petri net à XPDL. Nous nous concentrons sur la structure et l'équivalence comportementale. Nous présentons dans les figures (Haiping,2007) et (Al Hawjreh,2014) les méta-modèles de XPDL et du réseau de Petri qui sont respectivement le modèle source et le modèle cible de notre transformation.

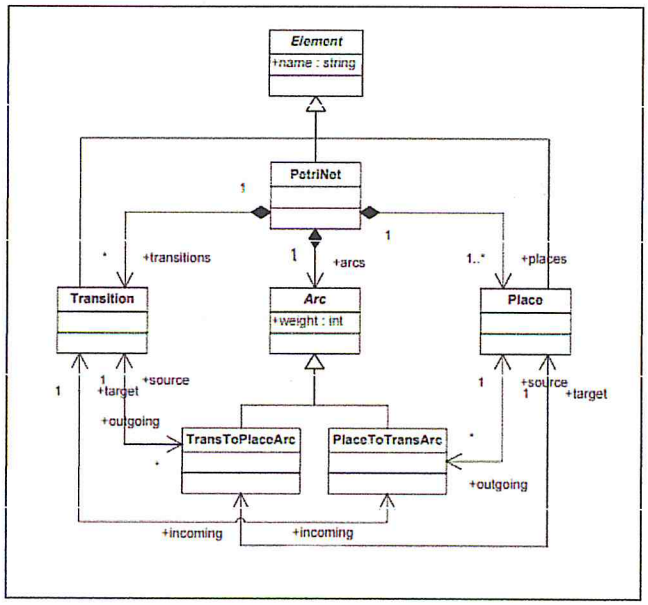


Figure III.4: *Méta-modèle du réseau de Petri.*

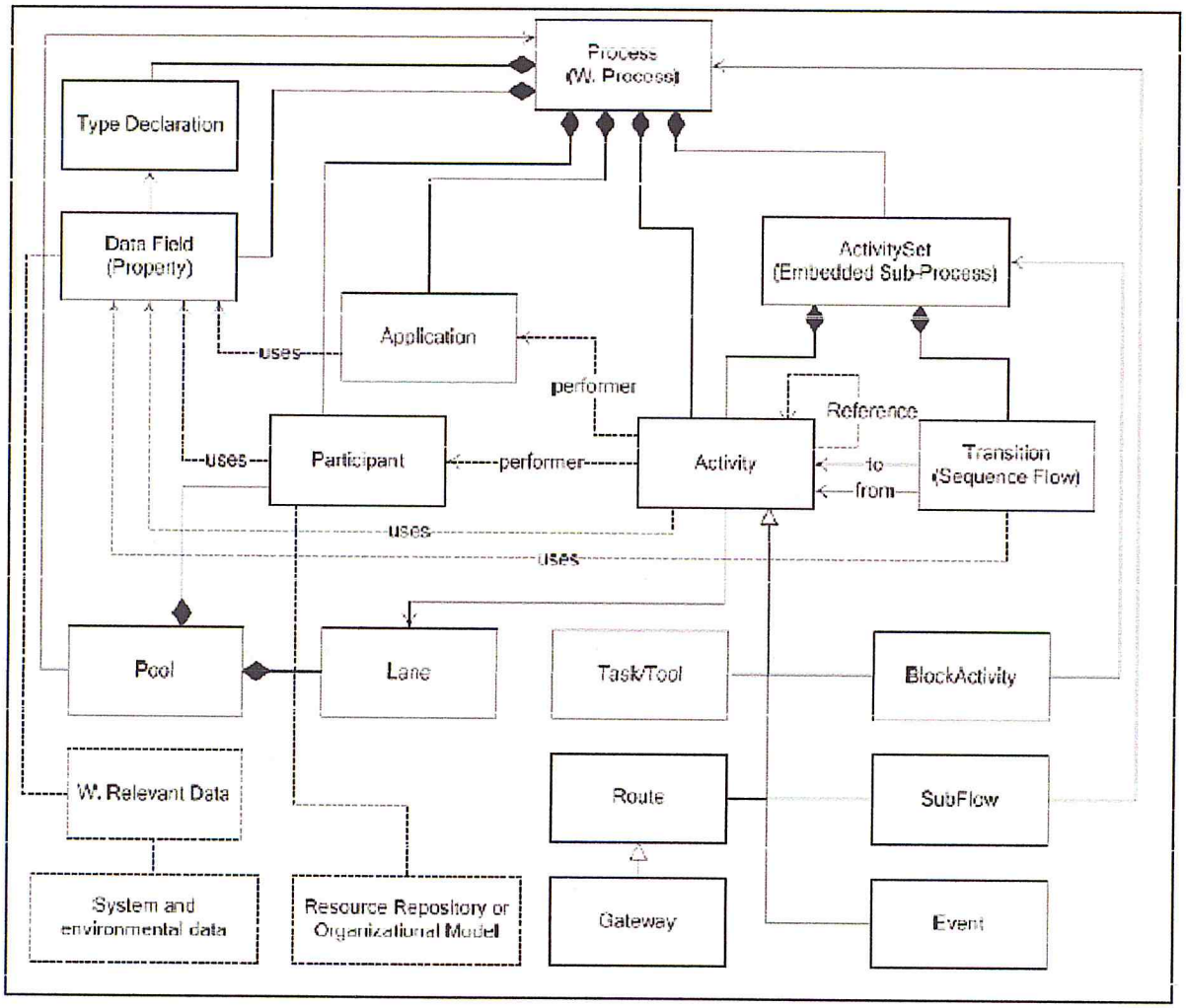


Figure III.5: *Meta modèle de XPDL (Haiping,2007)*

Comme le montre la ***Figure III.5***, un méta-modèle de processus comprend plusieurs éléments, par exemple, activités, participants, applications, transitions et champs de données, etc. Nous ignorons certains éléments qui ne sont pas liés à nos préoccupations, par exemple, les applications, les champs de données et autres attributs étendus de divers fournisseurs d'outils, etc. Dans cette section, nous présentons la sémantique relative au mappage en trois aspects, à savoir les activités, les liens et les structures de routage.

2.1. Equivalence de XPDL et réseau de Petri

2.1.1. Package

C'est l'élément qui contient toutes les autres éléments de XPDL, il sera tout simplement mappé en un réseau de Petri muni de deux places, une jouera le rôle de la place de départ, elle va contenir le premier marquage du réseau, la deuxième sera la place de fin.

2.1.2. Les activités

Dans le cas général, les activités sont mappées en une place et une transition relié par un arc de la place vers la transition, voir la ***Figure III.6*** (a).

Une activité initiale est une activité qui aucun antécédent, dans ce cas elle sera mappée en un arc sortant de la place de début allant vers une transition pour que le place de début généré par la transformation du Package sera la source de son arc, ***Figure III.6*** (b).

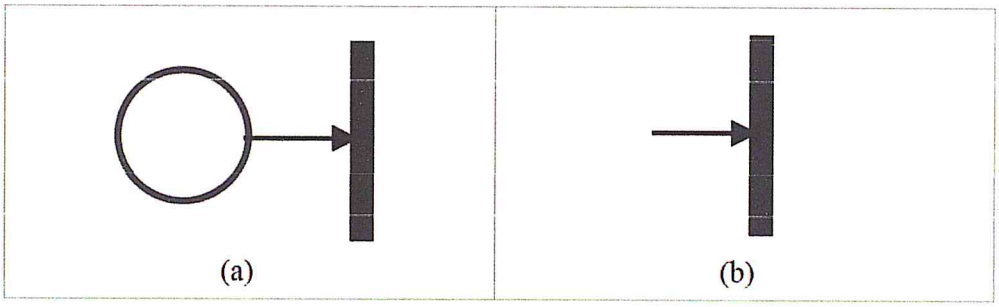


Figure III.6: *Représentation d'une activité en réseau de Petri dans le cas général.*

Une activité qui a multiple liens entrants et sortants au même temps et qui n'est pas muni d'une structure de routage est un cas complexe qui n'est pas traité dans les travaux mentionné dans la section précédente. Une activité à multiple entrées et sorties se comporte comme une structure de routage XOR-join (fusion exclusive) avec les entrées et comme une structure de routage AND-split (division inclusive) avec les sorties. Cette dernière va être mappée en ce qui est équivalent de mapper deux activités successives, où la première aura plusieurs entrées et une seul sortie et la deuxième aura une seule entrée et plusieurs sorties. De cette manière nous allons assurer un comportement équivalent et la possibilité de mapper cette activité.

Les structures de routage peuvent causer un changement dans le mapping des activités voisine, cela va être discuté dans 2.1.4.

2.1.3. Les liens

Dans le cas général, les liens seront mappés en des arcs. Ces arcs vont relier la transition de l'activité source avec la place de l'activité destination. Les structures de routage peuvent causer un changement sur le mapping des liens voisins, cela va être discuté dans 2.1.4.

2.1.4. Les structures de routage

Dans un modèle de processus XPDL, il existe des structures de routage explicites autres que des liens. Chaque activité peut avoir deux sortes de types de jointure et de partage (join et split), c'est-à-dire AND-join, XOR-join, AND-split et XOR-split, comme le montre la figure 7 (a).

Cependant, dans un réseau de Petri, il n'y a pas de structure correspondante. Un réseau de Petri implémente la structure de routage grâce à la connexion correcte entre les transitions et les places, comme le montre la figure 7 (b).

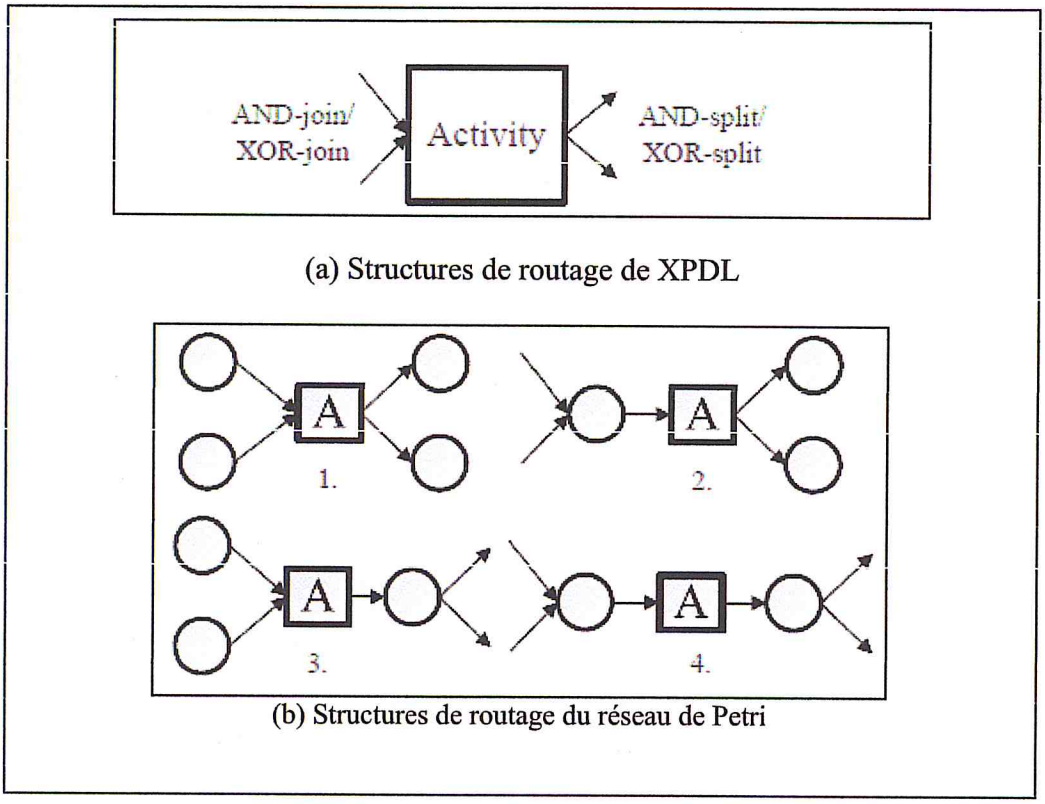


Figure III.7: Les structures de routage.

- ✓ **AND-join (fusion inclusive):** ne s'exécute que si toutes les entrées sont activée en parallèle.
- ✓ **XOR-join (fusion exclusive):** s'exécute dès qu'une des entrées est activée.
- ✓ **AND-split (division inclusive):** active toutes ses sorties en parallèle.
- ✓ **XOR-split (division exclusive):** active une seule sortie à la fois.

Afin de mapper les structures AND-split et XOR-join, il suffit de mapper les activités dont ses dernières sont contenu. Par contre, les structures AND-join et XOR-split seront mappés différemment.

Les travaux mentionnés dessus se met d'accord que AND-join doit être mappé en une place allant vers une transition par élément entrant, voir ***Figure III.3*** (a). Mais en pratique, il n'est pas pratique de mapper un élément en un nombre d'élément inconnu qui dépend du nombre d'entrées. Afin de résoudre ce soucis, nous avons opté à mapper AND-join en transition

seulement, puis mapper les liens entrant en deux arcs ayant une place entre eux, voir ***Figure III.10.***

Juste comme mentionné dans les travaux dessus, XOR-split sera mappée en une place allant vers une transition allant vers une place et les éléments sortant de cette dernière vont tous sortir de la dernière place, voir la partie noire de la ***Figure III.8*** (b). Cela va causer un changement sur le mapping des activités et des liens qui suivent, une activité sera mappée en une transition seulement (la partie rouge de la ***Figure III.8*** (b)) et un lien sera mappé en un arc de allant d'une place (la dernière de XOR-split) à une transition (l'activité destination), voir la partie bleue de la ***Figure III.8*** (b).

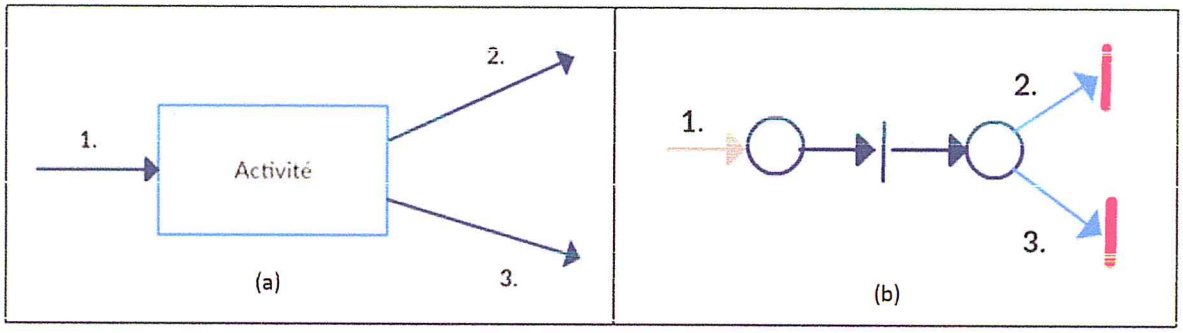


Figure III.8 *Transformation de la structure de routage XOR-split.*

2.2. Algorithme

Nous avons présenté le mapping et maintenant nous allons expliquer comment appliquer ce mapping et le fonctionnement de notre algorithme.

Notre algorithme consiste à prendre chaque élément du modèle XPDL et lui appliquer un ensemble de testes afin de finalement déduire son équivalent en réseau de Petri. Nous avons utilisé un organigramme pour expliquer le fonctionnement de notre algorithme. Notre organigramme est divisé en trois sous-organigrammes qui représentent le cas général, le cas d'une activité et le cas d'une transition.

2.2.1. Cas général

L'algorithme vérifier si le type du prochain élément à être transformer, si l'élément est un package, ce dernier va être transformé en réseau de Petri et générer deux places: une place servira comme la place de départ contenant le marquage initiale de notre réseau de Petri et l'autre servira comme une place de fin, voir ***Figure III.9***. Sinon il passe à vérifier si c'est une activité.

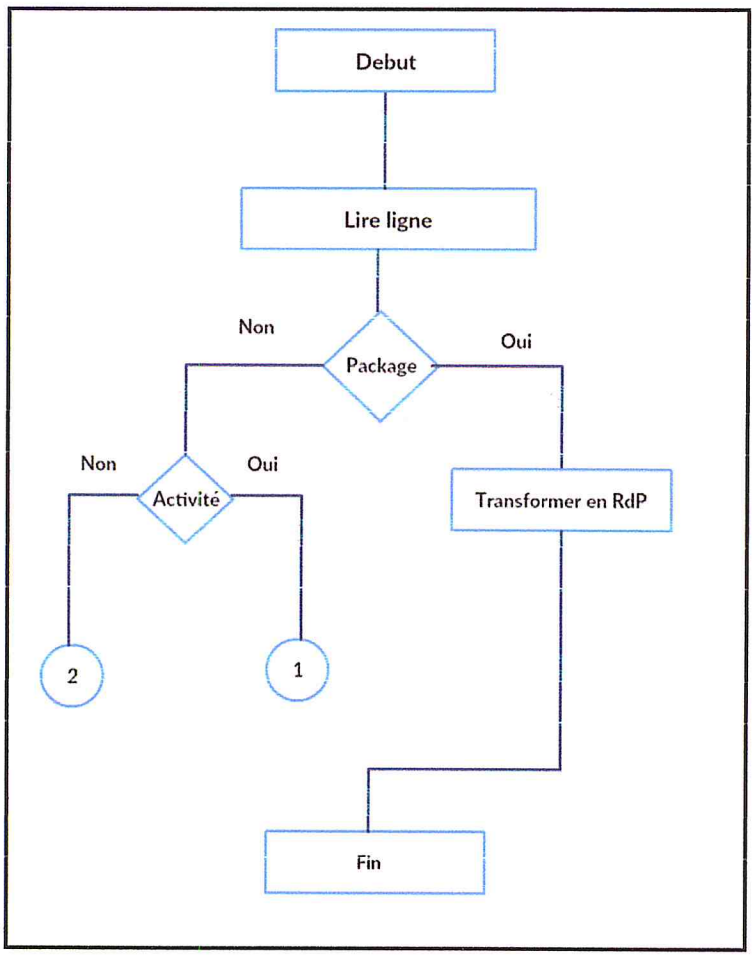


Figure III.9 *Organigramme général de l'algorithme de transformation de XPDL en réseau de Petri.*

2.2.2. Activités et Structures de routage:

Nous avons groupé les activités et les structures de routage parce qu'une structure de routage est en réalité une activité routée. L'algorithme à ce niveau continu à tester pour déterminer finalement en quel élément de réseau de Petri cette activité ou structure de routage doit être transformée, voir figure 10.

Nous avons utilisé deux abréviations dans cet organigramme et voici leur signification:

- ✓ **MM**: L'activité à Multiple entrées et Multiple sorties
- ✓ **SRC:XOR:** L'activité à une structure de routage XOR (join ou split) comme antécédent.

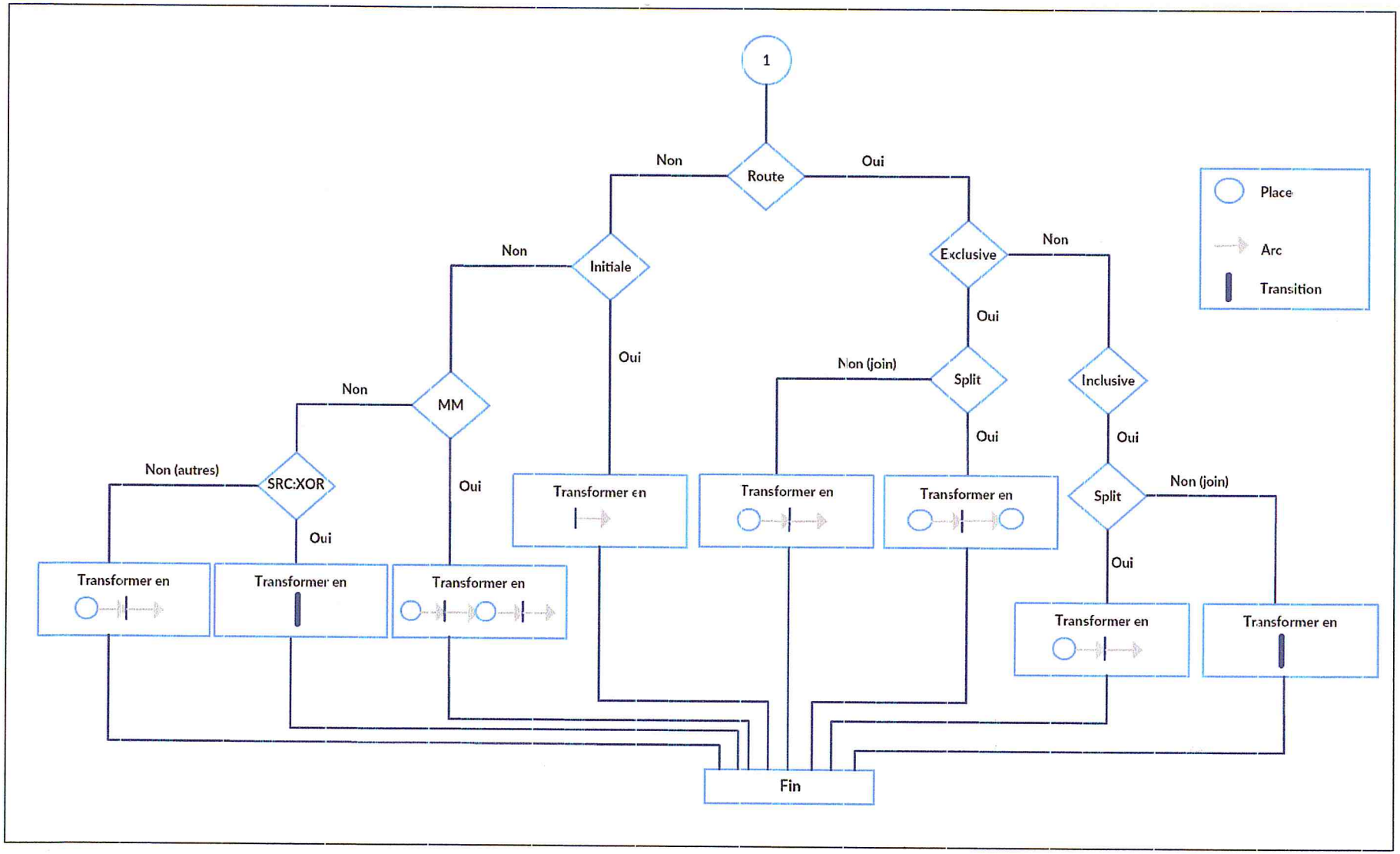


Figure 10: Organigramme représentant la transformation des activités et structures de routage.

2.2.3. Liens:

Si ce n'est pas une activité alors c'est certainement un lien, l'algorithme va donner un équivalent en réseau de Petri au lien selon plusieurs critères, voir figure 11.

Nous avons utilisé quelques abréviations dans cet organigramme et leurs signification est comme suit:

- ✓ **Src:XOR/MU:** A une structure de routage XOR (join ou split) ou une activité non routée à Multiples entrées et Unique sortie comme antécédent.
- ✓ **Des:MM:** Est l'antécédent d'une activité non routé a Multiple entrées et Multiple sorties.
- ✓ **Des:AND/UM:** Est suivi par une structure de routage AND (join ou split) ou une activité non routée avec une Unique entrée et Multiple sorties.
- ✓ **Des:XOR/MU:** Est suivi par une structure de routage XOR (join ou split) ou une activité non routée à Multiples entrées et Unique sortie comme antécédent.

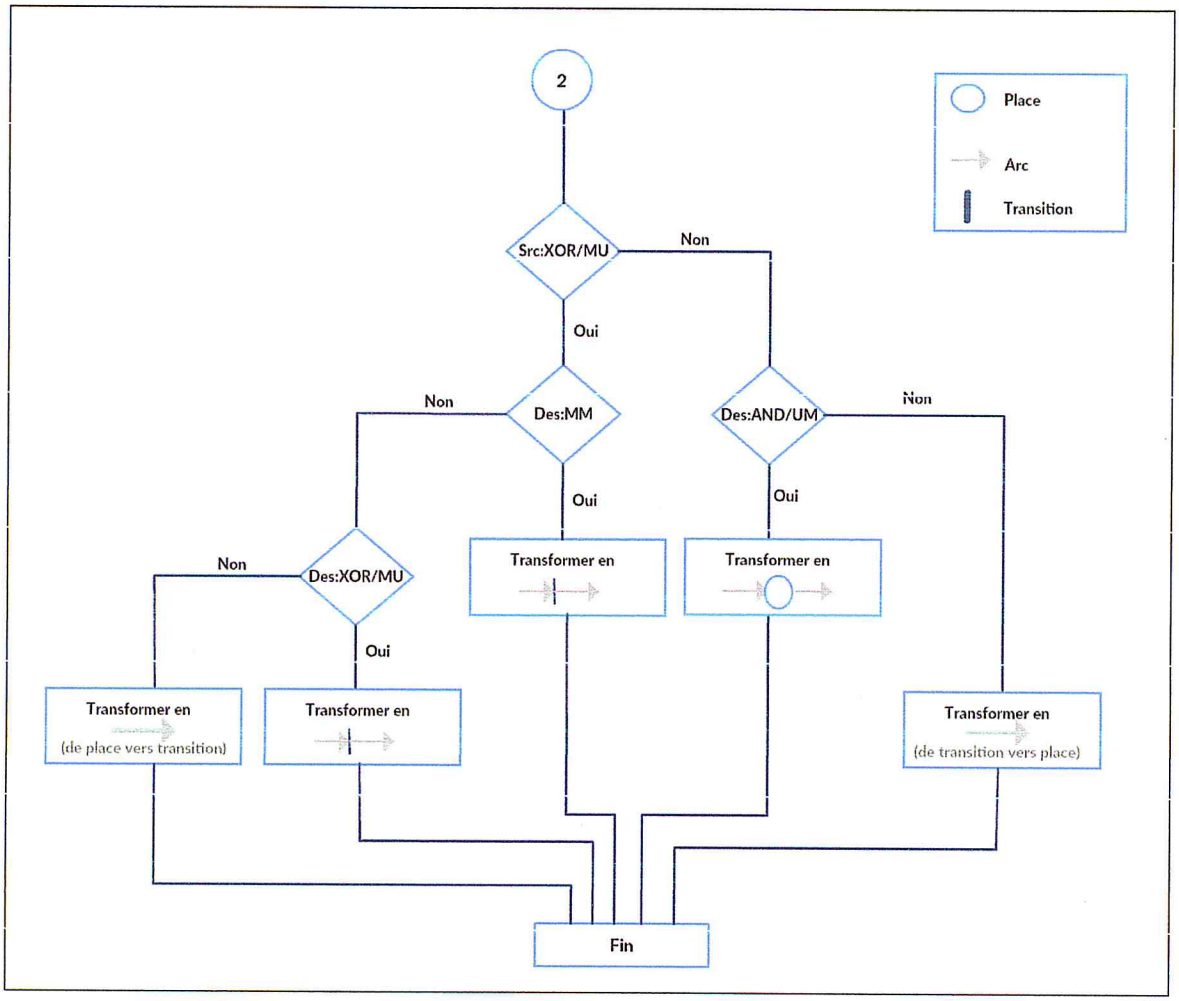


Figure III.11: Organigramme représentant la transformation des transitions.

3. Conclusion

Cette section vise à transformer automatiquement les modèles de processus XPDL en réseau de Petri avec une structure et des comportements équivalents. Nous avons présenté la sémantique du mapping ainsi que l'algorithme de transformation.

Il nous reste maintenant une dernière étape qui est l'implémentation des règles de mapping et de l'algorithme présenté dans cette section, c'est ce que nous allons voir dans la section suivante.

IV. Implémentation de la transformation de XPDL en réseau de Petri

1. Introduction

Dans le domaine de la transformation des modèles il existes à nos jours une diversité de langages et approches avec différents paradigmes tel que ATL, Kermeta, epsilon, QVT, VIATRA, ETL,UMLRSDS…etc.

Des études comparatives ont été faites sur ces différentes approches de transformation des modèles, on se basant sur plusieurs critères d'évaluation et caractéristiques de qualités pour choisir la meilleure approche. Dans ce qui suit nous allons présenter les différentes approches et langages de transformation des modèles et les travaux faits concernant l'évaluation de ces approches.

2. Présentation des langages et approches de transformation

- ✓ **Queryview transformation (QVT) :**

QVT est une approche développée par l'OMG le langage a une nature hybrid. Il dispose de d'une partie déclarative devisé aussi en deux langages, relationnel et Core. QVT a aussi un style de spécification de langage impératif et appelé aussi opérationnelle ce dernier peut être utilisé pour l'implémentation des règles qui ne sont pas applicable dans le langage relationnel.(Rahimi, 2013)

- ✓ **Kermeta :**

Kermeta est un langage puissant basé sur Basé sur un paradigme de méta-modélisation exécutable orienté objet, orienté model, orienté aspect et impératif. Le méta-model est définit dans le langage plus-tôt que dans un fichier Ecore. Kermeta permet la navigation et la manipulation d'un modèle de manière orientée objet avec une syntaxe fortement influencée par Eiffel. Kermeta est construit comme une extension à EMOF il décrit la structure et le comportement de model.(Farhana Islam, 2013)

- ✓ **Atlas transformation language (ATL) :**

L'Atlas Transformation Language (ATL) est un langage de transformation de modèles accompagnéhybride d'une boite à outils (toolkit), initialement proposé par le groupe de recherche ATLAS INRIA & LINA. il permet aux utilisateurs de définir la transformation dans les deux styles déclaratif et impératif, Dans le domaine de l'ingénierie dirigée par les modèles (IDM), ATL fournit aux développeurs un moyen de spécifier la façon de produire un certain nombre de modèles cibles à partir d'un ensemble de modèles de source (Ramdan,

2016).

- ✓ **VIATRA**

VIsualAutomated model TRAnsformation(VIATRA) est un outil de transformation développé à l'université de technologie et d'économie de Budapest, le but est de développer un langage qui va prendre en charge le cycle de vie d'ingénierie de la transformation des modèles notamment la spécification, le design, l'exécution, la validation et la maintenance. VIATRA fournit des moyens généraux pour spécifier et mettre en œuvre diverse transformation entre modèles.(Rahimi, 2013)

- ✓ **UML-RSDS :**

(UML-RSDS)The Reactive System Development Support est une approche de développement de logiciel dirigé par model développée au *king'scollege* de Londres, ce
Le langage est basé sur UML et MOF, et fournit les notations de spécification nécessaires pour la définition de la transformation du modèle, à différents niveaux d'abstraction. En outre, il existe une sémantique formelle établie pour des parties de langage pour soutenir la vérification et la synthèse du code exécutable. (Rahimi, 2013)

3. Travaux existants

- ✓ **Les travaux de « Farhana et Md» :**

(Farhana Islam, 2013) Propose une étude comparative entre des approches de transformation des Modèles kermeta et ATL, il a évalué les deux approches selon plusieurs critères d'évaluation notamment la courbe d'apprentissage, le cout, Expressivité…etc. Il aussi évaluée selon les *softgoals* de programmeur, manager, l'analyste et l'architecte de logiciel. Puis il a vérifié laquelle parmi les deux approches satisfait les quatre partenaires. Cette étude a montré que ATL et mieux que kermeta.

- ✓ **Les travaux de «Rahimi et Shekoufeh» :**

(Rahimi, 2013) a fait une étude comparative entre les Cinq approches de transformation de modèles mentionnés dessus, les cinq approches ont été évalués selon deux simples problèmes de raffinement et de réexpression, la différence significative entre les approches concernes

leurs paradigmes de spécification et leurs niveaux d'abstraction, le résultat générale dans le terme de caractéristiques ISO/IEC 9126 suggère que kermeta et ATL peuvent être des candidats pour implémenter ces types de transformation et il ont constaté que ATL peut générer une effective solution pour les cas les plus complexe.

4. Synthèse

Selon les travaux faites concernant la comparaison entres les approches de transformation des modèles et le choix de la meilleur entre eux, en suivant plusieurs critères d'évaluations et en effectuant plusieurs études de cas, nous avons constaté que ATL est le meilleur en plusieurs critères et le critères et le plus important est la possibilité de générer des solution pour les cas complexes, et c'est le cas dans notre travail, d'où nous choisissons ATL pour réaliser la transformation de modèle XPDL en réseaux de Pétri.

5. Présentation de l'ATL

L'Atlas Transformation Language (ATL) est un langage de transformation de modèles accompagné d'une boite à outils (toolkit), initialement proposé par le groupe de recherche ATLAS INRIA & LINA. Dans le domaine de l'ingénierie dirigée par les modèles (IDM), ATL fournit aux développeurs un moyen de spécifier la façon de produire un certain nombre de modèles cibles à partir d'un ensemble de modèles de source (ATLAS-INRIA et LINA, 2006).

Le langage ATL est hybride regroupe à la fois les paradigmes de programmations déclarative et impérative. La partie déclarative permet de faire correspondre directement un élément du méta modèle source de la transformation avec un élément du méta-modèle cible de la transformation.

La partie impérative d'ATL complète ces correspondances directes entre éléments. Elle comporte des déclarations conditionnelles, des déclarations de variables, des déclarations de boucle, etc.

Il définit deux types de règles de transformation : ***les règles déclaratives*** et ***lesrèglesappelées.***

Règles déclaratives (Matchedrules):

Les règles déclaratives permettent de faire correspondre certains éléments du modèle source et de générer à partir de ces éléments certains éléments du modèle cible.

Règles appelées(Calledrules):

Les règles appelées permettent de créer explicitement un ensemble d'éléments du modèle cible à partir de code impératif.

Le langage ATL a des types et des expressions qui sont basées sur l'Object ConstraintLanguage (OCL) de l'OMG. Un programme de transformation ATL ou *modules* se compose principalement d'entête *(header)*, de *helpers* et de règles de transformation *(transformationrules)* (Jouault et Kurtev, 2005).

-Le *header* définit le nom du module et les modèles d'entrées/sorties.

- Les *helpers* ressemblent à des fonctions ou des méthodes qui peuvent être appelées dans des règles de transformation ou bien par d'autres *helpers*. Chaque *helper* doit impérativement avoir une valeur de retour.
-Les règles définissent comment les modèles d'entrée sont transformés en modèles cibles. Elles forment l'élément de base dans ATL (Jouault et Kurtev, 2005).

ATL est accompagné d'un ensemble d'outils construits autour de la plateforme Eclipse, appelés ADT (ATL Development Tools). Cette boite à outil est composée du moteur ATL de transformation (*Engine block*) et d'un environnement de développement intégré ATL qui comporte plusieurs utilitaires comme les éditeurs dédiés, les débogueurs, la coloration syntaxique, etc.

6. Architecture technique générale

L'implémentation de notre approche est divisée en trois parties. Le but de la **premièrepartie** est de nettoyer le modèle XPDL du processus métierpour avoir un modèle conforme au méta-modèle qui contient seulement les éléments XPDL relatifs au comportement de processus métier. La **deuxième partie**prend en entrée le modèle résultant de la première partie et le transforme en un modèle RdP à l'aide desrègles de transformation.Nousavons utilisé Atlas Transformation Language (ATL) pour implémenter ces règles.

La **troisième parties**'intéresse à la conversion modèle RdP (RdP.xmi) résultant de la deuxiemeparite en un modèle RdP (RdP.tpn).cette partie a été réalisé avec java.La ***Figure III.12*** présente l'architecture technique de notre approche.

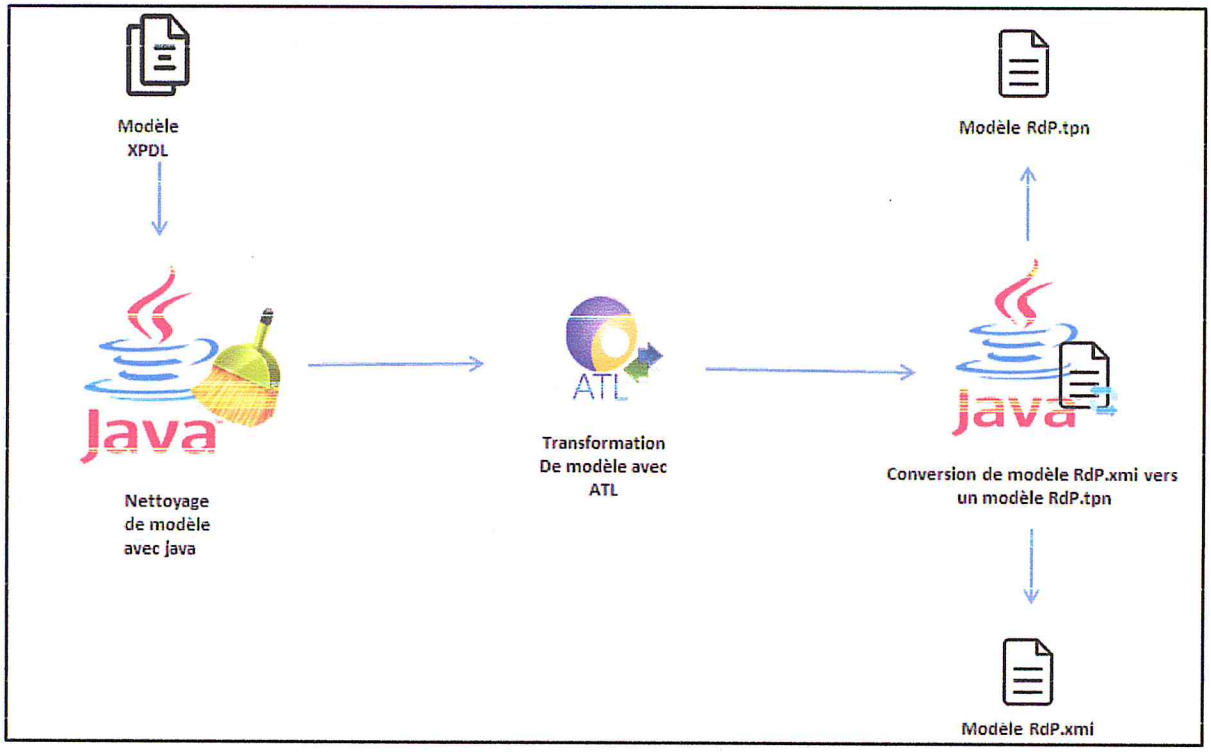


***Figure III.12** l'architecture technique générale de l'application.*

Dans ce qui suit nous allons présenter de chacune des parties de notre approche.

6.1. Nettoyage de Modèle XPDL

Dans cette étape nous allons nettoyer le modèle XPDL on éliminant les éléments XPDL qui n'ont pas d'influence sur le comportement de processus métiers ce qui nous donne un modèle de processus métier conforme au méta-modèle après la phase de nottoyage.étapes a été réalisé avec java et le modèle résultant va être transformer avec ATL en modèle RdP.

6.2. La transformation du modèle XPDL en un modèle RdP

Un des objectifs de notre travail est de transformer le modèle XPDL de processus métiers en un modèle RdP. La transformation prend en entrée le modèle XPDL nettoyé dans l'étape précédente et produit en sortie un modèle de processus métiers en RdP. Ce dernier va être conforme à notre méta-modèle RdP. La réalisation de cette transformation nécessite en premier lieu de définir des règles de transformation que nous allons présenter en détail dans cette section. La ***Figure III.13*** met en évidence la transformation du modèle XPDL en modèle RdP.

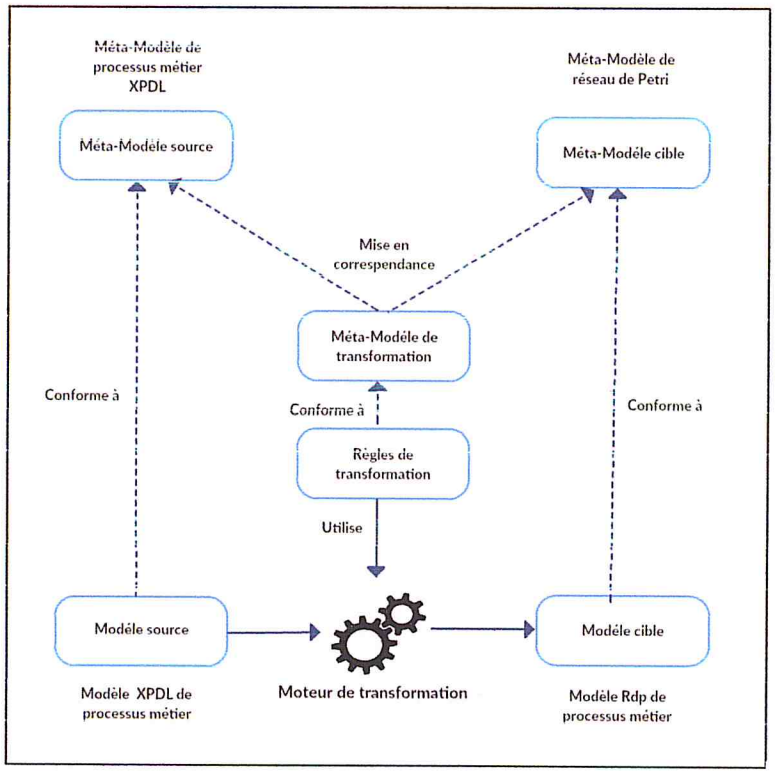


Figure III.13 La transformation de modèle XPDL vers modèle RdP.

6.2.1. La description des règles de transformation

La transformation de modèles nécessite l'identification d'un certain nombre de règles basées sur le mapping présenté dans la section précédente. Ces règles décrivent comment transformer un concept source vers un concept cible. Le travail sur les règles de transformation correspond finalement au « cœur applicatif » de notre travail. Nous définissions les règles de transformation avec le langage de transformation ATL (Atlas Transformation Language) comme suit:

- ✓ **Règle R1 :**«transformPackage»

Le package est un élément du processus métiers XPDL. Il est équivalent à l'élément réseau de Petri du modèle RdP. Cette règle transforme le package en un élément PetriNet plus une place initiale (source) et une place finale (puits).

```
XPDL!Package.allInstances();
ruletransformPAckage{
        frompackage:XPDL!Package

        to
        petri:PetriNet!PetriNet(
                Id<-package.Id,
                Name<-package.Name),
        placeInit:PetriNet!Place(Id<-'place_initiale'),
        placeFinale:PetriNet!Place(Id<- 'place_finale')

 }
```

Figure III.14 *Description de Règle 1 de transformation en ATL.*

✓ **Règle R2 :**«transformXORsplitActivity»

Cette règle permet de transformer les activités qui possèdent une structure de routage de type XOR split en cinq éléments de réseau de Petri :

- Une place.
- Un arc de place vers transition.
- Une transition.
- Un arc de transition vers place.
- Une place.

```
ruletransformXOrSplitActivity{
            fromxorAct:XPDL!Activity(
ifthisModule.isInitial(xorAct)=falseandthisModule.isFinal(xorAct)=false
                  andxorAct.HasRoute.toString()='true'then
                  ifthisModule.isExclusive(xorAct.Route)    then
                  true
                  else
                        false
                        endif
                  else
                        ifthisModule.hasMultipleInputs(xorAct)=truethen
                        trueelsefalseendif-- ici  on transform les activity ayant
plusieurs arc entrants
                        endif
            )
            to
            place:PetriNet!Place(Id<-xorAct.Id
            ),
            placeToTransArc:PetriNet!PlaceToTransArc(
                  Id<-xorAct.Id,
                  Source<-place,
                  Target<-trans),
            trans:PetriNet!Transition(
                  Id<-xorAct.Id,
                  Name<-xorAct.Name),
            TransToPlaceArc:PetriNet!TransToPlaceArc(
                  Id<-xorAct.Id,
                  Source<-trans,
                  Target<-place1),
            place1:PetriNet!Place(Id<-xorAct.Id+'place1'
            )}
```

Figure III.15 *Description de Règle 2 de transformation en ATL.*

✓ **Règle R3 :**«transformAndJoinActivity»

Cette règle permet de transformer les activités qui possèdent une structure de routage de type AND join en une transition de réseau de Petri.

```
ruletransformAndJoinActivity{
      fromandAct:
XPDL!Activity(ifthisModule.isInitial(andAct)=falseandthisModule.has
MultipleInputs(andAct)=false

      andthisModule.isFinal(andAct)=falseandandAct.HasRoute.toStrin
g()='true'then
                  ifthisModule.isInclusive(andAct.Route)=truethen
                  true
                  else
                        false
                        endif
                  else
                        false
                        endif
            )
      to
      trans:PetriNet!Transition(
            Id<-andAct.Id,
            Name<-andAct.Name)
}
```

Figure III.16 Description de Règle 3 de transformation en ATL.

✓ **Règle R4 :**«transformActivityComingFromXOR»

Cette règle permet de transformer les activités qui possèdent un prédécesseur de de type XORjoin en une transition de réseau de Petri.

```
      ruletransformActivityComingFromXOR{

      fromXORact:XPDL!Activity(ifthisModule.isInitial(XORact)=falseand
                  thisModule.isFinal(XORact)=falseand
                  thisModule.hasMultipleInputs(XORact)=falseand
                  thisModule.isComingFromXorActivity(XORact)=truethen
                  ifXORact.HasRoute.toString()='false'then
                  true
                  else
                        false
                        endif
                  else
                        false
                        endif)
            to
            trans:PetriNet!Transition(
                  Id<-XORact.Id,
            Name<-XORact.Name)
      }
```

Figure III.17 Description de Règle 4 de transformation en ATL.

✓ **Règle R5 :**«transformActivityInitiales»

Cette règle permet de transformer les activitésinitiales (les activités qui ne possèdent pas desprédécesseurs) en deux éléments de réseau de Petri :

- Un arc de place vers transition.
- Une transition.

```
ruletransformActivityInitiales{
      from  act :XPDL!Activity (thisModule.isInitial(act)=trueand
            thisModule.isFinal(act)=falseand
            thisModule.hasMultipleInputs(act)=false )
      to
      trans: PetriNet!Transition(
            Id<-act.Id,
            Name<-act.Name)
            ,
      placeTotran :PetriNet!PlaceToTransArc(
            Id<-act.Id,
            Source<-thisModule.getPackage-
>collect(b|thisModule.resolveTemp(b, 'placeInit')),
            Target<-trans)
   }
```

Figure III.18 *Description de Règle 5 de transformation en ATL.*

✓ **Règle R6 :**«transformActivityFinales»

Cette règle permet de transformer les activités finales (les activités qui ne possèdent pas des successeurs) en quatre éléments de réseau de Petri :

- Une place.
- Un arc de place vers transition.
- Une transition.
- Un arc de transition vers place.

```
ruletransformActivityFinales{
from  act :XPDL!Activity
(thisModule.isFinal(act)=trueandthisModule.hasMultipleInputs(act)=false)
to
      place3: PetriNet!Place(Id<-act.Id),
      placeTotran :PetriNet!PlaceToTransArc(
            Id<-act.Id,
            Source<-place3,
            Target<-trans),
      trans: PetriNet!Transition(
            Id<-act.Id,
            Name<-act.Name)
            ,
      tranToplace :PetriNet!TransToPlaceArc(
            Id<-act.Id,
            Source<-trans,
            Target<-thisModule.getPackage-
>collect(m|thisModule.resolveTemp(m, 'placeFinale')))
}
```

Figure III.19 *Description de Règle 6 de transformation en ATL*

✓ **Règle R6 :**«transformActivity»

Cette règle permet de transformer les activités simples (les activités qui ne vérifient pas les conditions des règles précédentes) en trois éléments de réseau de Petri :

- Une place.
- Un arc de place vers transition.
- Une transition.

```
rule transformActivity{
      from  act :XPDL!Activity
(thisModule.isInitial(act)=falseandthisModule.isFinal(act)=false
andact.HasRoute.toString() = 'false'
      andthisModule.isComingFromXorActivity(act)=false
andthisModule.hasMultipleInputs(act)=false)
to
      place2 :PetriNet!Place(
            Id <-act.Id
            ),
      trans: PetriNet!Transition(
            Id<-act.Id,
            Name<-act.Name)
            ,
      placeTotran:PetriNet!PlaceToTransArc(
            Id<-act.Id,
            Source<-place2,
            Target<-trans)
}
```

Figure III.20 Description de Règle 6 de transformation en ATL.

✓ **Règle R7 :**«transformtransBetweenXOR»

- Cette règle permet de transformer les transition XPDL qui sont entre deux structures de routage de type XOR en trois éléments de réseau de Petri : Un arc de transition vers place, Une place et Un arc de place vers transition.

```
ruletransformTransBetweenXOR{

      fromtr:XPDL!Transition(thisModule.isTransComingFromXORRoute(tr)=tru
eandthisModule.isTransGoingToXORRoute(tr)=true)
            to
            placeToTransArc:PetriNet!PlaceToTransArc(Id<-tr.Id,
      Source<-(thisModule.allActivities->collect(e |
thisModule.resolveTemp(e, 'place1') )) ->any(c |
ifc.oclIsUndefined()thenfalseelsec.Id= tr.From+'placei'endif),
      Target<- transi)
                  ,
            transi:PetriNet!Transition(Id<-tr.Id,
Name<- 'transition intermediaire XOR-XOR'),
            transToPlaceArc:PetriNet!TransToPlaceArc(Id<-tr.Id,
                  Source<-transi,
                  Target<-(thisModule.allActivities->collect(e |
thisModule.resolveTemp(e, 'place') ))
                  ->any(c | ifc.oclIsUndefined()thenfalseelsec.Id=
tr.Toendif))
      }
```

Figure III.21 Description de Règle 7 de transformation en ATL.

✓ **Règle R8 :**«transformtoAndTransition»

Cette règle permet de transformer les transitions XPDL qui ont un élément XPDL cible de type AND en trois éléments de réseau de Petri :

- Un arc de transition vers place.
- Une place.
- Un arc de place vers transition.

```
ruletransformToAndTransition{
            fromtr :XPDL!Transition(
(thisModule.isTransToAndRoute(tr)=true)
andthisModule.isTransComingFromXORRoute(tr)=false  )
            to
place: PetriNet!Place(Id<-tr.Id),
                    transToplac:PetriNet!TransToPlaceArc(
                          Id<-tr.Id,
                          Source<-(thisModule.allActivities->collect(e |
thisModule.resolveTemp(e, 'trans')))->select(c |
ifc.oclIsUndefined()thenfalseelsec.Id= tr.Fromendif),
      Target<-place),
      placeTotran:PetriNet!PlaceToTransArc(
                          Id<-tr.Id,
Target<-(thisModule.allActivities->collect(e | thisModule.resolveTemp(e,
'trans')))->select(c | ifc.oclIsUndefined()thenfalseelsec.Id= tr.Toendif))
}
```

Figure III.22 *Description de Règle 8 de transformation en ATL.*

✓ **Règle R9 :**«transformtoAndTransition»

Cette règle permet de transformer les transitions XPDL simple (les transitions qui ne vérifient pas les conditions des autres règles) en un arc de transition vers place.

```
ruletransformTransition{
      from  t
:XPDL!Transition((thisModule.isTransToAndRoute(t)=false)andthisModule.isTransComi
ngFromXORRoute(t)=false  )
to
a: PetriNet!TransToPlaceArc(
            Id<-t.Id+'abbas',
          Source<- (thisModule.allActivities->collect(e |
thisModule.resolveTemp(e, 'trans') ))->any(k |
ifk.oclIsUndefined()thenfalseelsek.Id= t.Fromendif),
          Target<-(((((thisModule.allActivities->collect(e |
thisModule.resolveTemp(e, 'place'))).union(
            thisModule.allActivities->collect(b | thisModule.resolveTemp(b,
'place2') ))).union(
                  thisModule.allActivities->collect(f |
thisModule.resolveTemp(f, 'place1')))).union(
                        thisModule.allActivities->collect(p |
thisModule.resolveTemp(p, 'place3')))).union(thisModule.allActivities->collect(e
| thisModule.resolveTemp(e, 'place5'))))->any(m | ifm.oclIsUndefined()
            thenfalseelsem.Id= t.Toendif))
}
```

Figure III.23 *Description de Règle 9 de transformation en ATL.*

✓ **Règle R10 :**«transComingFromXor»

Cette règle permet de transformer les transitions XPDL qui ont antécédent de type XOR en un élément de réseau de Petri « arc de place vers transition ».

```
ruletransComingFromXor{
      fromtra:
XPDL!Transition(thisModule.isTransComingFromXORRoute(tra)=trueandthisModule.i
sTransGoingToXORRoute(tra)=falseandthisModule.isTransToAndRoute(tra)=false)
            to
            placeToTransArc:PetriNet!PlaceToTransArc(
                  Id<-tra.Id,
            Source<- (thisModule.allActivities->collect(e |
thisModule.resolveTemp(e, 'place1') ))->any(c |
ifc.oclIsUndefined()thenfalseelsec.Id= tra.From+'place1'endif),
          Target<- (thisModule.allActivities->collect(el |
thisModule.resolveTemp(el, 'trans')))->any(c |
ifc.oclIsUndefined()thenfalseelsec.Id= tra.Toendif)
            )
    }
```

Figure III.24 *Description de Règle 10 de transformation en ATL.*

✓ **Règle R11 :**«transformeTransBetweenXORandAND»

Cette règle permet de transformer les transitions XPDL qui ont un prédécesseur de type XOR et un successeur de type AND en un élément de réseau de Petri « arc de place vers transition ».

```
ruletransformeTransBetweenXORandAND{
      fromtr:XPDL!Transition(thisModule.isTransComingFromXORRoute(tr)=trueandth
isModule.isTransToAndRoute(tr)=true)
            to
            placeToTransArc:PetriNet!PlaceToTransArc(
                  Id<-tr.Id,
                  Source<-(thisModule.allActivities->collect(e |
thisModule.resolveTemp(e, 'place1') ))->any(c |
ifc.oclIsUndefined()thenfalseelsec.Id= tr.From+'place1'endif),
                  Target<- (thisModule.allActivities->collect(el |
thisModule.resolveTemp(el, 'trans')))->any(c |
ifc.oclIsUndefined()thenfalseelsec.Id =  tr.Toendif))
}
```

Figure III.25 *Description de Règle 11 de transformation en ATL.*

7. Conversion de modèle RdP.

Cette étape permet de convertir le modèle de RdP (RdP.xmi) résultant de la partie précédente en un modèle RdP (RdP.tpn) pour qu'il soit compatible avec l'outil de vérification des modèles de processus métiers réalisé au sein de CDTA. Nous avons réalisé cette étape avec java en parcourant le modèle résultant de l'étape précédente et on réécrivant chacun de ses éléments dans un autre fichier (RdP.tpn) suivant la syntaxe de réseau de Petri textuel. ***Figure III.26*** représente la syntaxe de réseau de Petri textuel.

```
// net nom_fichier

net

// tr ti places_en_amant -> palces_en_aval

tr

// pl pi : {étiquette} (nombe_de_jeton) si réseau étiqueté
// pl pi (nombe_de_jeton) si réseau non étiqueté
```

Figure III.26*.Syntaxe de réseau de Petri textuel.*

8. Automatisation de la transformation

Dans le but d'automatiser la transformation de modèle de processus métier XPDL en modèle RdP, et suit aux étapes précédentes nous avons opté de créer une application web qui permet de faire la transformation via un navigateur web ce qui permet la disponibilité de notre application sur le Cloud de CDTA ou sur internet. Pour se faire nous avons choisir la technologie de développement web java EE vu qu'elle est disponible sur eclipse et la possibilité d'intégrer les autres parties réalisé avec java et ATL dans cette application.la figure suivante (***Figure III.27)*** montre l'interface principale de l'application web.

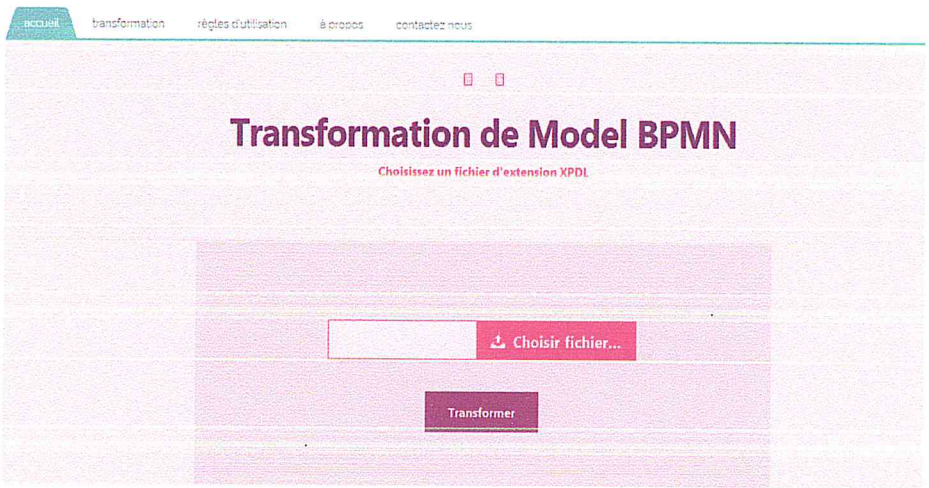


Figure III.27 *l'interface principale de l'application.*

Cette interface permet l'utilisateur d'importer le modèle XPDL en cliquant sur le bouton « choisir fichier » puis il peut lancer la transformation en cliquant sur « transformer ». Après

l'utilisateur va être redirigé vers une autre interface pour qu'il puisse télécharger les modèles résultants. La figure suivante (***Figure III.28***) montre l'interface de téléchargement.

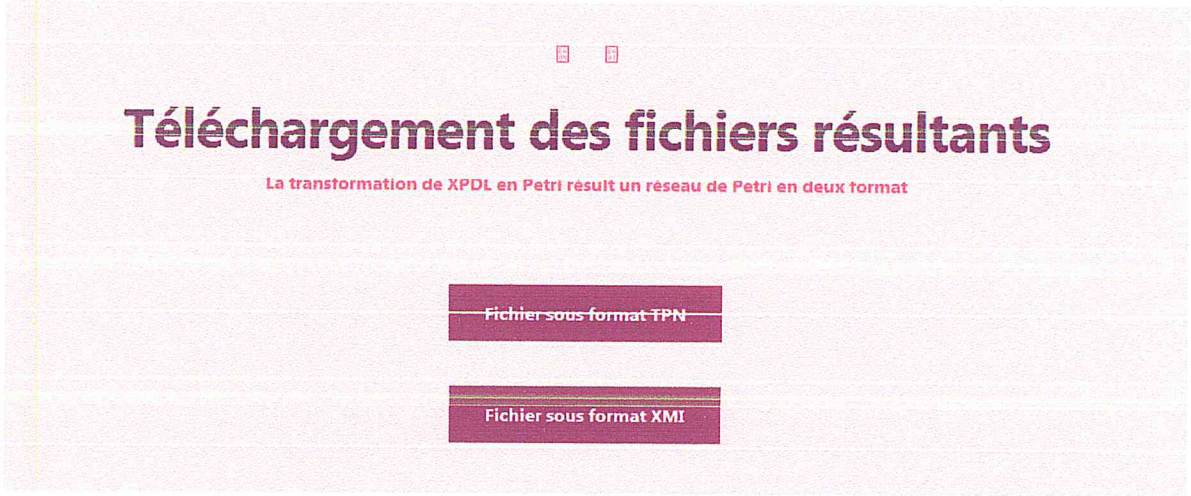


Figure III.28 *l'interface de téléchargement des fichiers résultants.*

L'utilisateur peut télécharger les deux modèle de RdP,(RdP.tpn) en cliquant sur le bouton « fichier sous format TPN » et (RdP.xmi) en cliquant sur « fichier sous format XMI ».

L'application contient également d'autre interfaces qui contient des informations sur l'application tel que les règles d'utilisations et des informations sur les le centre de développement des technologies avancés (CDTA).

9. Application d'un jeu de test

Dans le but de valider les notre approche de transformation que nous avons définies dans ce chapitre et de présenter le fonctionnement outils, nous allons appliquer dans ce qui suite un exemple de cas de processus métier « processus de sélection » transformé dans la phase précédente (de BPMN en XPDL). La figure suivante (***Figure III.29***) représente le processus de sélection XPDL.

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 Id="20dcf848-bdef-48d2-853f-098b3109add8"
Name="Selection Process" xmlns="http://www.wfmc.org/2009/XPDL2.2">
  <PackageHeader>
    <XPDLVersion>2.2</XPDLVersion>
    <Vendor>Bizagi Process Modeler.</Vendor>
    <Created>2011-07-19T17:20:59.0246536+01:00</Created>
    <ModificationDate>2013-10-04T17:26:40.648717+01:00</ModificationDate>
    <Description>Selection Process</Description>
    <Documentation />
  </PackageHeader>
  <RedefinableHeader>
    <Author>LauraG</Author>
    <Version>1.0</Version>
    <Countrykey>CO</Countrykey>
  </RedefinableHeader>
  <ExternalPackages />
  <Participants>
    <Participant Id="66e7b01f-5042-48e4-8d4c-0e0f3c867689" Name="Human Resources Analyst">
      <ParticipantType Type="ROLE" />
      <Description>The persons who belongs Human Resources area, he or she must  lead the selection process.</Description>
      <ExtendedAttributes>
        <ExtendedAttribute Name="Human_Resources_Analyst" />
      </ExtendedAttributes>
    </Participant>
    <Participant Id="8ead00db-b5ba-4c0e-969c-09910c078bca" Name="Requester">
      <ParticipantType Type="ROLE" />
      <Description>The persons who makes the personnel requisition. </Description>
      <ExtendedAttributes>
        <ExtendedAttribute Name="Requester" />
      </ExtendedAttributes>
```

Figure III.29 *Le processus de sélection XPDL.*

✓ Nettoyage de modèle

Dans cette étape nous allons importer ce modèle dans l'application puis lancer la transformation, après les éléments XPDL qui ne sont pas relatif au comportement de processus métiers vont être supprimés. La figure suivante (***Figure III.30)*** représente le modèle nettoyé.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><XPDL:Package Id="20dcf848-bdef-48d2-853f-098b3109add8"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:SchemaLocation="http://XPDL/1.0 XPDL.ecore"
xmlns:XPDL="http://XPDL/1.0"><WorkflowProcesses>
    <WorkflowProcess Id="afdf54ba-dac6-4026-90fb-8b5d30d872b2" Name="Main Process"><ActivitySets/></WorkflowProcess>
    <WorkflowProcess Id="ffbf8b6f-cef5-4630-8db0-70258a297fc5" Name="Selection Process"><ActivitySets/><Activities>
        <Activity HasRoute="false" Id="4e4d5a54-393f-481e-b545-e101015d8680" Name="">
          <Description/>
          <Event>
            <StartEvent/>
          </Event>
          <Documentation/>
          <NodeGraphicsInfos>
            <NodeGraphicsInfo>
              <Coordinates/>
            </NodeGraphicsInfo>
          </NodeGraphicsInfos>
          <ExtendedAttributes/>
        </Activity>
        <Activity HasRoute="false" Id="c5723c6e-2aa4-4fcc-81f0-95643ce3f5cd" Name="Entrer Test Result">
```

Figure III.30 *Le processus de sélection XPDL nettoyé.*

✓ Transformation de modèle

Cette partie prend en entrée le modèle nettoyé dans la partie précédente et le transforme en un modèle RdP (RdP.xmi).la figure (***Figure III.31)*** suivante montre le modèle de RdP (RdP.xmi) résultant de la transformation ATL.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:PetriNet="http://PetriNet/1.0">
  <PetriNet:PetriNet Id="20dcf848-bdef-48d2-853f-098b3109add8"/>
  <PetriNet:Place Id="place_initiale" outgoing="4e4d5a54-393f-481e-b545-e101015d8680"/>
  <PetriNet:Place Id="place_finale" incoming="078bd5aa-2189-4b00-a55e-55c4739ddb20 0337f3ba-44cd-4918-8746-138c858ef2ac"/>
  <PetriNet:Place Id="ed8e8ab2-082c-49b5-b85a-9be273847367" incoming="5a1cf906-5eb3-4934-9c9d-4e472588dd7aabbas" outgoing="ed8e
  <PetriNet:PlaceToTransArc Id="ed8e8ab2-082c-49b5-b85a-9be273847367" Source="ed8e8ab2-082c-49b5-b85a-9be273847367" Target="ed8
  <PetriNet:Transition Name="References ok?" Id="ed8e8ab2-082c-49b5-b85a-9be273847367" incoming="ed8e8ab2-082c-49b5-b85a-9be273
  <PetriNet:TransToPlaceArc Id="ed8e8ab2-082c-49b5-b85a-9be273847367" Source="ed8e8ab2-082c-49b5-b85a-9be273847367" Target="ed8
  <PetriNet:Place Id="ed8e8ab2-082c-49b5-b85a-9be273847367place1" incoming="ed8e8ab2-082c-49b5-b85a-9be273847367" outgoing="0ee
  <PetriNet:Place Id="ce1272a4-9c87-403a-9fa8-44f142dde6b9" incoming="fcedb9c8-dda2-4658-8fdd-44789d4dc477abbas" outgoing="ce12
  <PetriNet:PlaceToTransArc Id="ce1272a4-9c87-403a-9fa8-44f142dde6b9" Source="ce1272a4-9c87-403a-9fa8-44f142dde6b9" Target="ce1
  <PetriNet:Transition Name="Candidate Selected?" Id="ce1272a4-9c87-403a-9fa8-44f142dde6b9" incoming="ce1272a4-9c87-403a-9fa8-4
  <PetriNet:TransToPlaceArc Id="ce1272a4-9c87-403a-9fa8-44f142dde6b9" Source="ce1272a4-9c87-403a-9fa8-44f142dde6b9" Target="ce1
  <PetriNet:Place Id="ce1272a4-9c87-403a-9fa8-44f142dde6b9place1" incoming="ce1272a4-9c87-403a-9fa8-44f142dde6b9" outgoing="aef
  <PetriNet:Place Id="9e77f47d-d1a7-40e3-a766-dd82b938d3d1" incoming="a3e797b0-9af0-4d3f-b75e-28851a7dd90eabbas" outgoing="9e77
  <PetriNet:PlaceToTransArc Id="9e77f47d-d1a7-40e3-a766-dd82b938d3d1" Source="9e77f47d-d1a7-40e3-a766-dd82b938d3d1" Target="9e7
  <PetriNet:Transition Name="Other Interview?" Id="9e77f47d-d1a7-40e3-a766-dd82b938d3d1" incoming="9e77f47d-d1a7-40e3-a766-dd82
  <PetriNet:TransToPlaceArc Id="9e77f47d-d1a7-40e3-a766-dd82b938d3d1" Source="9e77f47d-d1a7-40e3-a766-dd82b938d3d1" Target="9e7
  <PetriNet:Place Id="9e77f47d-d1a7-40e3-a766-dd82b938d3d1place1" incoming="9e77f47d-d1a7-40e3-a766-dd82b938d3d1" outgoing="457
  <PetriNet:Place Id="87e2322e-0141-46d6-a6dd-cf4158b6f656" incoming="912eccd9-d495-4e0d-8ff5-6bcabca18dbdabbas" outgoing="87e2
  <PetriNet:PlaceToTransArc Id="87e2322e-0141-46d6-a6dd-cf4158b6f656" Source="87e2322e-0141-46d6-a6dd-cf4158b6f656" Target="87e
```

Figure III.31 *Le processus de sélection XPDL nettoyé.*

- ✓ **Conversion de modèle**

Cette partie prend en entrée le modèle nettoyé dans la partie précédente et le converti en un modèle RdP (RdP.tpn).la figure (***Figure III.32)*** suivante montre le modèle de RdP résultant de la conversion.

```
net SelectionProcess
tr InterviewScheduling f61be1c3-81ae-4e18-88b6-31d389355eba 87e2322e-0141-46d6-a6dd-cf4158b6f656place1 -> f61be
tr Referencesok? ed8e8ab2-082c-49b5-b85a-9be273847367 -> ed8e8ab2-082c-49b5-b85a-9be273847367place1
tr CandidateSelected? ce1272a4-9c87-403a-9fa8-44f142dde6b9 -> ce1272a4-9c87-403a-9fa8-44f142dde6b9place1
tr OtherInterview? 9e77f47d-d1a7-40e3-a766-dd82b938d3d1 -> 9e77f47d-d1a7-40e3-a766-dd82b938d3d1place1
tr SelectProcessDetails 34794110-2785-47ee-8e28-6b188b93583e ed8e8ab2-082c-49b5-b85a-9be273847367place1 ce1272a
tr ProcessContinues? 87e2322e-0141-46d6-a6dd-cf4158b6f656 -> 87e2322e-0141-46d6-a6dd-cf4158b6f656place1
tr OfferAccepted? 103d8206-9da3-41b6-9f6e-0199011d931b -> 103d8206-9da3-41b6-9f6e-0199011d931bplace1
tr MedicalExaminationok? 5d6bf6d5-a4ec-41b6-a3f1-2c0388859bf7 -> 5d6bf6d5-a4ec-41b6-a3f1-2c0388859bf7place1
tr CheckingReferences 9e77f47d-d1a7-40e3-a766-dd82b938d3d1place1 -> ed8e8ab2-082c-49b5-b85a-9be273847367
tr InformaboutInterviewScheduling 9e77f47d-d1a7-40e3-a766-dd82b938d3d1place1 -> 948876bf-3bbb-4163-8c4c-efa74d0
tr MedicalExamination 103d8206-9da3-41b6-9f6e-0199011d931bplace1 -> 5d6bf6d5-a4ec-41b6-a3f1-2c0388859bf7
tr SelectCandidates ed8e8ab2-082c-49b5-b85a-9be273847367place1 -> ce1272a4-9c87-403a-9fa8-44f142dde6b9
tr InformAboutDecision 5d6bf6d5-a4ec-41b6-a3f1-2c0388859bf7place1 -> 078bd5aa-2189-4b00-a55e-55c4739ddb20
tr OfferSalaryNegotiation ce1272a4-9c87-403a-9fa8-44f142dde6b9place1 -> 103d8206-9da3-41b6-9f6e-0199011d931b
tr Start_Event place_initiale -> 588457a1-2cb2-4e0f-85b5-5aadc4ffd62f
tr End_Event 078bd5aa-2189-4b00-a55e-55c4739ddb20 -> place_finale
tr End_Event_1 0337f3ba-44cd-4918-8746-138c858ef2ac 87e2322e-0141-46d6-a6dd-cf4158b6f656place1 -> place_finale
tr EntrerTestResult c5723c6e-2aa4-4fcc-81f0-95643ce3f5cd -> f61be1c3-81ae-4e18-88b6-31d389355eba
tr EnterInterviewResults 948876bf-3bbb-4163-8c4c-efa74d0efb17 -> f61be1c3-81ae-4e18-88b6-31d389355eba
tr PsychotechnicalTestScheduling 588457a1-2cb2-4e0f-85b5-5aadc4ffd62f -> c5723c6e-2aa4-4fcc-81f0-95643ce3f5cd
pl place_initiale (1)
pl 34794110-2785-47ee-8e28-6b188b93583e (1)
pl 0337f3ba-44cd-4918-8746-138c858ef2ac (1)
```

Figure III.32 *Le processus de sélection XPDL nettoyé.*

10. Conclusion

Dans cette section, nous avons présenté une implémentation et une expérimentation de notre approche qui permet de mettre en application nos résultats présentés dans la section précédente sur la transformation de modèle XPDL en modèle RdP, et l'implémentation de notre Application de transformation de modèle XPDL en modèle RdP. Ainsi que les différents outils utilisés.

V. Conclusion

Dans ce chapitre nous avons fait une étude comparative sur les travaux qui s'intéresse à la transformation de XPDL vers le réseau de Petri. Comme le but de la transformation est la vérification du comportement par la suite, nous avons opté à une transformation qui assure l'équivalence structurelle et comportementale. Nous avons proposé notre propre méthode de transformation en se basons sur les règles de mapping trouvées dans la littérature.

Nous avons implémenter notre méthode en utilisant le langage de transformation de modèle ATL après l'avoir choisir à l'aide d'une étude comparative entre les approches et langages de transformation de modèles existants. Finalement, nous avons fait un jeu d'essai pour mieux présenter notre outil.

Conclusion générale

Conclusion générale

L'objectif de ce travail consiste en l'automatisation de la transformation de modèle de processus métier BPMN en réseau de Petri dans le cadre d'un projet de recherche SCIO-Web Social (Service de Collaboration Inter- Organisationnelle basée web social) au sien du Centre de Développement des Technologies Avancées (CDTA).

Pour ce faire, plusieurs technologies ont été utilisées. Nous citons les études préalablement faites concernant la modélisation des processus métier, à savoir : BPMN, XPDL et les réseaux de Petri, la transformation de modèles BPMN en XPDL et XPDL en réseau de Petri ainsi que le langage de programmation Java EE et l'approche ATL.

Nous avons fait une étude comparative des différentes méthodes et techniques de transformation de modèles BPMN en modèles XPDL ainsi que leurs principes de fonctionnement afin de choisir la méthode la plus adéquate ainsi qu'un outil qui permet la mis en œuvre de cette dernière. Ensuite nous avons fait une étude comparative entre les outils qui assurent cette transformation, en choisissant Bizagi comme l'outil le plus convenable.

De cette étude et recherche minutieuses effectuées, nous avons pu proposer des règles de mapping ainsi qu'une méthode de transformation et compléter les méthodes existantes dans la littérature. Puis élaborer un outil qui permet la transformation de modèles XPDL en réseau de Petri en assurant une équivalence de la structure et du comportement.

Enfin, nous pouvons humblement affirmer qu'ils ont été intégralement atteints avec satisfaction objectifs fixés au début de l'étude.

En perspective, nous souhaitons la vérification formelle de la méthode de transformation proposé dans ce document. Ainsi que la transformation des modèles BPMN en d'autres types de réseaux de Petri qui répond au futur besoin des sociétés.

Bibliographie

Farhana Islam, M. R. (2013, 09 04). *Kermeta and ATL.* (C. B. Frameworks, Interprète) University of Ottawa, Ottawa, Departement of informatics , canada.

Rahimi, S. K. (2013, mars). A Comparative Study of Model Transformation Approaches through a Systematic Procedural Framework and Goal Question Metrics Paradigm. *A Thesis Submitted for the Degree of Doctor of Philosophy*. london, Department of Informatics , UK: Kings College London.

Both, Andreas, and Wolf Zimmermann. "On more predictable implementations of reliable workflows in service-oriented architectures." In *Web Services, 2009. ECOWS'09. Seventh IEEE European Conference on*, pp. 87-96. IEEE, 2009.

BOC, G. (2017). *Adonis welcome page*. Consulté le 05 15, 2017, sur Adonis: http://en.adonis-community.com/welcome/legal/

Farhana Islam, M. R. (2013, 09 04). *Kermeta and ATL.* (C. B. Frameworks, Interprète) University of Ottawa, Ottawa, Departement of informatics , canada.

Jung, M. K. (2004, december). Mapping from BPMN-Formed Business Processes to XPDL Business Processes. 422-427. (ICEB, Éd.)

Rahimi, S. K. (2013, mars). A Comparative Study of Model Transformation Approaches through a Systematic Procedural Framework and Goal Question Metrics Paradigm. *A Thesis Submitted for the Degree of Doctor of Philosophy*. london, Department of Informatics , UK: Kings College London.

Al Hawajreh, Nizar Ismail. "Formal Verification of Workflow Processes." Master's thesis, 2014.

Xiao, D., & Zhang, Q. (2010, May). The implementation of xpdl workflow verification service based on saas. In *Service Sciences (ICSS), 2010 International Conference on* (pp. 154-158). IEEE.

Zha, Haiping, Yun Yang, Jianmin Wang, and Lijie Wen. "Transforming XPDL to Petri nets." In *International Conference on Business Process Management*, pp. 197-207. Springer Berlin Heidelberg, 2007.

WFMC. (2012, 08 30). XML Process Definition Language. *Process Definition Interface*.

Benoît, C. (2008). Ingénierie Dirigée par les Modèles (IDM): État de l'art. Toulouse, Institut de Recherche en Informatique de Toulouse, france.

CHINOSI, M. (2012). BPMN: An introduction to the standard. *Computer Standards & Interfaces [En ligne]*, pp. 124 –134.

Costa, J. d. (2012). *BPMN 2.0*. Consulté le 05 16, 2017, sur Edu tech wiki: http://edutechwiki.unige.ch/fr/Bpmn_2.0#Ambitions_BPMN_2.0_et_diff.C3.A9rence_avec_BPMN_1.2

Hawajreh, N. I. (2014, 05 1). Formal Verification of Workflow Processes. *Master Thesis*. QATAR, College of Engineering Master of Computing, IMARATES.

Ramdane, I. &. (2016, 09). Modélisation BPMN des processus collaboratifs interorganisationnels. *thème de master*. blida, USDB, Algerie.

WFMC. (2012, 08 30). XML Process Definition Language. *Process Definition Interface*.