RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ DE 8 MAI 1945 - GUELMA -

FACULTÉ DES MATHÉMATIQUES, D'INFORMATIQUE ET DES SCIENCES DE LA MATIÈRE

Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Systeme informatique

Thème

TRANSFORMATION BPMN VERS RÉSEAUX DE PÉTRI AVEC UNE APPROCHE MDE

Encadré Par :

BERREHOUMA Nabil

Présenté par :

CHOUIAL Wafa

Juin 2022

Remerciements

D’abord, je rends grâce à Dieu de m’avoir donné la chance de faire un Master et surtout de m’aider à men sortir.

J’adresse toute ma reconnaissance au M. BERREHOUMA Nabil qui a dirigé ce travail. Je le remercie pour sa patience et sa compréhension durant ces années de Master.

Je le remercie infiniment pour son apport scientifique incontestable, mais aussi pour toutes ses qualités humaines, sa compréhension, sa tolérance, sa patience et son aide.

Je tiens à remercier les membres du jury d’avoir fait l’honneur d’accepter de participer au jury et de juger ce travail.

J’exprime mon remerciement a tous les enseignants de département informatique qui m’ont suivi durant mon parcours académique et qui ont su me transmettre leurs savoirs faire.

dédicaces

Je dédie ce travail,

A ma Chère maman .

A mon mari qui m'encourage de terminer mes études.

A mes enfants Rimes, Adem.

A mes frères.

A toute ma famille.

A mes amis Zehwa ;loubna ;Asma ; Et tous les étudiants d'informatique.

A tous ceux qui m'aiment.

RÉSUMÉ

L'informatisation des processus métiers d'une entreprise est une étape importante de l'industrialisation. Cette industrialisation permet aux entreprises de gagner du temps et de se déplacer vers des domaines à plus forte valeur ajoutée. La modélisation est une étape essentielle de l'informatisation des processus, car elle permet d'étudier et de rationaliser les processus métiers. Business Process Modeling Notation (BPMN) est une norme de capture des processus métier dans les premières étapes du développement du système. Le mélange structurel dans BPMN permet de créer des modèles avec des erreurs sémantiques, qui sont particulièrement graves car les erreurs commises dans les premières étapes du développement du système sont les plus coûteuses et les plus difficiles à corriger.

Par conséquent, la capacité de vérifier statiquement l'exactitude sémantique d'un modèle est une caractéristique souhaitable d'un outil de modélisation basé sur BPMN, cet mémoire propose une mise en correspondance de BPMN à un langage formel, à savoir les réseaux de Petri, qui peut utiliser des techniques analytiques efficaces. Le mappage proposé a été mise en œuvre comme un outil qui combiné aux outils existants basés sur le réseau de Petri, permet une analyse statique du modèle BPMN. La formalisation aide également à identifier les lacunes dans la spécification BPMN standard.

Le but de ce travail est de définir une approche de transformation automatique

d'un modèle BPMN vers un réseau de Petri équivalent sur lequel nous pouvons faire une analyse qualitative en utilisant des outils existants. Ce travail tente alors de proposer une solutions d'instanciation de processus et montrer l'importance de placer le client au centre de ses préoccupations. L'utilisation d'une méthodologique générique et réutilisable est possible pour guider les diverses étapes de la transformation de processus métier dans le but d'aboutir un outil graphique pour la transformation des modèles BPMN vers les Réseaux de Petri (RDP) cette méthodologie sera basée sur la transformation de graphes en formats de spécification ATL à l'aide des outils EMF (Eclipse Modeling Framework) .

Mots-clés : Processus métiers,BPMN ,Réseaux de Petri , ATL,EMF

ABSTRACT

The computerisation of a company's business processes is an important step in industrialisation. This industrialisation allows companies to save time and move to higher value-added areas. Modelling is an essential step in the computerisation of processes, as it enables the study and rationalisation of business processes. Business Process Modeling Notation (BPMN) is a standard for capturing business processes in the early stages of system development. The structural blending in BPMN allows for the creation of models with semantic errors, which are particularly serious as errors made in the early stages of system development are the most costly and difficult to correct.

Therefore, the ability to statically check the semantic correctness of a model is a desirable feature of a BPMN-based modelling tool, this dissertation proposes a mapping of BPMN to a formal language, namely Petri nets, which can use efficient analytical techniques. The proposed mapping has been implemented as a tool that combined with existing Petri net based tools, allows a static analysis of the BPMN model. The formalisation also helps to identify gaps in the standard BPMN specification. The aim of this work is to define an approach to automatically transform a BPMN model into an equivalent Petri net on which we can perform a qualitative analysis using existing tools. The use of a generic and reusable methodology is possible to guide the various stages of the transformation of business processes in order to achieve a graphical tool

for the transformation of BPMN models to Petri nets (RDP). This methodology will be based on the transformation of graphs into ATL specification formats using EMF tools (Eclipse Modeling Framework).

Keywords : Business Processes, BPMN, Petri nets, ATL, EMF.

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION GÉNÉRALE

Beaucoup de changements touchent les entreprises actuelles que se soit sur la forme organisationnelle ou sur les méthodes de conception et de fabrication. Ces changements sont la conséquence des nouvelles normes concurrentielles crées par le marché en évolution constante, chose qui engendre une plus forte concurrence.

Avec la mondialisation du marché, des cycles de vie des produits plus courts, un besoin croissant de flexibilité et des changements fréquents de technologie, les entreprises doivent prendre la modélisation au sérieux pour faire face à toutes ces complexités. La modélisation d'entreprise est une tâche complexe qui implique la représentation et la spécification de divers aspects des opérations commerciales. Informations, ressources et fonctions organisationnelles. Ces dernières années, le terme "processus" est devenu largement utilisé. Il est reconnu que pour se conformer à la nouvelle philosophie d'entreprise, l'organisation doit être conçue pour fournir le flux d'informations vertical et horizontal nécessaire pour atteindre l'objectif global. De l'organisation. Une approche processus consiste à décrire systématiquement une organisation ou un ensemble d'activités processus pour organiser les contributions à la satisfaction client. Il existe plusieurs types de processus (contrôle, métier et support). Le but de la modélisation de processus est de créer une abstraction du processus et de servir de base à une définition détaillée, à la recherche et aux possibilités de reconception pour éliminer les activités sans valeur ajoutée. La modélisation des processus doit

être capable de comprendre clairement et de manière transparente les activités considérées, les dépendances entre les activités et les rôles requis pour le processus (personnes, machines, informations, etc.). La capacité à représenter des comportements tels que la concurrence et la sélection augmente la probabilité de définir un modèle logiquement incorrect avec des erreurs de contrôle de flux (impasse, vivacité, etc.).

Le but de ce travail est de définir une approche de transformation automatique d'un modèle BPMN vers un réseau de Pétri équivalent sur lequel nous pouvons faire une analyse qualitative en utilisant des outils existants.

Ce travail contient trois parties suivantes :

1. l'approche de l'ingénierie des modèles (MDE : Model Driven Engineering)

2. Métamodélisation des modèles BPMN (source) et les réseaux de Petri (Cible)

3. Définition des Règles de la transformation (ATL) BPMN vers RdP et l'Exécution de la transformation sur un modèle BPMN source à l'aide des outils EMF (Eclipse Modeling Framework).

Ce mémoire est structuré de la manière suivante :

- Le premier chapitre donne un aperçu sur l'ingénierie digirgée par les modèles et son évolution. Il se concentre sur le concept des modèles, l'évolution des métamodèles dans une organisation et les types de transformation EMF (Eclipse Modeling Framework) et ATL (Transformation Language).

- Le deuxième chapitre présente une définition de processus métier en entreprise. ses différents modèles. Ce chapitre débute par rappeler la modélisation des processus, les objectifs de modélisation et les critères de modélisation, cette partie montre un moyen de modéliser les processus métier en s'appuyant sur le standard de notation BPMN

- Le troisième chapitre décrit les différents concepts des réseaux de Petri, nous avons présenté les réseaux de Petri de haut niveau que nous avons vus. Les réseaux de Petri aident à représenter le système en rassemblant l'état avec l'activité Au passage vers un lieu ou un événement.

- Le quatriéme chapitre propose la première contribution consistant en une approche entièrement automatisée basée sur des transformations de graphes qui modifient les modèles de processus métier et les réseaux de Petri en formats de spécification ATL à l'aide des outils EMF (Eclipse Modeling Framework).).

Cette approche se déroule en deux étapes : La première étape propose deux métamodèles, un pour le modèle de processus métier et un pour le graphe réseau de Petri. L'environnement de développement ATL fournit également de nombreuses fonctionnalités supplémentaires pour la gestion des modèles et des métamodèles. Ces fonctionnalités incluent non seulement une notation de texte simple spécifiquement pour les spécifications de métamodèle, mais également un ensemble de ponts standard entre la syntaxe de texte commune et sa représentation de modèle correspondante

Une conclusion viendra présenter un bilan récapitulatif de notre travail, des conclusions générales et les perspectives de recherche concluront cette étude.

CHAPITRE 1

L'INGÉNIERIE DIRIGÉE PAR LES MODÈLES

1.1 Introduction

La modélisation, au sens large, est en effet l'utilisation efficace d'une représentation simplifiée d'un aspect de la réalité pour un objectif donné. Loin de se réduire à l'expression d'une solution à un niveau d'abstraction plus élevé que le code, la modélisation en informatique peut être vue comme la séparation des différents besoins fonctionnels et préoccupations extra-fonctionnelles (telles que sécurité, fiabilité, efficacité, performance, ponctualité, flexibilité, etc.) issus des exigences. Ce que propose l'approche de l'ingénierie des modèles (IDM, ou MDE en anglais pour Model Driven Engineering) est simplement de mécaniser le processus que les ingénieurs expérimentés suivent à la main. L'intérêt pour l'IDM a été fortement amplifié à la fin du 20ième siècle lorsque l'organisme de standardisation OMG (Object Modeling Group) a rendu publique son initiative MDA (Model Driven Architecture), qui peut être vue comme une restriction de l'IDM à la gestion de l'aspect particulier de dépendance d'un logiciel à une plateforme d'exécution.

Dans ce chapitre, on va présenter le concept de transformation de modèles en commençant par une représentation de concepts de transformation de modèle dans le

cadre général, ensuite, on présente une énumération des différents types de transformations, suivie par une classification des différents approches, et enfin on présente, aussi, un cadre spécifique des transformations de modèles basé sur les transformations de graphe.

1.2 Architecture piloté par modèle MDA

1.2.1 Introduction

L'architecture piloté ou dirigée par modèle est une approche pour la modélisation et pour la génération d'applications

La MDA -proposée par l'OMG (Object Management Group)- est une méthode de spécification d'un système informatique qui sépare la spécification des fonctions du système de l'implémentation de ces fonctions sur une plate-forme spécifique. Cette approche se concentre sur les modèles, fournit un niveau d'abstraction plus élevé pendant le développement et permet de séparer les modèles indépendants de la plate-forme (modèles PIM) des modèles cibles spécifiques à la plate-forme (modèles PSM). Les concepts de métamodèles et de transformations de modèles sont particulièrement importants dans les architectures pilotées par des modèles. Les métamodèles sont définis chez OMG à l'aide du standard MOF (Meta Object Facility).Un langage standard spécifique à la transformation de modèles appelé QVT a été défini par l'OMG, qui a également défini un mécanisme d'échange de modèles, basé sur XML et appelé XMI [15].

1.2.2 Les fondamentaux de MDA

L'initiative Model Driven Architecture (MDA) d'OMG est motivée par la nécessité de revoir la manière de conception des applications en s'appyant esssentiellement sur l'usage des modèles, car les modèles sont plus durables que le code.

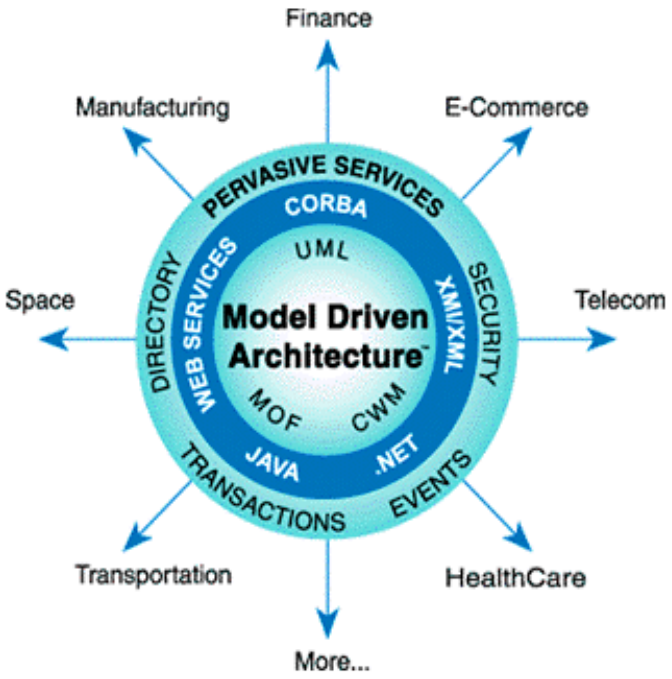


FIGURE 1.1 – L'architecture pilotée par le modèle(MDA) [15]

Le principe de MDA est de séparer les spécifications fonctionnelles des spécifications d'implémentation sur une plate-forme particulière. L'idée centrale de MDA est le développement de modèles, d'abord à partir de l'analyse, puis de la conception au code, la transformation, la dérivation et la continuité. En utilisant le langage déclaratif (basé sur des règles) "QVT" qui peut être transformé avec "query/view/-transform", OMG fournit un standard pour spécifier les transformations de modèle. Il fournit une architecture et un langage dédiés qui facilitent la génération de modèles à partir d'autres modèles. Défini pour la première fois en 2002 et adopté par l'Object Management Group (OMG) en 2005, QVT fait partie de la norme MetaObject Facility (MOF) [23]

1.2.3 Les Niveaux de Modèles MDA

L'approche MDA distingue deux aspects clés du processus de développement d'applications : l'aspect commercial, qui représente la fonctionnalité de l'application, et l'aspect technique, qui prévoie la technologie de enjeu en œuvre de l'utilisation .

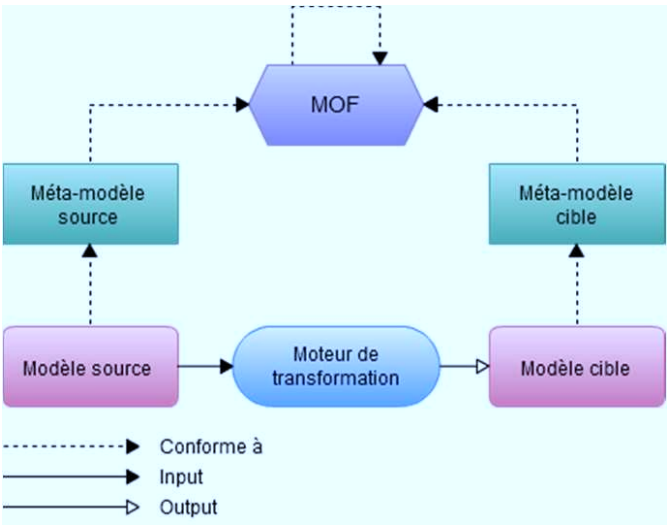


FIGURE 1.2 – Aperçu global de l'approche MDA [12]

Chaque aspect est représenté par un ensemble de modèles qui transmettent les informations nécessaires pour générer le code source de votre application. Passez d'une vision méditative du modèle à une vision productive. MDA définit trois niveaux de modèle qui représentent le niveau d'abstraction de votre application : CIM, PIM, PDM et PSM. [24]

CIM (computation independant model) modèle indépendant de calcul :

Les modèles d'exigence CIM décrivent les besoins fonctionnels de l'application, aussi bien les services qu'elle offre que les entités avec lesquelles elle interagit. Leur rôle est de décrire l'application indépendamment des détails liés à son implémentation. Les CIM peuvent servir de référence pour s'assurer que l'application finie correspond aux demandes des clients . -Est indépendant de tout système informatique -Décrit les concepts de l'activité métier, le savoir faire les -Processus, la terminologie et les règles de gestion (de haut niveau) -Décrit la situation dans lequel le système est utilisé -N'est modifié uniquement que si les connaissances ou les besoins Métier changent (très longue durée de vie) Les exigences modélisées dans le CIM seront prise en compte dans les constructions des PIM (Platform Independent Model) et des PSM (Platform Specific Model) [24]

PIM (plateform independant model) modèle indépendant des plates-formes :

Les modèles PIM sont les modèles d'analyse et de conception de l'application. La phase de conception à cette étape du processus suppose l'application de Design pattern, le découpage de l'application en modules et sous-modules, etc. Le rôle des PIM est de donner une vision structurelle et dynamique de l'application, toujours indépendamment de la conception technique de l'application. Un modèle PIM est un modèle de conception qui : -Décrit le système indépendamment de toute plate-forme technique et de toute technologie utilisée pour déployer l'application -Représente le logique métier spécifique au système (fonctionnement des entités et des services) - Est pérenne dans le temps - Consiste-en des diagrammes UML de classes (avec des contraintes en OCL) Les différents niveaux de PIM précisent les choix de persistance, de gestion des transactions, de sécurité [24].

PDM (plateform dependant model) modèle des plates-formes :

Décrit une architecture technique (plusieurs par projet) -Contient des informations pour la transformation des -Modèles vers une plateforme -Est spécifique à une plateforme -Est un modèle de transformation pour permettre le passage du PIM vers le PSM[24].

PSM (plateform specific model) modèle dépendant des plates-formes :

Le PSM est le modèle qui se rapproche le plus du code final de l'application. Un PSM est un modèle de code qui décrit l'implémentation d'une application sur une plateforme particulière, il est donc lié à une plateforme d'exécution. Le premier niveau, issu de la transformation d'un PIM par l'adaptation des modèles UML aux spécificités la plate-forme Les autres niveaux PSM sont obtenus par transformations successives en prenant en compte le langage (Java, C, PHP...), les choix de conception, Le dernier niveau, ou PSM d'implantation, décrit, en autres, le code du programme, les schémas

des tables, les bibliothèques utilisées, les descripteurs de déploiement . -Sert à la génération du code exécutable pour les plates-formes techniques particulières -Décrit comment le système utilisera la plate-forme. -Est dépendant de la plate-forme.[24]

Code source

Représente le résultat final du processus MDA, le code source est obtenu par génération automatique (partielle ou totale) du code de l'application à partir du PSM. Le code source obtenu peut toujours être enrichi ou modifié manuellement[12].

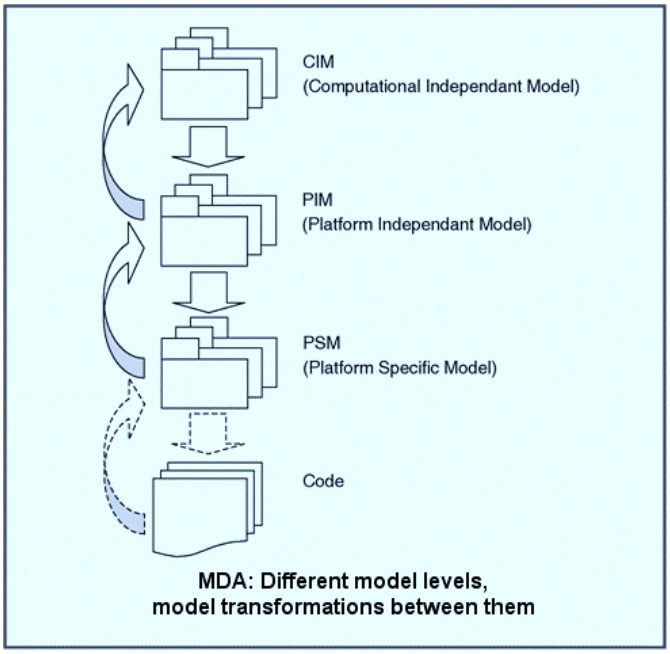


FIGURE 1.3 – Aperçu global de l'approche MDA [2]

1.2.4 Les transformations de modèles

Les modèles CIM, PIM et PSM constituent une étape majeure dans l'approche MDA. Chacun de ces modèles contient les informations nécessaires pour générer le code source de l'application. Le code est obtenu par génération automatique à partir de PSM, qui est obtenu en convertissant en continu le modèle CIM en PIM et le modèle PIM en PSM. La transformation du modèle est une étape importante dans le

processus MDE. Grâce à la transformation, le modèle devient un élément productif de MDA. La réalisation de la conversion garantit des liens de traçabilité entre les différents modèles du processus MDE. Ces liens garantissent la qualité du processus de développement logiciel de MDA [3] .

Transformations CIM vers PIM

Le modèle CIM représente les besoins de l'utilisateur. Cette étape consiste à créer un modèle PIM partiel à partir de CIM. Le but est d'afficher les informations contenues dans le CIM vers le modèle PIM. Cela garantit que vos besoins sont communiqués et respectés tout au long du processus de MDA [3].

Transformations PIM vers PDM

Le modèle PIM modélise les aspects structurels et dynamiques de votre application. Cette étape se traduit par l'amélioration du modèle PIM. Les améliorations du PIM consistent à ajouter des informations utiles au PIM et à préciser son contenu. Par conséquent, PDM est affiné et les informations contenues dans PDM sont plus précises.[3].

Transformations PDM vers PSM

Dans cette étape, vous allez créer un modèle PSM à partir des informations fournies par le modèle PIM et ajouter des informations techniques liées à la plateforme d'exécution cible. Cela créera une connexion à la plateforme d'exécution. Le PSM fournit des informations utiles pour générer le code d'application et dépend de la plate-forme. Vous pouvez créer autant de PSM que votre plateforme cible [2].

Transformations PSM vers code

Convertir un modèle PSM en code source consiste à générer tout ou partie du code source de l'application à partir du modèle PSM de l'application. À proprement parler,

cette étape n'est pas considérée comme une conversion MDA. En effet, selon MDA, les transformations sont définies par des transformations d'un modèle à un autre, et chaque modèle est structuré par son métamodèle. Cependant, étant donné que le code source n'a pas de métamodèle, la conversion PSM en code est considérée comme une transcription textuelle du modèle PSM [2].

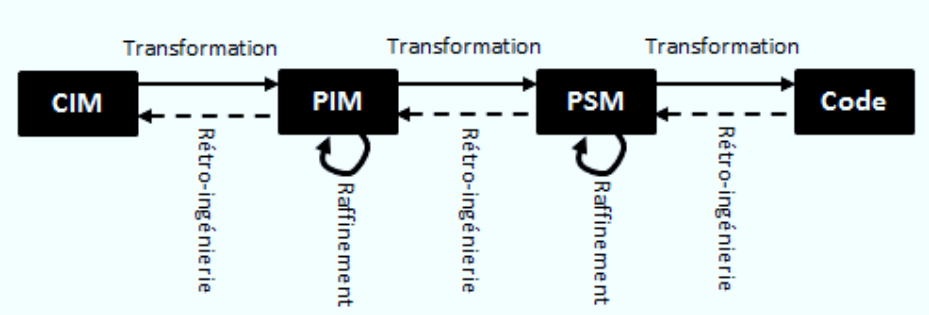


FIGURE 1.4 – Les transformations de modèles MDA [2]

1.2.5 L'architecture de MDA

Lorsque vous écrivez des programmes en Java (ou tout autre langage), faites attention à la grammaire du langage Java. C'est une forme de modélisation. Cette forme est définie par la grammaire BNF (forme BackusNaur) qui se définit elle-même. La grammaire BNF dans ce cas est appelée métaformalisme. Il en va de même pour le développement de modèles utilisant l'approche MDA [6].

Les niveaux méta

Dans l'approche MDA, l'application informatique est l'entité modélisée. Le modèle représente une partie ou la totalité des informations nécessaires pour créer ces applications. Par conséquent, dans MDA, les applications informatiques sont représentées par un ou plusieurs modèles. Un métamodèle définit la structure (et non la sémantique) d'un modèle conforme à ce métamodèle. Un métamodèle est un diagramme de classes qui définit les entités du modèle, les propriétés de leurs connexions et leurs règles d'intégrité. Pour être valide, chaque modèle doit être conforme à son

métamodèle.Cette relation de conformité est primordiale dans l'approche MDA, il est possible ainsi de construire des outils capables de manipuler les modèles. Le méta-métamodèle est pour ainsi dire le métamodèle des métamodèles. Et entretient donc avec eux la même relation qu'a un métamodèle avec ses modèles. Les métamodèles sont des modèles conformes à leur métamétamodèle.Dans MDA, quel que soit le niveau méta, tous les éléments sont considérés comme des modèles. Par conséquent, les métamodèles et les métamétamodèles sont aussi des modèles.

On aurait pu continuer à monter dans les niveaux méta, mais le fait est que le métamétamodèle, utilisé par MDA, se définit lui-même. Autrement dit, il est son propre métamodèle ce qui amène à 04 niveaux de hiérarchie. Il n'existe en effet qu'un seul métamétamodèle utilisé par MDA et qui est le standard MOF (Meta Object Facility) qui a pour caractéristique de se définir lui-même [6].

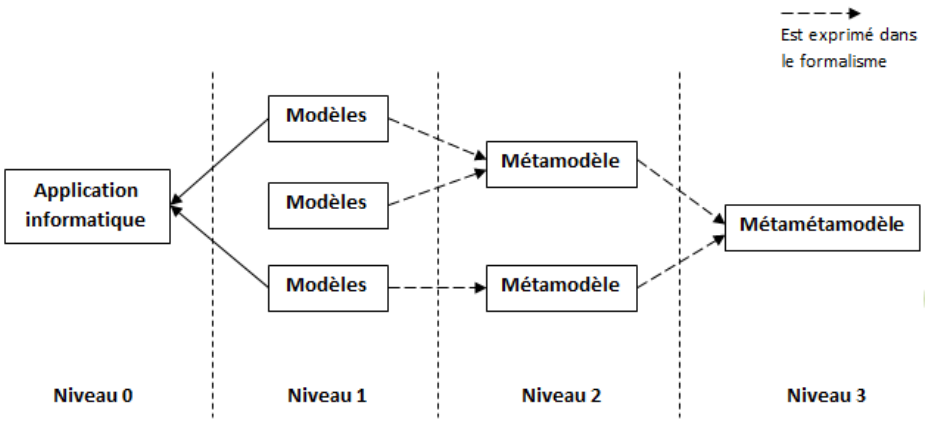


FIGURE 1.5 – L'architecture de MDA [2]

L'architecture de MOF1.4

L'OMG a défini le standard MOF (Meta Object Facility). La version 1.4 de MOF est celle utilisée par les modèles publics de l'OMG et donc, elle est largement répandue. Le rôle du métamétamodèle MOF1.4 est de structurer les métamodèles de l'architecture. Il représente les métamodèles sous la forme de diagrammes de classes. Le

diagramme de classes qui modélise le métamodèle est composé de métaclasses, métaattributs, etc. Cette nomination sert à différencier avec les diagrammes de classes des modèles. Nous avons dit que le métamétamodèle MOF1.4 est le métamodèle des métamodèles, par conséquent, il est lui-même représenté sous la forme de diagramme de classes [12].

Le métamodèle UML et les profils UML

MDA recommande l'utilisation du métamodèle UML pour l'élaboration des modèles CIM et PIM, et l'utilisation des profils UML pour l'élaboration des modèles PSM. Un métamodèle UML est un diagramme de classes conforme au standard MOF1.4. Les modèles CIM, PIM et PSM sont des modèles UML conformes au métamodèle UML.

En plus d'être un standard de modélisation très largement répandu, UML définit un ensemble de diagrammes pour la modélisation des différents aspects (fonctionnels, structurels, dynamiques...) des applications orientées objet, indépendamment des plateformes d'exécution ce qui le rend idéal pour l'élaboration des modèles CIM et PIM.

UML définit aussi la notion de profil UML. Un profil UML est une extension - ou adaptation - d'UML à un domaine particulier. Dans le contexte MDA, les profils UML permettent de décrire la structure d'une plateforme d'exécution particulière grâce à des diagrammes UML stéréotypés. Un profil UML correspond à une plateforme unique, il est dépendant des plateformes d'exécution et donc idéal pour la modélisation des PSM. Une autre approche possible pour l'élaboration des PSM, ce sont les métamodèles de plateformes. Un PM est un métamodèle conforme au standard MOF1.4 et décrit une plateforme particulière. Profil UML ou métamodèle de plateformes, les deux approches sont recommandées par MDA [12].

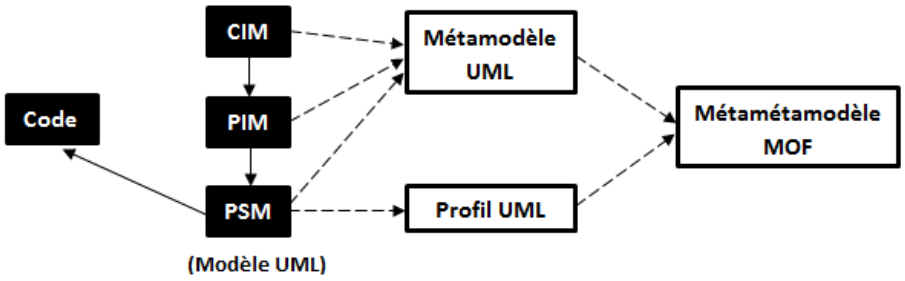


FIGURE 1.6 – Le métamodèle UML et les profils UML [12]

1.2.6 Les Transformations

L'approche MDA préconise de modéliser la transformation elle-même. MDA considère la conversion comme une application. Un modèle de transformation est une fonction qui prend un ensemble de modèles en entrée et renvoie un ensemble de modèles en sortie. Les modèles d'entrée et de sortie sont structurés par des métamodèles. Il existe trois manières différentes de modéliser une fonction de transformation. Le principe reste le même, mais la différence réside dans la formulation des règles de conversion.

— *Approche par programmation* : Dans cette approche, l'idée est de programmer une transformation de modèle de la même façon que n'importe quelle application informatique, en utilisant les langages orientés objet. Et où les données à manipuler représentent les modèles impliqués dans la transformation.

— *Approche par template* : L'approche par template consiste à définir des modèles templates et à remplacer leurs paramètres par les valeurs des modèles sources. Les modèles templates, ou modèles cibles paramétrés, sont des canevas des modèles cibles, ils contiennent des paramètres qui seront par la suite, lors de la transformation, substitués par les valeurs des modèles sources. Cette approche utilise un langage particulier pour la définition des modèles templates.

— *Approche par modélisation* : Par analogie à l'approche MDA, l'approche par modélisation modélise aussi les règles de transformations en se basant sur l'ingénierie dirigée par les modèles. Dans cette approche, un modèle de transformations est structuré par son métamodèle, et les modèles sont indépendants des plateformes d'exécution. L'objectif de cette approche est de pérenniser les modèles de transformations et de les rendre productifs. L'OMG a défini le standard MOF2.0 QVT (Query/View/Transformation) comme métamodèle pour structurer les modèles de transformations de modèles. Les modèles de transformations, conformes au métamodèle MOF2.0 QVT, expriment les règles de correspondance entre le métamodèle source et le métamodèle cible nécessaires à la transformation[6].

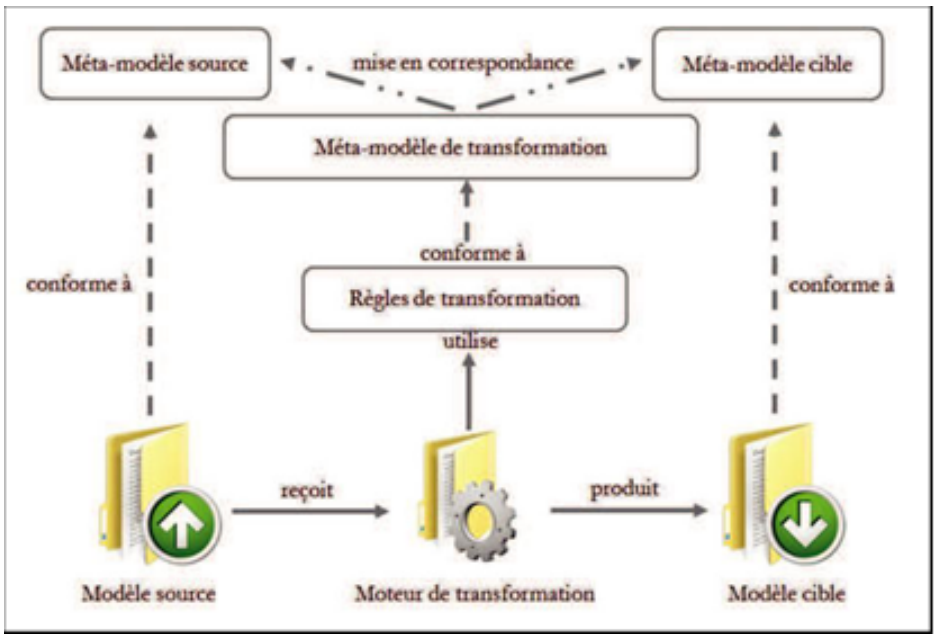


FIGURE 1.7 – Les approches de transformation MDA [6]

1.2.7 Objectifs de l'approche MDA

L'OMG à défini l'approche MDA pour résoudre les problèmes liés à l'évolution continue de la technologie. MDA est le fondateur de Model Driven Engineering. Tout

ce qui constitue l'approche, la modélisation, la transformation et la technologie de mise en œuvre de MDA est conçu pour aider MDA à atteindre les objectifs suivants :

— La pérennité des savoir-faire : L'approche MDA vise principalement à rendre les spécifications métier des entreprises pérennes, indépendamment des technologies de mise en œuvre.

— Les gains de productivité :Dans MDA les modèles deviennent un outil de production grâce à l'automatisation des transformations de modèles.

— La prise en compte des plateformes d'exécution :Le développement d'applications multiplateformes, ou encore la migration logicielle, sont facilités puisqu'il suffit à partir du PIM, d'effectuer les transformations PIM vers PSM en y intégrant à chaque fois, le modèle de la plateforme concernée.
[2][6].

1.3 Outils et Types Transformation

La transformation de modèles est au cœur d'une approche d'ingénierie dirigée par les modèles. Cependant, il n'y a toujours pas de consensus sur la manière de définir et de mettre en œuvre la transformation. De nombreuses outils ont été proposés dans la littérature. Pour pouvoir effectuer des transformations de modèles, ils doivent être représentés dans un langage de modélisation ou un métamodèle particulier. Sur la base des métamodèles source et cible de la transformation, nous distinguons deux types de transformations : intrinsèque et extrinsèque. Si les modèles impliqués sont issus du même métamodèle, la transformation est dite endogène. Cependant, si les modèles source et cible sont issus de métamodèles différents, la transformation est dite extrinsèque voire transformation.

Les transformations exogènes peuvent être : (i) Synthèse - transformation d'un certain niveau d'abstraction vers un niveau d'abstraction moins élevé. Un exemple typique est la génération de code, (ii) Rétro-ingénierie - inverse de la synthèse, (iii)

Migration - transformation d'un programme écrit dans un langage vers un autre langage du même niveau d'abstraction.

Les transformations exogènes synthèse et rétro-ingénierie sont aussi appelées transformations verticales, et les transformations exogènes migration sont aussi appelées transformations horizontales.

1.4 EMF (Eclipse modeling framework)

Le projet EMF est un cadre de modélisation et de génération de code pour la construction d'outils et d'autres applications basées sur un modèle de données structuré. À partir d'une spécification de modèle décrite dans XMI, EMF fournit des outils et un support d'exécution pour produire un ensemble de classes Java pour le modèle, ainsi qu'un ensemble de classes d'adaptateur qui permettent l'affichage et l'édition basée sur des commandes du modèle, et un éditeur de base .

EMF (core) est une norme commune pour les modèles de données, sur laquelle de nombreuses technologies et cadres sont basés. Cela inclut les solutions de serveur, les frameworks de persistance, les frameworks d'interface utilisateur et la prise en charge des transformations.

Eclipse Modeling Framework (EMF) est un environnement qui permet la modélisation, la métamodélisation ainsi que la génération de code au sein de la plateforme Eclipse. Avec EMF, il est possible de définir les modèles de différentes manieres. Traditionnellement, les modèles peuvent être construits en utilisant le Java annoté, XML Schéma Définition (XSD) ou bien la notation UML de Rational Rose. Indépendamment des formalismes utilisés pour le définir, un modèle EMF est la représentation commune qui les regroupe tous ensemble. En effet, EMF propose également sa propre notation, appelée Ecore comme langage canonique pour décrire ses modèles. Ainsi, quelle que soit l'annotation retenue pour définir un modèle EMF, nous obtiendrons à la fin un modèle Ecore génère. EMF dispose d'un éditeur d'arborescence proche de XML et d'un éditeur graphique proche de UML [9][25].

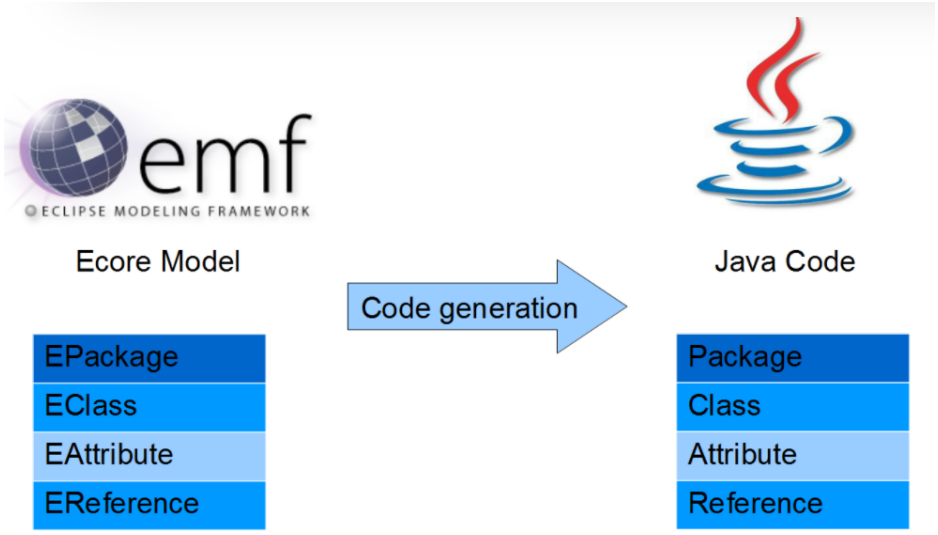


FIGURE 1.8 – Le projet EMF [25]

1.5 ATL (transformation langage)

ATLAS Transformation Language (ATL) est un langage de transformation de modèles plus ou moins inspiré par le standard QVT de l'Object Management Group. Il est disponible en tant que plugin dans le projet Eclipse.

ATL est un langage de transformation hybride. Il contient un mélange de constructions déclaratives et impératives. Le style encouragé est déclaratif.

Les transformations ATL sont unidirectionnelles, fonctionnant sur des modèles source en lecture seule et produisant une cible en écriture seule des modèles. Une transformation bidirectionnelle est implémentée comme un couple de transformations : une pour chaque direction. Pendant l'exécution d'une transformation, le modèle source peut être parcourue mais les modifications ne sont pas autorisées.

Le modèle cible ne peut pas être parcouru. Les modèles source et cible pour ATL peuvent être exprimés dans le XMI OMG format de sérialisation. Les métamodèles source et cible peuvent également être exprimés en XMI ou dans le KM3 plus pratique notation.

Une transformation ATL peut être décomposée en trois parties : un en-tête, des assistants et des règles.L'en-tête sert à déclarer des informations générales comme le

nom du module (c'est le nom de la transformation : il doit correspondre au fichier nom), les métamodèles source et cible et les bibliothèques importées.

Les helpers sont des sous-programmes (basés sur OCL) qui sont utilisés pour éviter la redondance de code. Les règles sont au cœur des transformations ATL car elles décrivent comment la cible les éléments (basés sur le métamodèle cible) sont produits à partir d'éléments sources (basés sur le métamodèle source)

ATL est accompagné d'un ensemble d'outils construits sur la plate-forme Legend of the Eclipse. ADT (ATL DevelopmentToolkit) est composé du moteur de transformation ATL (bloc Engine) et de l'environnement de développement intégré ATL (IDE : blocs Editor, Builder et Debug).[25].

1.5.1 Moteur ATL

Le moteur ATL (voir à droite) est responsable de la gestion des tâches principales d'ATL : compilation et exécution. Les transformations ATL sont compilées en programmes dans un byte-code spécifique. Le byte-code est exécuté par la machine virtuelle ATL (VM). La VM est spécialisée dans la manipulation de modèles et fournit un ensemble d'instructions pour la manipulation de modèles.

La machine virtuelle peut s'exécuter sur divers systèmes de gestion de modèles. Pour isoler les VM de leurs spécificités, un niveau intermédiaire est introduit appelé Model Handler Abstraction Layer. Cette couche traduit les instructions de la machine virtuelle pour la manipulation du modèle en instructions d'un gestionnaire de modèle spécifique. Les gestionnaires de modèles sont des composants qui fournissent une interface de programmation pour la manipulation de modèles [27].

1.6 OCL : Object Constraint Language

En modélisation , un modèle ne suffit pas pour une spécification précise et sans ambiguïté. Il est nécessaire de décrire des contraintes supplémentaires sur les objets dans le modèle. Ces contraintes sont souvent décrites en langage naturel. La pratique

a montré que cela entraînera toujours des ambiguïtés. Pour écrire des contraintes non ambiguës, des langages dits formels ont été développés [28]. OCL est un langage formel qui formalise l'expression des contraintes. C'est un langage adopté pour les diagrammes UML et en particulier pour les diagrammes de classe.

FIGURE 1.9 – Exemple sur OCL [27]

OCL est un pur langage d'expression. Par conséquent, une expression OCL est garantie sans effet secondaire. Cela signifie que l'état du système ne changera jamais à cause d'une évaluation d'une expression OCL. De plus, OCL est un langage de modélisation (pas un langage de programmation), il n'est pas possible d'écrire une logique de programme ou un contrôle de flux dans OCL.

1.7 Conclusion

Ce chapitre a introduit les principes généraux du MDE qui sont la métamodélisation d'une part et transformation de modèles d'autre part. Ces deux axes représentent les deux principaux thèmes de MDE sur lesquels la plupart des recherches se concentrent actuellement. Les premiers résultats de la métamodélisation ont établi les concepts de base de l'architecture pilotée par les modèles (modèles et métamodèles). Dans le prochain chapitre, Nous aborderons -entre autres- la modélisation des processus métiers où un langage de modélisation (BPMN) sera présenté. Nous présenterons plus tard la démarche MDE pour l'élaboration d'un méta-modèle pour BPMN.

CHAPITRE 2

LES PROCESSUS MÉTIERS ET LE LANGAGE BPMN

2.1 Introduction

Les processus structurent l'ensemble des activités quotidiennes nécessaires à l'exercice de chaque métier. Ils consistent en une série de tâches et d'actions, parfois complexes, qu'un ensemble de collaborateurs se doivent accomplir afin d'obtenir un résultat déterminé (par exemple : satisfaction d'une demande client, lancement d'un produit, suivi des demandes d'achats, gestion des processus qualité, intégration d'un nouvel employé, gestion des intérimaires, publications d'appels d'offres...) [16]. Dans ce chapitre, Nous focalisons sur le concept des processus métiers au sein de l'entreprise, leur cycle de vie et leur management. le langage BPMN est un outil de modélisation des processus métiers largement utilisé. Une partie importante de ce chapitre est consacrée à ce langage.

2.2 Business Process Management

Un système de gestion des processus métier en anglais Business Process Management (BPM) permet de représenter les activités d'une organisation : ses processus. Il

met en évidence les interactions humaines et les échanges de données avec les systèmes d'informations existants. En définissant les étapes, les actions et les échanges. Le but du BPM est de rendre un processus explicite, c'est-à-dire de le modéliser, sous la forme par exemple d'un logigramme. Dans le domaine du BPM, il existe des normes comprenant des symboles spécifiques utilisés pour modéliser les processus opérationnels et qui permettent de faire la distinction entre les étapes, les tâches ou les activités effectuées par des personnes et celles qui sont automatisées (effectuées par un logiciel, par le matériel informatique, ou par une combinaison des deux) Il apporte une vision plus large et fait ressortir les problèmes de performance. L'idée étant de s'assurer que ses processus sont mis en œuvre pour répondre aux besoins clients avec le meilleur niveau de performance.

Le BPM permet de comprendre le fonctionnement des différents domaines de l'entreprise : Qualité, Achat, SI, Commercial, RH, Ventes, HelpDesk etc... et permet d'analyser et de fluidifier les processus mis en place dans une optique d'amélioration de la performance. Le BPM comprend tout où partie de ce qui suit :

- Un outil permettant de représenter graphiquement des processus ;
- Un moyen pour simuler et optimiser les processus avant leur déploiement ;
- Un système permettant d'exécuter des processus comprenant à la fois des activités humaines et automatisées ;
- Une interface permettant aux personnes d'effectuer les tâches leur incombant dans un processus ;
- un moyen d'accéder et d'interagir avec les systèmes existants dans l'entreprise (bases de données, systèmes de gestion de données, messagerie etc.) ;
- Des outils pour suivre et gérer les processus au fur et à mesure qu'ils s'opèrent ;
- Un moyen de collecter et de traiter les données issues des processus en temps réel.

Alors Le BPM est un moyen de définir et de gérer les étapes d'un processus métier, du début jusqu' à la fin. Ces processus consistent en une séquence d'activités présentant un intérêt pour une organisation [20].

FIGURE 2.1 – Pourquoi avez-vous besoin d'un BPM [20]

2.2.1 Cycle de vie d'un BPM

Nous resumons le cycle de vie d'un BPM dans les points suivants :

— Diagnostic de l'organisation : consiste essentiellement à l'étude de l'existant, identification et choix des processus métiers prioritaires. L'objectif : choisir les processus cibles à inclure dans la démarche [22].

— Analyse du processus : Au cours de cette étape, on examinee les processus métiers actuels et les schématiser de bout en bout. À ce stade, on n'y apporte aucune modification, on se contente seulement de comprendre en quoi ils consistent.

— Conception (Design) : consiste à établir un modèle du processus idéal afin de pouvoir commencer la mise en œuvre à l'étape suivante.

— Exécution : Au cours de l'étape d'exécution, parfois appelée l'étape de réalisation, mettez votre modèle en application. Au fur et à mesure, définissez des indicateurs de réussite ou d'échec, afin d'évaluer si ce processus est effectivement mieux que le précédent.

— Surveillance (Monitor) : Une fois que vous avez mis en œuvre les nouveaux

processus, surveillez-les pour voir s'ils fonctionnent bien. Ces nouveaux processus ont-ils réellement amélioré les points de blocage et les étapes inefficientes ? Sont-ils utilisés ? Parfois, même si l'idée semblait bonne sur le papier et a même montré des résultats concluants lors d'un petit test, il est possible qu'elle ne fonctionne pas lors de son déploiement à l'échelle de l'organisation. Si c'est le cas, cessez le déploiement ou envisagez une autre solution. En surveillant ces processus, vous pouvez identifier tout problème de manière proactive et y remédier si nécessaire.

— Optimisation (Optimize) : Au cours de l'étape d'optimisation, parfois appelée l'étape d'automatisation, continuez à ajuster et à améliorer votre processus métier. Même s'il fonctionne parfaitement, identifiez d'autres étapes inefficientes ou processus manuels à améliorer. [22].

2.2.2 Les avantages du BPM

Sans une vue d'ensemble des processus de votre entreprise, vous n'avez aucun moyen de savoir s'ils sont efficients et efficaces. Grâce au BPM, vous pouvez comprendre, analyser et améliorer vos processus métier. Lorsque vous établissez un modèle, vous décrivez votre processus idéal. Ensuite, si votre processus actuel n'y ressemble pas, vous comprenez pourquoi et l'améliorez.

N'oubliez pas que la gestion des processus métier ne se fait pas en un jour. Il s'agit plutôt d'un effort continu pour évaluer et améliorer vos processus. Par conséquent, vous pouvez apporter des améliorations significatives aux processus, améliorer l'efficience et l'efficacité, et permettre aux membres de votre équipe d'atteindre leurs objectifs plus rapidement et plus facilement Le BPM :

— schématise et améliore vos processus ;
— automatise les processus dans la mesure du possible ;
— réduit les pertes de temps ;
— élimine les points de blocage ;
— limite les erreurs ;

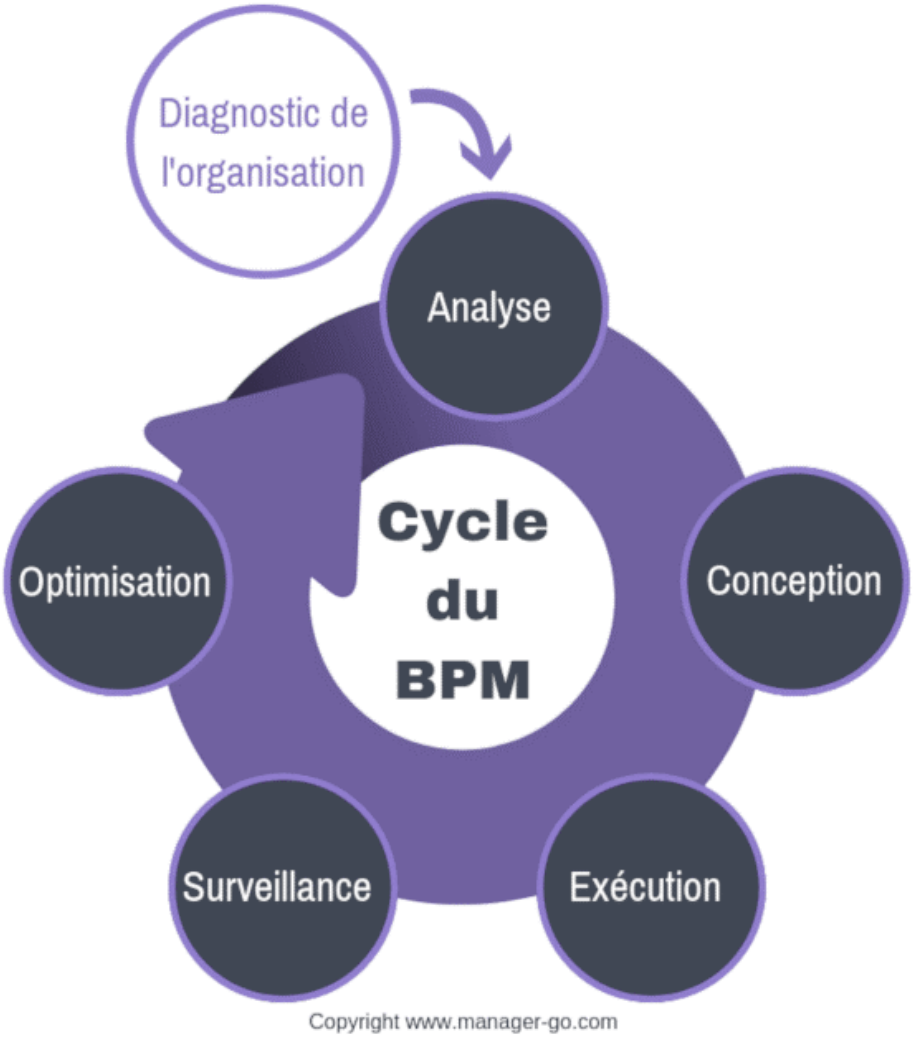


FIGURE 2.2 – Cycle du BPM [22]

— améliore l'efficience et l'efficacité ;
— génère des produits et services de meilleure qualité ;
— permet une meilleure satisfaction client ;
— simplifie les processus inefficients ;
— s'assure que vos processus métier contribuent clairement aux résultats de l'entreprise.

La gestion des processus métier n'est pas réservée aux grandes entreprises, même les petites équipes peuvent en profiter. Le BPM vous aide à optimiser les processus et à atteindre ces objectifs [17].

2.3 BPMN (Business Process Model and Notation)

BPMN (Business Process Model and Notation) est la notation la plus pratique et la plus largement utilisée dans la modélisation de processus. La notation BPMN est également une norme de notation internationale. BPMN peut certainement modéliser des diagrammes de processus complexes de manière agile et intuitive. En bref, BPMN est une technique de programmation d'organigrammes standardisée qui permet de créer et de partager des diagrammes faciles à comprendre. Ce dernier permet de modéliser visuellement les étapes d'un processus métier de bout en bout.

2.3.1 Histoire du BPMN

BPMN a été conçu et développé par le Business Process Management Institute (BPMI) et est géré par l'Object Management Group (OMG) depuis 2005. Il s'agit de la quatrième itération et la version 2.0 est sortie en 2010. Créer des règles plus détaillées pour la modélisation des processus métier à l'aide d'un certain nombre de symboles et de notations de diagramme de processus métier .

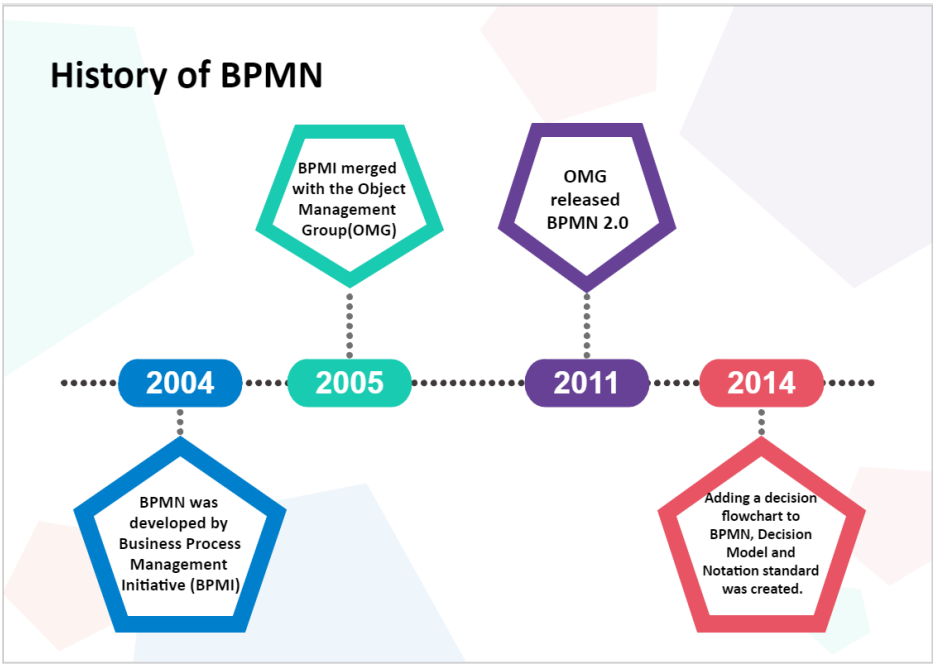


FIGURE 2.3 – Histoire du BPMN [19]

2.3.2 Les caractéristiques du BPMN

L'un des principaux avantages de BPMN est que la programmation des organigrammes peut être aussi simple ou complexe que nécessaire. Par conséquent, il est facile pour les parties prenantes à tous les niveaux (techniques ou non) de les comprendre. C'est probablement cette caractéristique qui explique la popularité du BPMN. Selon une enquête de 2016, 64 pourcent des entreprises souhaitent adopter le BPMN pour simplifier leurs processus métier. Dans la plupart des cas, l'objectif est simple. Économisez de l'argent en réduisant les coûts et en augmentant la productivité. [19].

2.3.3 Utilisation du BPMN

Le BPMN a pour objectif d'offrir à chacun une vision claire d'un processus de A à Z. Il fournit un parcours visuel permettant de combler les lacunes en montrant la séquence d'activités métier nécessaires pour passer d'un bout processus à l'autre Voici certains des avantages que l'adoption du BPMN peut apporter aux entreprises :

— Simplification de la communication et de la collaboration pour atteindre un objectif
— Représentation visuelle simple des étapes
— Possibilité de personnaliser le modèle selon le rôle des intervenants (analystes, participants au processus, responsables, développeurs, équipes externes, consultants, etc.)
— Identification des problèmes des processus qui nécessitent une solution
— Informations sur les pistes d'amélioration possibles
— Encouragement à la production de meilleurs résultats [18].

2.3.4 Éléments et symboles du BPMN

La notation BPMN comporte cinq catégories basiques d'éléments ainsi que de nombreux symboles et formes différents. En voici un aperçu :

1. **Objets de flux** : Ils illustrent un comportement dans un processus métier. Les voici :
   — Activités : travail effectué ou tâches d'une personne ou d'un système (représentées par un rectangle aux angles arrondis).
   — Événements : ce qui se passe durant un processus : Début, Intermédiaire et Fin (représentés par des cercles).
   — Branchements : ils décrivent le chemin de flux de séquence d'un processus (représentés par des losanges). Ils peuvent comprendre des informations supplémentaires, telles que les points de décision.
2. **Objets de données** : Ils fournissent des informations sur les données d'un processus. Les données sont représentées de quatre manières :
   (a) Entrées de données (représentées par une page avec un coin plié et une flèche vide vers la droite) : tâches dépendantes des données. Elles ne peuvent pas passer à l'étape suivante tant que des données spécifiques ne sont pas collectées.
   (b) Sorties de données (représentées par une page avec un coin plié et une flèche pleine vers la droite) : permettent d'illustrer les moments où le processus génère des données.
   (c) Collecte de données (représentée par une page avec un coin plié et trois lignes centrales pleines en bas) :
   (d) Stockage de données (représenté par un conteneur) : utilisé en tant qu'emplacement de collection des données recueillies via le processus [20].
3. **Objets de connexion** : Ils connectent les objets de flux entre eux, ou avec d'autres informations, et illustrent le flux d'un processus :
   — Flux séquentiel (représenté par une flèche pleine vers la droite) : décrit l'ordre dans lequel les activités sont réalisées.
   — Flux de message (représenté par une flèche pointillée vers la droite avec un cercle à l'extrémité gauche) : décrit les messages et l'ordre du flux entre les participants.

— Association (représentée par une ligne pointillée) : connecte le texte et les artefacts à un événement[7].

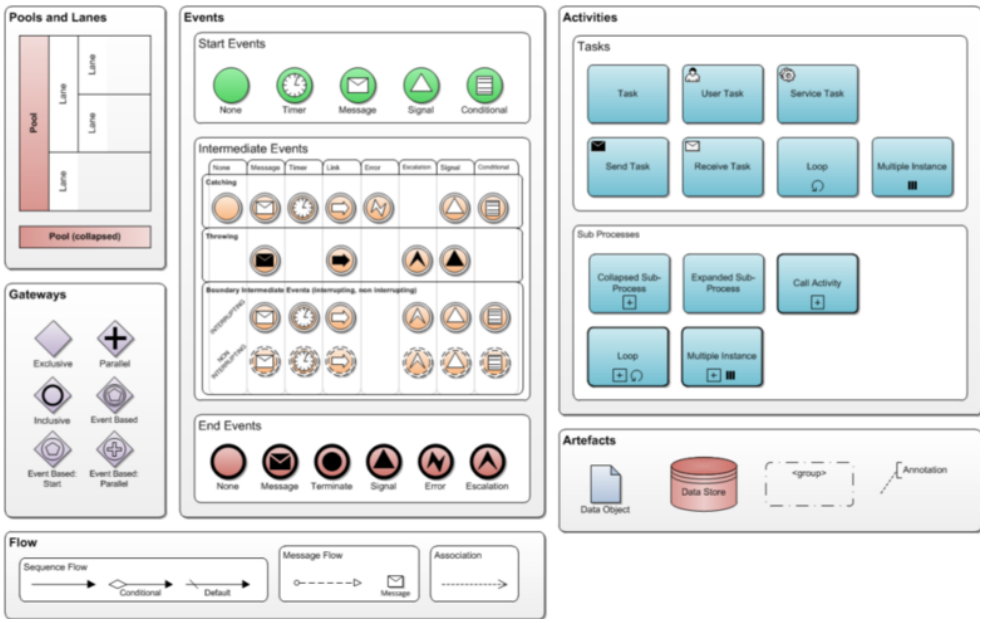


FIGURE 2.4 – Un sous-ensemble de base d'éléments BPMN [7]

4. **Pool and swimlane**

   Une swimlane représente les principaux participants d'un processus. Une autre swimlane peut se trouver dans une entreprise ou un service différent, mais être toujours impliqué dans le processus. Les couloirs d'une swimlane montrent les activités et le flux pour un certain rôle ou participant, définissant qui est responsable de quelles parties du processus[21].

5. **Artefact** : Informations supplémentaires que les développeurs ajoutent pour apporter un niveau de détail nécessaire au diagramme. Il existe trois types d'artefacts : les objets de données, les groupes ou les annotations. Un objet de données indique quelles données sont nécessaires à une activité. Un groupe montre un regroupement logique d'activités mais ne modifie pas le déroulement du diagramme. Une annotation fournit des explications supplémentaires sur une partie du diagramme [21].

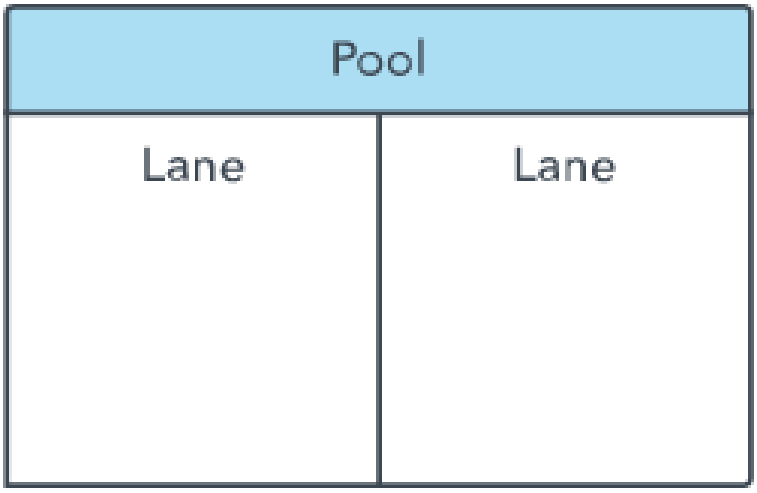


FIGURE 2.5 – pool and swimlane éléments BPMN [21]

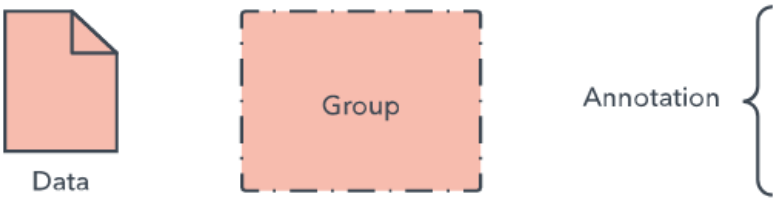


FIGURE 2.6 – Artefact éléments de BPMN [21]

2.3.5 Exemples d'un Diagramme BPMN

Le processus commence par un événement de type Timer qui lance le processus tous les vendredis, le gestionnaire de la liste des sujets traite et choisi ceux qui pourront rentrer dans le cycle de discussion ou non. L'activité (cycle de discussion) possède deux flux de séquences entrants et peut être entrepris si l'un des deux véhicule un flux et ce, en cas de présence de sujets à traiter ou suite à un besoin de rediscuter un sujet.

Le gestionnaire de la liste de sujets envoie un Email aux membres votants afin d'ouvrir une discussion sur les sujets (annonce des sujets de discussion). L'envoi des notifications de deadline est précédé par un Timer qui fixe un délai.

Les résultats sont admis si le taux des votants est acceptable. Si les votants représentent un faible taux, les sujets ne peuvent pas être résolus. Dans ce cas, une vérification concernant l'avertissement ou non des non-votants s'impose. Si le membre du groupe qui ne s'est pas présenté au vote n'était pas prévenu au préalable (pour le vote de la semaine en cours), alors une annonce de vote accompagnée d'un avertissement lui sont envoyés. Il est retiré de la liste des votants s'il ne participe pas à un second vote (dans les mêmes conditions) et le taux des votants est recalculé (les sujets sont reportés pour la semaine prochaine si le taux n'est pas concluant). Si le taux des votants est acceptable, le traitement des sujets proposés est vérifié afin de déterminer l'existence ou non d'une voix majoritaire et concluante. Si oui, le processus se termine, sinon le processus sera amené à ne conserver que les deux résultats les plus cotés et relancera dans le vote ceux qui n'avaient pas voté pour ces deux choix. Toutefois, une difficulté à déterminer ces deux résultats est possible, dans ce cas, on doit recourir à un nouveau cycle de vote et de discussion) [10].

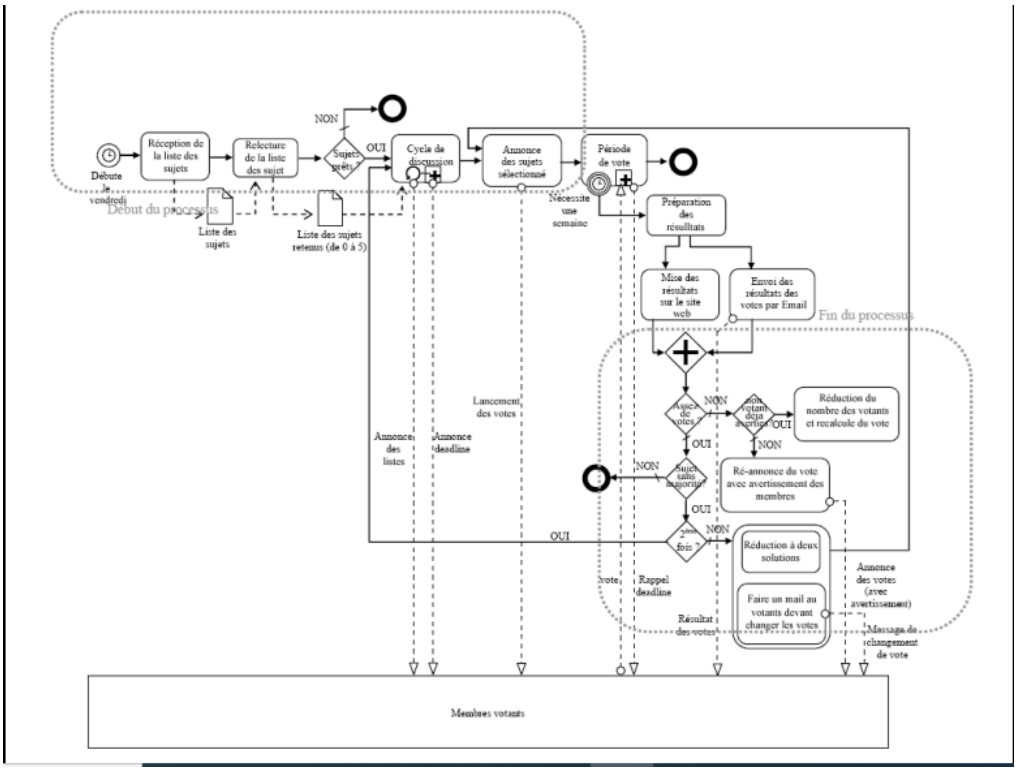


FIGURE 2.7 – Exemple Processus de vote par Email [10]

2.4 Conclusion

Dans ce chapitre , nous avons survolé les concepts fondamentaux relatifs aux processus métiers. Un intérêt particulier est accordé au langage BPMN que est la notation standard la mieux adaptée aux processus métier. La plupart des outils sur le marché prennent en charge ce langage. Nous constatons que malgré la puissance du BPMN , il soufre d'une limitation en ce qui concerne la capacité d'analyse et de vérification.

CHAPITRE 3

RÉSEAUX DE PÉTRI

3.1 Introduction

Les Réseaux de Pétri (RdP) sont des outils à la fois graphiques et mathématiques permettant de modéliser le comportement dynamique des systèmes à évènements discrets. Leur représentation graphique permet de visualiser d'une manière naturelle le parallélisme, la synchronisation, le partage de ressources...etc. Leur représentation mathématique permet d'analyser le modèle pour étudier ses propriétés et de les comparer avec le comportement du système réel. l'objectif de ce chapitre est de survoler les principaux concepts des réseaux de petri et montrer leur efficacité en tant qu'un outil formel pour l'analyse et la vérification des comportement particulièrement les processus métiers.

3.2 Historique

le réseau est présenté par Carl Adam Pétri dans sa thèse "Communication avec des Automates" en Allemagne à Bonn en 1962 ,Ce travail a continu à être développé par Anatol W. Holt, F. Commoner, M. Hack et leurs collègues dans le groupe de recherche

de Massachussetts Institute Of Technology (MIT) dans des années 70s. En 1975 la première conférence de réseau de Pétri et des méthodes relationnels ont été organisés à MIT. En 1981 le premier livre du réseau de Pétri a été publié en Anglais par J. Peterson. Aujourd'hui, suivre Pétri-Net Newsletter, chaque année il y a des 600 aux 800 d'œuvres des réseaux de Pétri sont publiés.

Les réseaux de Pétri fournissent un outil formel avec une bonne représentation graphique pour modéliser et analyser des systèmes discrets, y compris des systèmes concurrents et parallèles. L'intérêt majeur de ces réseaux réside dans leur possibilité d'analyser les systèmes modélisés. En effet, Ce formalisme bénéficie d'une multitude de techniques d'analyse et d'outils[1].

3.3 Définitions informelles de RdP

Un réseau de Pétri (RdP) est un graphe biparti orienté valué. Il a deux types de nœuds :

1. **Les places** : notées graphiquement par des cercles. Chaque place contient un nombre entier (positif ou nul) de marques (ou jetons). Ces derniers sont représentés par des points noirs.
2. **Les transitions** : notées graphiquement par un rectangle ou une barre. Une transition qui n'a pas de place en entrée est appelée transition source et une transition qui n'a pas de place en sortie est appelée transition puits.

Les places et les transitions sont reliées par des arcs orientés où :

1. Un arc relie, soit une place à une transition, soit une transition à une place mais jamais une place à une place ou une transition à une transition.
2. Chaque arc est étiqueté par une valeur (ou un poids), qui est un nombre entier positif. L'arc ayant k poids peut être interprété comme un ensemble de k arcs parallèles. Un arc qui n'a pas d'étiquette est un arc dont le poids est égal à 1 [13].

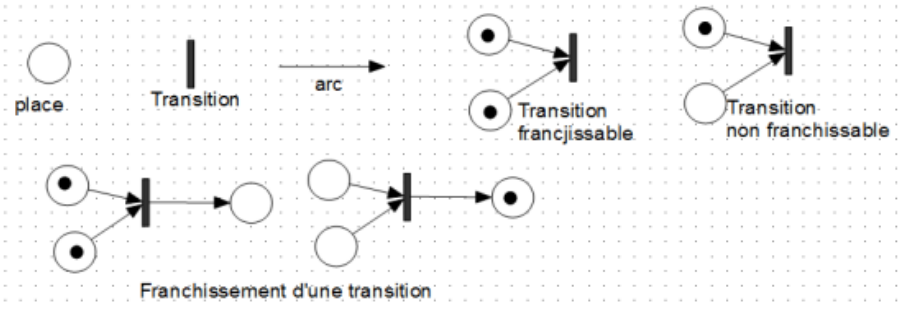


FIGURE 3.1 – Symboles et principes de base des Réseaux de Pétri [13]

3.3.1 Concepts de base pour les RdP

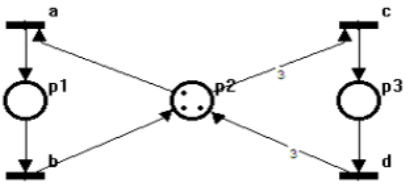
Il existe trios concepts de base

1. Condition : Une condition est un prédicat logique d'un état du système. Elle est soit vraie, soit fausse.
2. Evénement : Les événements sont des actions se déroulant dans le système. Le déclenchement d'un événement dépend de l'état du système. Un état du système peut être décrit comme un ensemble de conditions.
3. Déclenchement, précondition, post-condition :

   Les conditions nécessaires au déclenchement d'un événement sont les pré-conditions de l'événement. Lorsqu'un événement se produit, certaines de ses pré-conditions peuvent cesser d'être vrais alors que d'autres conditions, appelées post-conditions de l'événement deviennent vraies [1].

3.3.2 Exemple de RdP

Les matrices Pré (associée aux places précédentes) et Post (associée aux places suivantes) représentent respectivement les matrices obtenues en donnant le poids des arcs de chaque transition en fonction des places [11].



Type	Rôle	Représentation	Exemples
Nœuds	Place	Cercle	p1, p2, p3
	Transition	Trait	a,b,c,d
Arc	p → t	Flèche orientée	p2→a, p2→c, p1→b, p3→d
	t → p	Flèche orientée	a→p1, b→p2, c→p3, d→p2

FIGURE 3.2 – Exemple Réseau de Pétri [11]

3.4 Définitions formelles de RdP

Un Réseau de Petri (RdP) est une structure graphique comportant un ensemble de places et de transitions, reliées par des arcs orientés, éventuellement porteurs de poids. Ces arcs sont des liens entre place et transition ou entre transition et place exclusivement. Dans cette structure se déplacent des jetons (ou marques) qui apparaissent dans les places et sont susceptibles de franchir les transitions selon certains critères de franchissable et de franchissement

un réseau de pétri (R) est un triplet $R = (P, T, W)$ où P est l'ensemble des places (les places représentent les conditions) et T l'ensemble des transitions (les transitions représentent les événements ou les actions) tel que $P \cap T = Q$ et W est la fonction définissant le poids porté par les arcs où $W : ((PxT) \cup (TxP)) \rightarrow N = 0, 1, 2, ...$

- Le réseau R est fini si l'ensemble des places et des transitions est fini, c.-à-d. $|P \cup T| \in N$.

- Un réseau $R = (P, T, W)$ est ordinaire si pour tout $(x, y) \in ((PT) \cup (TP))$: $W(x, y) \leq 1$.

Dans un réseau ordinaire la fonction W est remplacée par F où : $F \subset ((PxT) \cup (TxP))telque(x, y) \in F \iff w(x, y)6 = 0$. Pour chaque $x \in P \cup T$:

*x représente l'ensemble des entrées de x :*x= $y \in P \cup T|W(y, x)6 = 0$

- x* représente l'ensemble des sorties de x :x*= $y \in P \cup T | W(y,x)6 = 0$

Les définitions concernant les réseaux de Pétri portées sur deux aspects :

Un aspect structurel : Quelles sont les actions, quels sont les sites, quelles sont les conditions pour qu'une action soit possible et quelles sont les conséquences d'une action.

Un aspect comportemental : Comment représenter le fonctionnement d'un réseau de Pétri ? C.-à-d. ce qui se passe quand une action ou plusieurs actions sont exécutées [14].

3.5 Qualités et faiblesses des réseaux de Pétri

En tant que formalisme de spécification, les réseaux de Pétri présentent un certain nombre de qualités très importantes.

— Ils disposent d'une définition formelle
— Ils présentent un grand pouvoir d'expression
— Ils sont exécutables
— Ils disposent de nombreuses techniques de vérification automatique des propriétés des modèles.
— Ils disposent d'une représentation graphique attrayante, qui accroît la lisibilité et facilite la compréhension des modèles.

Toutefois, malgré ces qualités, il semble que les réseaux de Pétri souffrent de certains reproches.

— Leur manque de structuration : l'utilisation des RDPs produirait des modèles dont la taille croît rapidement avec la complexité du système, et qui deviennent rapidement impossibles à comprendre et à gérer.
— Leur difficulté à prendre en compte l'aspect « structure de données » : A l' origine, les RDPs étaient essentiellement destinés à décrire la structure de contrôle d'un système et son évolution dynamique. Les réseaux Place/Transition échouent

en effet à décrire la structure des données manipulées par le système, leur fermeture à la modélisation de l'environnement extérieur et la manière dont ces données peuvent influencer le comportement dynamique.

Ces reproches ont certainement été légitimes à une époque, mais ils ont été entendus par la communauté RDPs, qui a développé des techniques de structuration des réseaux, et qui a traité le problème de la prise en compte des données en développant la théorie des réseaux de Petri de haut niveau [14].

3.6 Représentation de RDP

• Représentation graphique de RdP Un réseau de Petri se représente par un graphe biparti (composé de deux types de nœuds et dont aucun arc ne relie deux nœuds de même type) orienté (composé d'arc(s) ayant un sens) reliant des places et des transitions (les nœuds). Deux places ne peuvent pas être reliées entre elles, ni deux transitions. Les places peuvent contenir des jetons, représentant généralement des ressources disponibles.

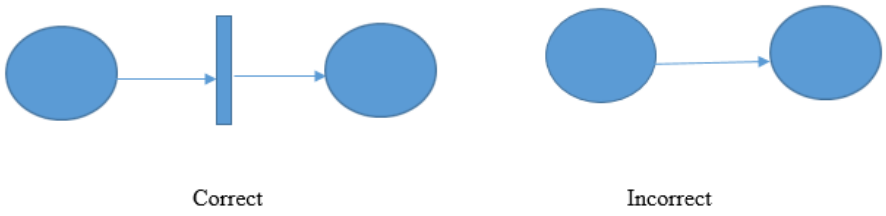


FIGURE 3.3 – types de nœuds

La distribution des jetons dans les places est appelée le marquage du réseau de Petri. Les entrées d'une transition sont les places desquelles part une flèche pointant vers cette transition, et les sorties d'une transition sont les places pointées par une flèche ayant pour origine cette transition. Le réseau de pétri est représenté par deux type de sommets alternés, les places Pi et les transitions Ti. Ces sommets sont reliés par des

arcs orientés. Tout arc doit relier une place à une transition ou bien une transition à une place [1].

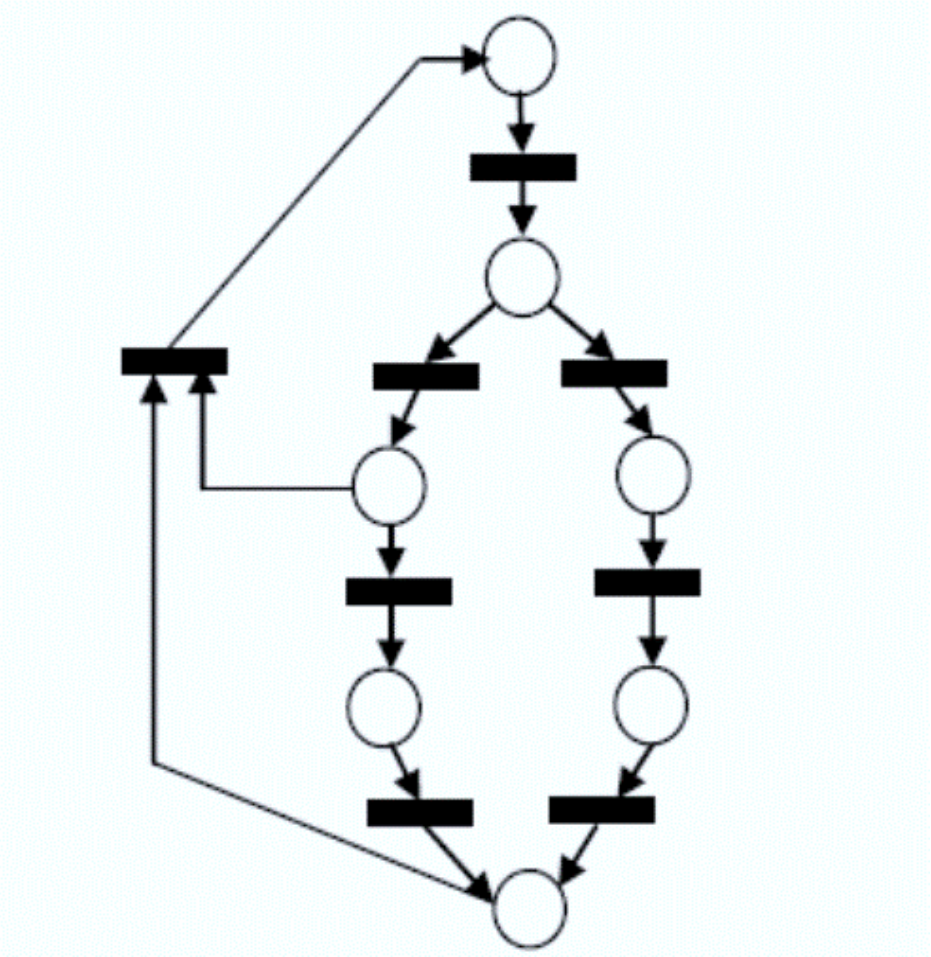


FIGURE 3.4 – Représentation graphique d'un réseau de Petri [1]

• Représentation matricielle Une représentation matricielle d'un RdP est offerte afin de simplifier les tâches d'analyse et de vérification effectuée sur un modèle RdP. Agir sur une représentation graphique d'un modèle RdP est une tâche délicate en le comparant avec une représentation matricielle. Il est possible de représenter la fonction W (fonction de poids) par des matrices

D'autre part un réseau de pétri R= (P, T, W) avec P=p1, p2, . . . , pm et T= t1, t2, . . ,tn, on appelle matrice des pré-conditions pré, la matrice m x n à coefficients dans N telle que pré (i,j)= W(pi, tj) indique le nombre de marques que doit contenir

la place pi pour que la transition tj devienne franchissable. De la même manière on définit la matrice des post conditions-post, la matrice n × m telle que post (i,j)= W(tj, pi) contient le nombre de marques déposées dans pi lors du franchissement de la transition tj. La matrice C= post – pré est appelée matrice d'incidence du réseau (m représente le nombre de places d'un réseau de pétri et n le nombre de transitions). Le marquage d'un réseau de pétri est représenté par un vecteur de dimension m à coefficients dans N. La règle de franchissement d'un réseau de pétri est définie par : M' (p) =M (p) + C (p, t) . [07] La figure représente un réseau de pétri marqué avec un vecteur de marquage M tel que : M = (1, 0, 0,0).

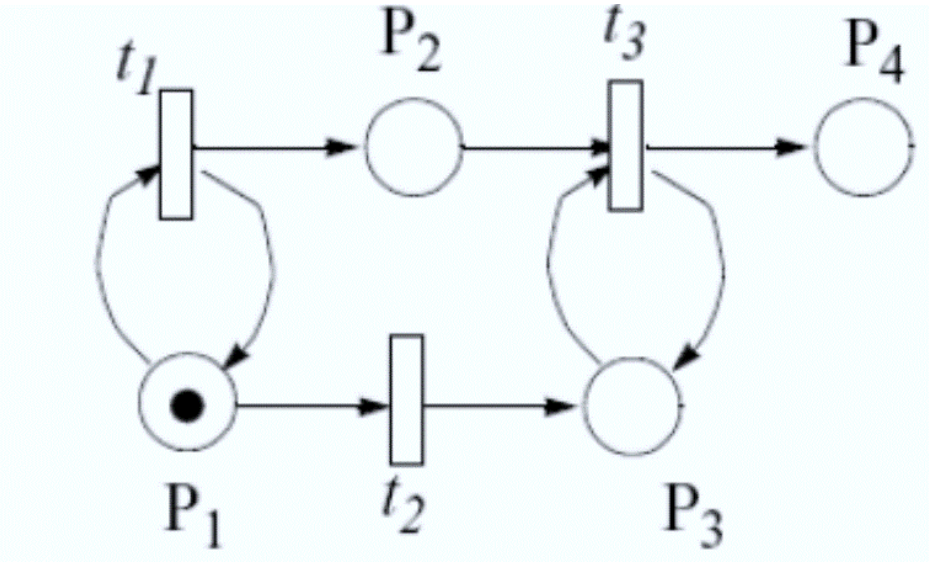


FIGURE 3.5 – un réseau de pétri marqué avec un vecteur de marquage [1]

Pour le réseau ci-dessus P= p1, p2, p3, p4 T= t1, t2, t3, la représentation matricielle est donnée ci-dessous du RdP de la figure suivante.

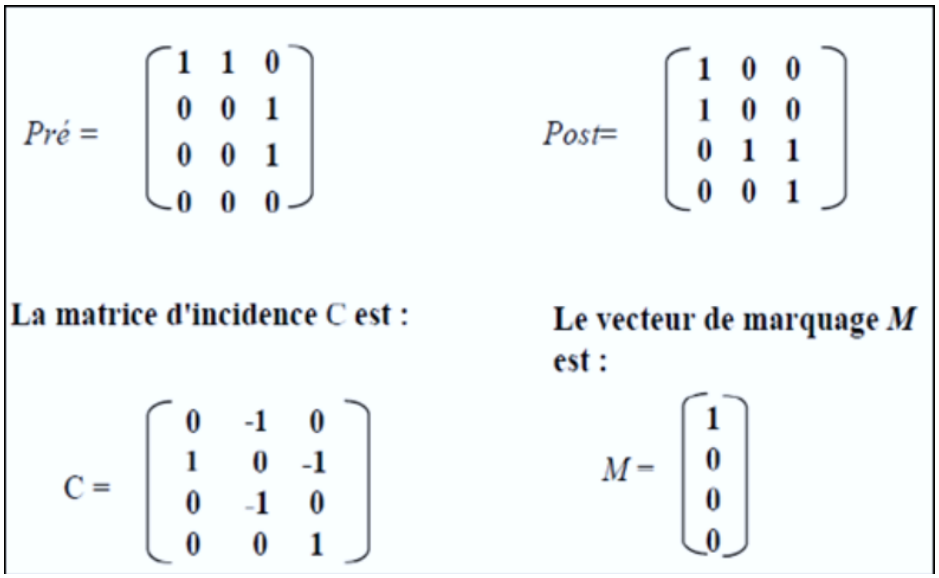


FIGURE 3.6 – la représentation matricielle est donnée ci-dessous du RdP [1]

3.7 Méthodes d'analyse pour les réseaux de pétri

Les méthodes d'analyse des RdP peuvent être classées en trois groupes : • Méthode d'arbre de couverture • Approche d'équations matricielles • Technique de réduction et de décomposition.

3.7.1 Méthode d'arbre de couverture

L'idée la plus naturelle pour étudier les propriétés d'un RdP est de construire le graphe de tous les marquages accessibles. Le graphe des marquages accessibles est un graphe dont chaque sommet correspond à un marquage accessible et dont chaque arc correspond au franchissement d'une transition permettant de passer d'un marquage à l'autre. Pour un RdP marqué (partpartir de chaque nouveau marquage on peut accéder à d'autres nouveau marquages. Le résultat de ce processus est un arbre

dont chaque nœud est un marquage accessible et chaque arc est une transition franchissable qui transforme un marquage à un autre L'arbre de couverture est une méthode alternative. Le mécanisme de construction est le même que pour le graphe des marquages accessibles. A ceci près que pour chaque nouveau marquage (nœud du graphe) ajouté, on vérifie s'il n'est pas supérieur à un marquage déjà présent sur au moins une séquence entre et le nouveau marquage. Si tel est le cas tous les marquages de place supérieurs sont remplacés par Ce symbole matérialise le fait que la place en question peut contenir autant de jetons que souhaité (elle est donc non bornée). Dans la suite, les places non bornées le demeurent naturellement et ceci quelles que soient les transitions franchies, ainsi le symbole ne disparaît jamais. Cette méthode produit le graphe de couverture, un graphe fini dans tous les cas [11].

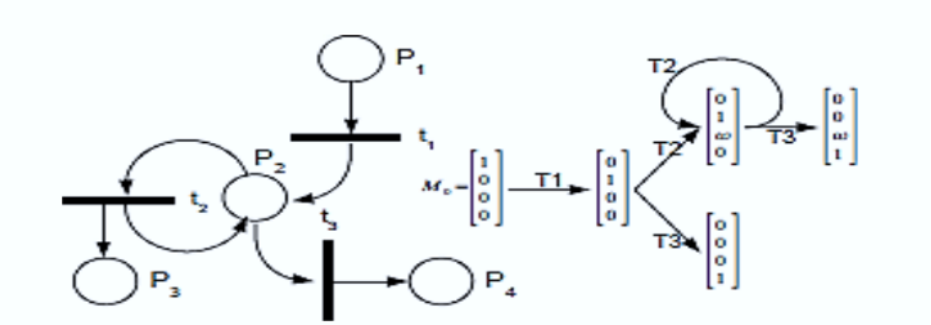


FIGURE 3.7 – RdP et leur arbre de couverture [11]

3.7.2 Approche d'équations matricielles

L'analyse par équations matriciel (algèbre linéaire) permet d'étudier des propriétés d'un réseau (caractère borné, vivacité) indépendamment d'un marquage initial. De ce fait, on parlera de propriétés structurelles du réseau. Par exemple, on pourra dire qu'un réseau est structurellement borné s'il est borné pour tout marquage initial fini. De la même façon, si pour tout marquage initial, le réseau est vivant, on dira que le réseau est structurellement vivant.

3.7.3 Technique de réduction et de décomposition

L'analyse des RDPs par réduction permet d'obtenir à partir d'un RdP marqué, un RdP marqué plus simple, c'est-à-dire avec un nombre réduit de places et un nombre réduit de transitions ceci en appliquant un ensemble de règles dites règles de réduction [11].

3.8 L'exécution d'un réseau de Petri

• Séquence de franchissement

Une séquence de franchissement « s » est une suite de transitions (t1, t2, . . ., tn) qui permet, à partir d'un marquage « M », de passer au marquage « M' » par le franchissement successif des transitions définissant la séquence.

• Marquage accessible

Le marquage d'un Réseau de Petri à un instant donné est une vectrice colonne dont la valeur de la ième composante est le nombre de marques de Pi a cet instant.

Le franchissement d'une transition conduit à un changement du marquage. Le passage du marquage Mk au marquage Ml par franchissement de la transition TJ est noté : Mk (Tj > Ml). le nombre de marques dans la place Pi pour le marquage Mk est noté (Pi) . A partir d'un Même marquage, il peut être possible de franchir plusieurs transitions, menant ainsi à des marquages différents. L'ensemble des marquages accessibles à partir du marquage M0 est l'ensemble des marquages obtenus à partir de M0 par franchissements successifs d'une ou plusieurs transition(s). Cet ensemble est noté (RM0).

• Graphe de marquage

On peut représenter l'ensemble de marquage accessible par un graphe si ce dernier et fini. Le graphe de marquage a comme sommet l'ensemble de marquage accessible (R,M0) Un arc orienté relie deux sommet Mi et Mj s'il existe une transition t franchissable permettant de passer d'un marquage à un autre Mi (t>Mj) Les arcs du graphe sont étiqueté par les transitions correspondantes [11].

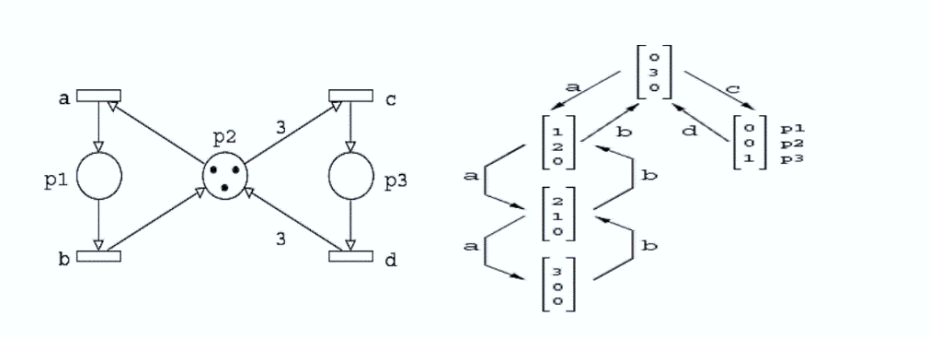


FIGURE 3.8 – RdP et leur graphe de marquage [11]

• L'exécution séquentielle d'un réseau de Petri

L'exécution séquentielle d'un réseau de Petri est définie en termes d'un ensemble de séquences d'occurrence. Une séquence d'occurrence est une séquence de transitions franchissables dénotée par $\delta = M0t1M1t2...telqueMi-1(ti > Mi).Unesquencet1t2...estunes$ $M0.Silasquencefiniet1t2...tnconduitunnouveaumarquageMpartirdumarquageM.$

• Exécution concurrente

Une exécution concurrente d'un réseau de Pétri est une exécution dans laquelle plusieurs transitions peuvent se franchir au même temps, elle est souvent déterminée par la notion de processus. Ceci permet de donner une interprétation de la concurrence dans un réseau de Pétri selon la sémantique basée sur la vraie concurrence (sémantique d'ordre partiel) qui est interprétée dans la théorie des réseaux de Pétri par un type spécial de réseaux appelés réseaux d'occurrences [11].

3.9 Propriétés des RdP

• Réseau K-borné

Une place Pi est bornée pour un marquage initial M0 si pour tout marquage accessible à partir de M0 , le nombre de marques dans Pi reste borné. Elle est dite k- bornée si le nombre de marques dans Pi est toujours inférieur ou égal à k. Un RdP marqué est (k) borné si toutes ses places sont (k) bornées. Un RdP marqué peut ne pas être borné : sur l'exemple représente, la transition T1 admet la place P1 comme unique

place d'entrée. La place P1 a une marque : la transition T1 est franchissable. Comme P1 est aussi place de sortie de T1, le franchissement de T1 ne change pas le marquage de P1. La transition T1 est donc franchissable en permanence et peut donc être franchie un nombre de fois infini. Chaque franchissement de T1 ajoutant une marque dans la place P2, le marquage de celle-ci peut donc tendre vers l'infini.[13].

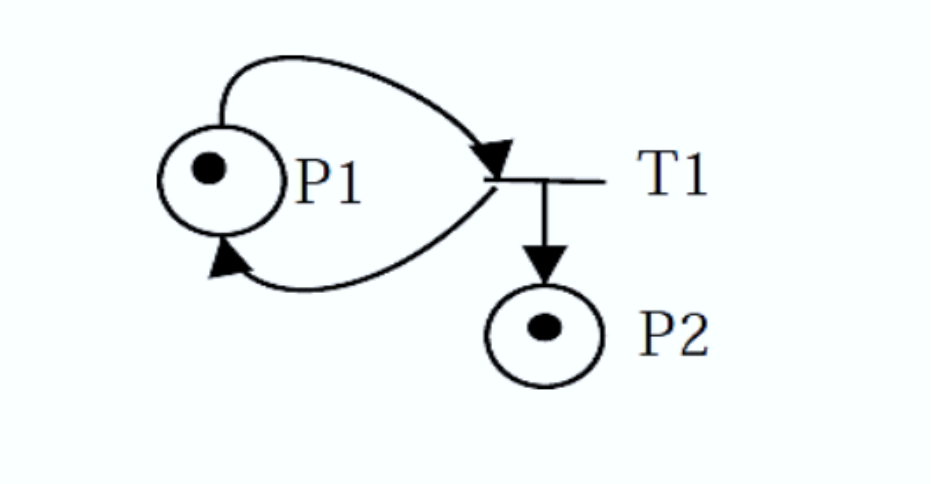


FIGURE 3.9 – RdP K-borné [13]

• Réseau vivant

L'évolution du marquage d'un RdP se fait par franchissement de transitions. Lorsqu'au cours de son évolution, certaines transitions ne sont jamais franchies, cela indique que l'évènement associé à la transition ne se produit pas et que le marquage d'une partie du RdP n'´évolue pas. Cela indique que le sous-système modélisé par cette partie-là ne fonctionnera pas. Il y a donc un problème au niveau de la conception du système. L'idée est d'être capable de d´détecter systématiquement ce phénomène par l'analyse de propriétés du modèle RdP du système afin de disposer d'un outil d'aide à la conception des systèmes. Si une transition Tj est vivante alors, à tout instant, on sait que Tj peut être franchie dans le futur. Dans le cas d'un RdP modélisant un système fonctionnant en permanence, si une transition n'est pas vivante et si une fonction du système est associée au franchissement de cette transition, cela veut dire qu'à partir d'un certain instant, cette fonction ne sera plus disponible dans le futur, ce qui peut traduire une erreur ou une panne [13].

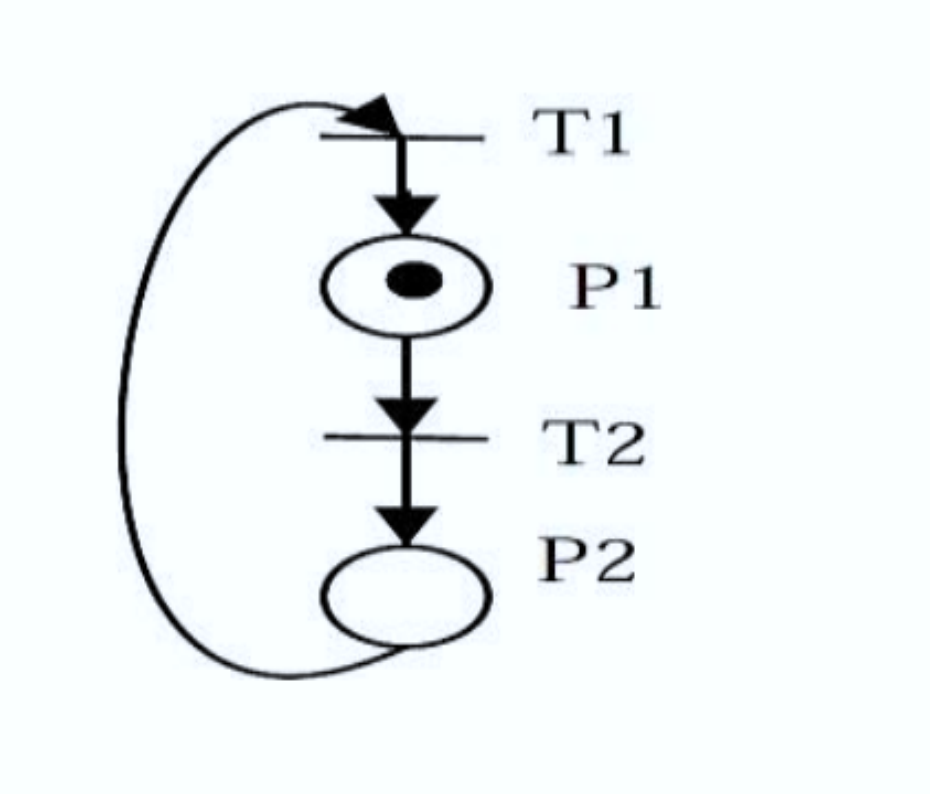


FIGURE 3.10 – RdP vivant [13]

• Réseau réinitialisable

Soit un réseau de Petri marqué < M0 > et soit (R,M0) l'ensemble de ses marquages accessibles. Ce réseau marqué est réinitialisable si et seulement si : $\forall M \in (M0), Mnon = M0, \exists StelqueMS \rightarrow M0OnditgalementqueM0estuntatd'accueil$

(en anglais "home state") pour le réseau de Petri marqué. Soit un réseau de Petri marqué < M0 > s'il est réinitialisable et que toute ses transitions sont quasi-vivantes

ou vivantes, alors il est vivant [13].

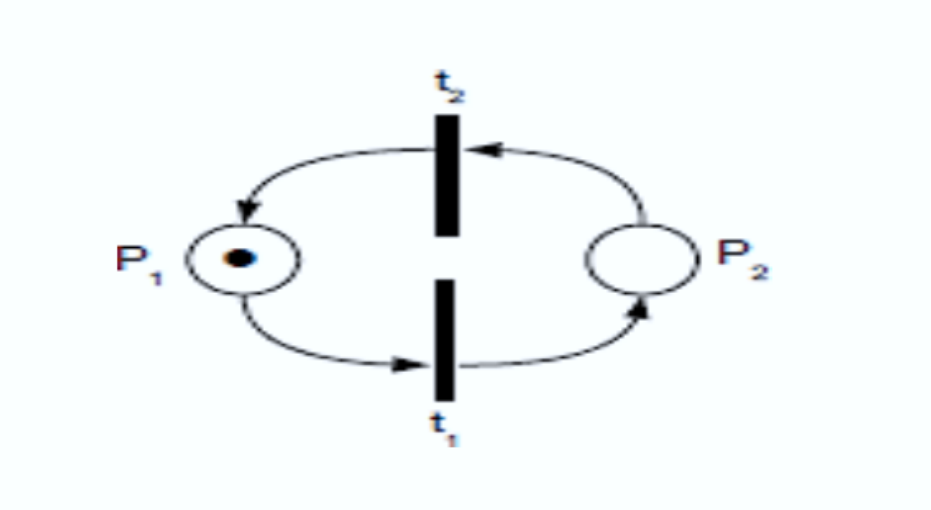


FIGURE 3.11 – RdP réinitialisable [13]

3.10 Marquage d'un Réseau de Petri

Le marquage d'un RdP est précisé par la présence à l'intérieur des places d'un nombre fini (positif ou nul), de marques ou de jetons. Une place est donc vide ou marquée. Lorsque la place représente une condition logique (ex. : machine à l'arrêt, convoyeur en panne), la présence d'un jeton indique que cette condition est vraie ; fausse dans le cas contraire. Une place donc peut représenter une ressource du système (un stock par exemple), elle peut contenir plusieurs jetons (dans l'exemple du stock, le nombre de jetons peut indiquer le nombre de pièces stockés).

Les méthodes d'analyse des réseaux de pétri prennent pleine puissance avec leurs mise en œuvre par le biais d'un ensemble d'outils d'analyse tels que : INA (Integrated Net Analyzer), PEP (Programming Environment based on Pétri nets), TINA (TIme Pétri Net Analyzer) etc [1].

3.11 Utilisation des réseaux de pétri

Dans certains cas, les réseaux de pétri ordinaires ne peuvent exprimer toutes les propriétés que nous voudrons modéliser, et de ce fait, des extensions s'avèrent utiles afin de pallier à ces insuffisances. Parmi les extensions les plus utilisées, nous citons :

• Les réseaux de pétri généralisés : Un réseau de pétri généralisé est un réseau de pétri dans lequel des poids (nombres entiers strictement positifs) sont associés aux arcs.

• Les réseaux de pétri temporisés : C'est une extension temporelle des réseaux de pétri.

• Les réseaux de pétri colorés : Dans un réseau de pétri coloré, on associe une valeur à chaque jeton.

• Les réseaux de pétri continus : Le nombre de jetons dans un réseau de pétri continu est un réel positif. Le franchissement s'effectue comme un flot continu en introduisant la notion de vitesse traduite par le nombre de marques franchies pendant une unité de temps. Quelques utilisations des réseaux de pétri dans les différents domaines, non seulement informatiques, ont été résumées dans la table [13].

Réseau de Pétri ordinaire	- Modélisation des systèmes - Modélisation des processus d'affaires - Gestion des flux. - Programmation concurrente. - Génie de la qualité - Diagnostic
Réseau de Pétri généralise	- Gestion des flux complexes. -Modélisation de chaines logistiques. -Utilisation pour les techniques quantitatives.
Réseau de Pétri temporisé	- Gestion du temps. -Modélisation d'attentes.
Réseau de Pétri coloré	- Modélisation des systèmes de collaboration.
Réseau de Pétri continu	- Modélisation des réactions chimiques.

FIGURE 3.12 – utilisations des réseaux de pétri dans les différents domaines [13]

3.12 Propriétés des réseaux de pétri

Les propriétés génériques informent le développeur sur le comportement général du réseau de pétri. Celles-ci doivent être complétées par l'analyse des propriétés spé- cifiques du système modélisé. Ces propriétés sont souvent spécifiées par des formules logiques telles que la logique temporelle linéaire (LTL) et la logique temporelle arborescente (CTL). La vérification des propriétés nécessite la construction du graphe d'accessibilité.

• Propriétés génériques

La vivacité, le caractère borné et la réinitialisable sont les principales propriétés génériques qui peuvent être automatiquement vérifiées dans un réseau de pétri.

Vivacité et blocage :

Un réseau de pétri est vivant si et seulement si toutes ses transitions sont vivantes. Une transition t d'un réseau de pétri ayant M0 comme marquage initial est dite vivante si pour tout marquage M du réseau de pétri atteignable à partir de M0, nous pouvons toujours trouver une séquence franchissable de transitions dans laquelle la transition t figure. La vivacité d'un réseau de pétri dépend de son marquage initial M0. Un réseau de pétri vivant garantit l'absence de blocages et de parties mortes (non atteintes) dans la structure du réseau. Il garantit ainsi la possibilité de toujours atteindre les services du système modélisé dans le réseau.

Réseau borné, Réseau Sauf :

Un réseau de pétri est borné si et seulement si toutes ses places sont bornées. Une place p est bornée si pour tout marquage M, le nombre de jetons dans la place est inférieur à une constante k : 8 M, M(p) < k. Le caractère borné d'un réseau de pétri renseigne sur les valeurs limites des ressources demandées par le système. Si un réseau est borné, et la borne est égale à 1, alors il est dit sauf

• Conflit

Un conflit structurel correspond à un ensemble d'au moins deux transitions t1 et t2 avec une place d'entrée en commun. Ceci est noté comme suit : k = < p , t1,t2,. . . ,tn

>. Un conflit effectif est l'existence d'un conflit structurel k, et d'un marquage M, tel que le nombre de marques dans p est inférieur au nombre de transitions de sortie de p qui sont validées par M.

• Réinitialisabilité :

Un réseau de pétri est réinitialisable si et seulement si pour tout marquage M, il existe une séquence de transitions qui permet de revenir au marquage initial M0. Cette propriété renseigne sur le fonctionnement répétitif, ce qui est pertinent pour la majorité des systèmes interactifs.

• Propriétés spécifiques

Les propriétés spécifiques sont regroupées en quatre types : Les propriétés d'accessibilité, de sûreté, de vivacité et d'équité.

• L'accessibilité (reachability) : détermine si une situation est accessible ou non à partir du graphe d'accessibilité.

• La sûreté (safety) : certaines situations ne devront jamais être atteintes.

• La vivacité (liveness) : une situation finira tôt ou tard par avoir lieu.

• L'équité (fairness) : une situation aura lieu une infinité de fois [4].

3.13 Outils de modélisation des RDPS

L'aspect formel des RdPs a encouragé les développeurs à mettre au point une multitude d'outils de simulation et de vérification des RdPs selon la technique de vérification de modèle. On propose de faire un résumé sur les fonctionnalités qu'offre chacun d'eux Le tableau ci-dessous les classe par rapport aux points suivants.

• Présence d'un environnement graphique d'édition des RDPs.

• Possibilité de Simulation du RDP.

• Possibilité d'analyse des propriétés génériques du RDP.

• Possibilité de vérification des contraintes CTL (logique temporelle arborescente).

• Et LTL (logique temporelle linéaire).

• Possibilité de supporter le format d'échange XML.

- Possibilité de supporter les RDP hiérarchiques [5].

Outil	InterfaceGraphique	Simulation	Analyse	CTL	LTL	XML	RDPhiérarchique	TOTAL(oui)
CPNTools	Oui	Oui	Oui	Oui	Non	Oui	oui	6
CPNAMI	Oui	Oui	Oui	Oui	Oui	Non	non	5
PROD	Non	Non	Oui	Oui	Oui	Non	non	3
JARP	Oui	Non	Oui	Non	Non	Oui	non	3
Maria	Non	Non	Oui	Oui	Oui	Non	non	3
LoLa	Non	Non	Oui	Oui	Non	Non	non	2
Petri NetKernel	Oui	Non	Non	Non	Non	Oui	non	2
Great SPN	Oui	Non	Oui	Non	Non	Non	Non	2
INA	Non	Oui	Oui	Oui	Non	Non	Non	2
OPMSE	Oui	Oui	Non	Non	Non	Non	non	2
TimeNet	Oui	oui	oui	-	-	oui	-	4

FIGURE 3.13 – outils de modélisation des RDPS [5]

3.14 Conclusion

Dans ce chapitre, nous avons présenté les réseaux de Petri de haut niveau que nous avons vus. Les réseau de petri offre la possibilité de modéliser le comportement dynamique d'un système et donne la possibilité à faire des analyses formelles la-dessus. Nous essayons dans le prochain chapitre de proposer notre approche consistant à générer automatiquement un réseau de petri à partir un BPMN et laissant le soin aux analystes d'exploiter nos résultats pour effectuer leurs études et les répercuter sur le modèle du processus métier initial.

CHAPITRE 4
CONTRIBUTION

4.1 Introduction

Les opérations de transformation manuelles ou automatiques ne sont pas nouvelles pour les développeurs d'applications informatiques. En effet, la compilation (génération de code en langage machine) est une transformation automatique d'un langage de programmation de haut niveau, alors que la programmation n'est rien d'autre qu'une transformation manuelle de patrons de conception en un langage de programmation de haut niveau (langage JAVA).

Dans ce chapitre nous présenterons le principe de notre approche, qui consiste à transformer le BPMN vers les RDP en se basant sur l'approche de l'ingénierie dirigée par les modèles. Nous commencerons, par établir un Meta-modèles EMF associé au BPMN et un deuxième méta modèle associé au RDP. Ensuite, nous proposons nos règles de transformation avec le langage de transformation ATL. À la fin nous avons présenté un cas d'étude afin de mettre en évidence le résultat obtenu.

4.2 Definitions de Transformation

Une transformation est la génération automatique d'un modèle cible à partir d'un modèle source selon une définition de transformation . Une définition de transformation est un ensemble de règles de transformation qui décrivent ensemble comment transformer le modèle dans la langue cible . Une règle de transformation est une description de la façon de transformer une ou plusieurs structures de du modèle source en une ou plusieurs structures de la modèle cible . La figure 4.1 illustre ces deux étapes de transformation du modèle [9].

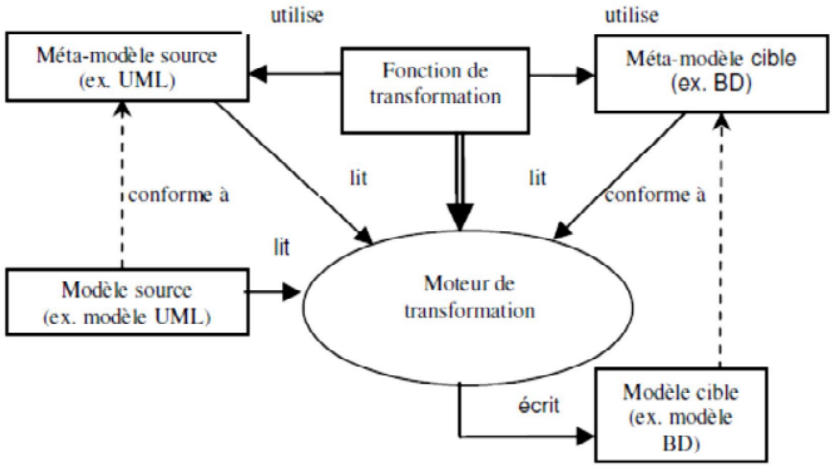


FIGURE 4.1 – Schéma de base d'une transformation de modèles [9]

4.3 Approches de transformation

Pour effectuer la transformation, plusieurs approches ont été proposées dans la littérature. Nous les résumons dans les points suivant :

— **Approches par programmation** : la transformation est décrite sous forme d'un programme informatique à l'image de n'importe quelle application informatique.

— **Approches par templates** : Les approches basées sur des templates sont utilisées dans la génération du code source à partir des modèles.

— **Approches par structure** : Ces approches consistent à définir des canevas des modèles cibles souhaités. Ces canevas sont des modèles cibles paramétrés. L'exécution d'une transformation consiste à prendre un modèle cible paramétré et à remplacer ses paramètres par les valeurs d'un modèle source.
— **Approches relationnelle** : Ces approches utilisent les contraintes pour spécifier les relations entre les éléments du modèle source et ceux du modèle cible en utilisant une logique déclarative basée sur des relations mathématiques .
— **Approches par modélisation** : Elles consistent quant à elles à appliquer les concepts de l'ingénierie des modèles aux transformations des modèles elles-mêmes. L'objectif est de modéliser les transformations de modèles et de rendre les modèles de transformation pérennes et productifs, en exprimant leur indépendance vis-à-vis des plates-formes d'exécution.

4.4 Langages de transformation de modèles

Chaque approche de transformation de modèle requiert un langage de transformation. Une série de critères caractérisent un langage de transformation.

— être exécutable ;
— avoir une implémentation efficace.
— être expressif et non ambiguë.
— fournir une description précise, concise et claire des règles de transformations.
— différencier les règles de sélection des éléments du modèle source et les règles de transformation.
— proposer des constructions graphiques (une notation visuelle est plus concise et intuitive qu'une notation textuelle).
— proposer la possibilité de faire des transformations composites à l'aide des opérateurs de séquences, de conditions et de répétions.
— offrir la possibilité de faire des transformations bidirectionnelles et une mise à jour incrémentale des transformations (possibilité de reporter les changements

effectués sur un modèle source dans le modèle cible). Le QVT (Queries, Views, Transformations) et l'ATL (Atlas Transformation Language) sont les langages les plus utilisés à ce jour[9].

4.5 Mise en oeuvre de notre approche

4.5.1 Manipuler des modéles avec EMF

EMF permet de stocker les modèles sous forme de fichier pour en assurer la persistance. EMF permet également de traiter différents types de fichiers : conformes à des standards reconnus (UML, XML, XMI).

FIGURE 4.2 – Eclipse Modeling Framework (EMF)

L'objectif général du frawork EMF est de proposer un outillage qui permet la modélisation et la métamodélisation. EMF propose plusieurs services :

— La transformation des modèles d'entrées, présentés sous diverses formes, en Core Model (Ecore Model).
— La gestion de la persistance du Core Model sous format XMI.
— la transformation du Core Model en code Java. EMF peut de base gérer en entrée des modèles présentés sous trois formats :UML ,XMI ,Code Java Annoté

Dans EMF, il est possible de définir un méta-modèle et de générer les interfaces afin de pouvoir manipuler les instances du méta-modèle dans Eclipse [26].

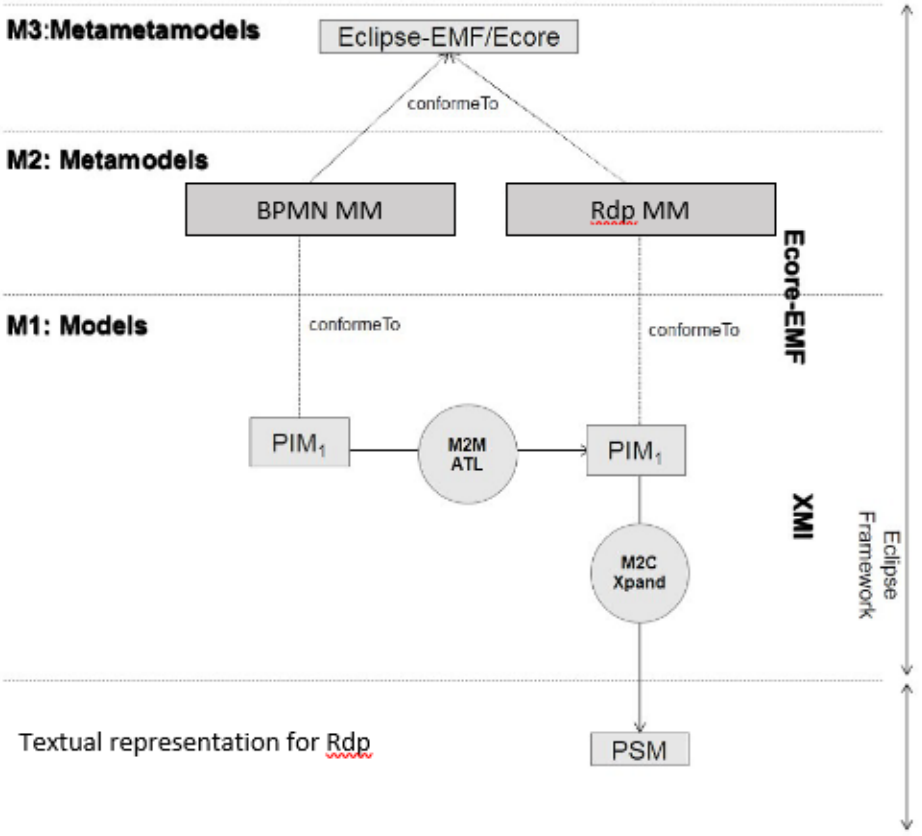


FIGURE 4.3 – Outils utilisés pour implémenter l'approche proposée[26]

4.5.2 Mise en oeuvre des méta-modèles (Ecore)

Cette section présente les méta-modèles sources qui sont le BPMN, et le méta-modèle cible qui est les réseaux de Petri.

Méta Modèle BPMN

La figure 4.4 capture le méta-modèle EMF du BPMN.

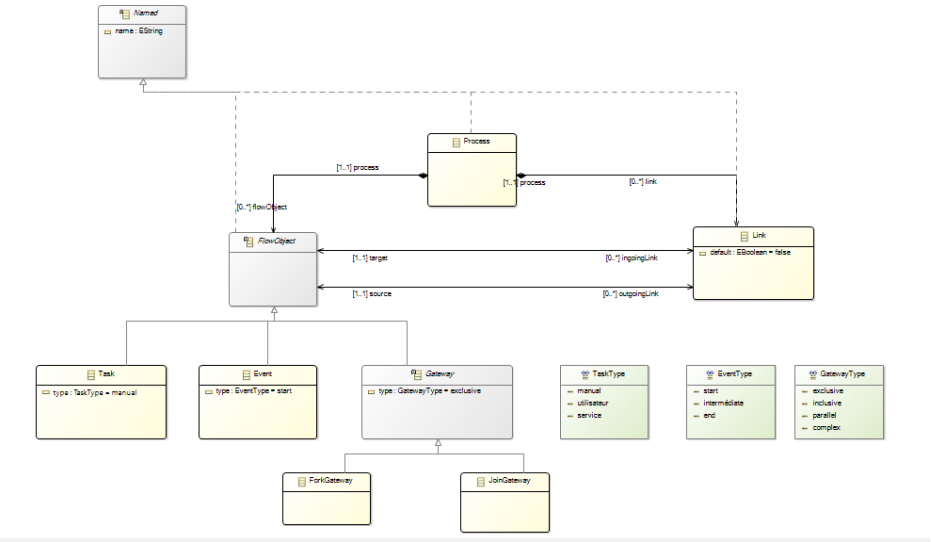


FIGURE 4.4 – Métaméta-modèle 1 Ecore (BPMN) réalisée sous l'environnement EMF

— **La classe process** :elle est considéré comme la classe mére qui représente un BPMN.

— **La classe Named** :elle porte l'attribut name qui indique que les taches , l'événements et les branchements ne peuvent pas avoir de noms vides, chaque type prend un nom.

— **la classe Task** : Elle représente une action. Chaque tâche à un début et une fin et donc une tâche ne peut débuter que si la tâche précédente est terminée. Une tâche à un type permettant de préciser son fonctionnement
. manuelle
. utilisateur
. service

— **La classe Évènement** : Les évènements servent à identifier un état particulier dans le processus, il existe 3 types d'évènements : Start-message-End.
Les évènements start etEnd doivent toujours être présents sur un processus BPMN. Ils sont le squelette du processus.
- S'il ne contient pas d'évent début, alors les objets de flux ne possédants pas

d'arcs entrant, constituent le début du processus et doivent démarrer en parallèle.

- S'il ne contient pas d'évent fin, alors les objets de flux ne possédants pas d'arcs sortant, ce dernier s'achèvera une fois tous ses sommets sont atteints.

— **La classe Gateway** : Le branchement est un objet essentiel dans la norme BPMN. Il sert à représenter la condition de routage entre le(s) flux en entrée et le(s) flux en sortie.

Les branchements sont autant utilisés pour diviser un flux en plusieurs flux que pour réunir plusieurs flux en un seul. Les différents branchements possibles dans BPMN sont : exclusifs-inclusives-parallèles-complex

— **La classe Link** : Une classe des inputs Arc ou des outputs Arc ; Elle possède un source et un cible .

Méta Modèle Réseaux de Petri

La figure 4.5 capture le méta-modèle EMF pour les réseaux de petri .

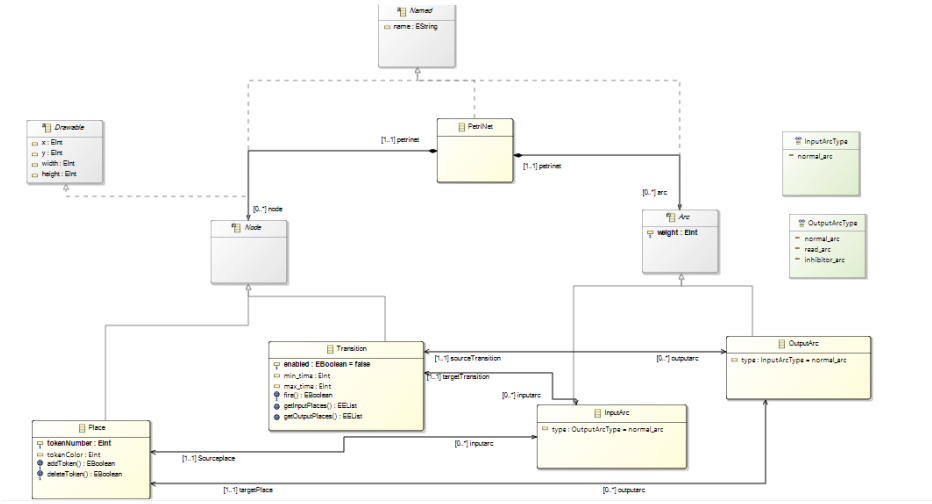


FIGURE 4.5 – Métaméta-modèle 2 Ecore (RDP) réalisée sous l'environnement EMF

— **Classe PetriNet** : elle est considérée comme la classe mère qui représente le réseau de Petri.

— **Classe Place** : elle indique une place qui est visualisée à l'aide d'un cercle. Elle possède deux attributs color token, tokenNumber qui représente le nombre de jeton de la place, cet attribut est visualisé avec un nombre entier dans le cercle représentant la place et deux opérations, l'ajout et suppression d'un jeton.

— **Classe Transition** : elle est visualisée à l'aide d'un rectangle. Une instance de la Classe Transition peut être relié à une instance de la classe Place par un arc d'entré ou bien de sortie.

— **Classe Arc** : Une classe désigne des inputs Arc ou des outputs Arc ; Elle possède un attribut weight qui représente le poids de l'arc.

— **Classe InputArc** : elle relie la classe Place avec la classe Transition, elle est visualisée par une flèche portant l'attribut Type.

— **classe OutputArc** : elle relie la classe Transition avec la classe Place, elle est visualisée par une flèche portant l'attribut Type.

4.5.3 Prise en main de Ecore

Pour créer un méta-modèle utilisant EMF il faut suivre les étapes suivantes :

1. Lancer Eclipse EMF,

2. Aller à file ! new ! Project, la fenêtre suivante 4.6 :

3. Créer un projet Eclipse Modeling Framework vide (Empty EMF Project) et précise son nom et l'emplacement d'enregistrement.

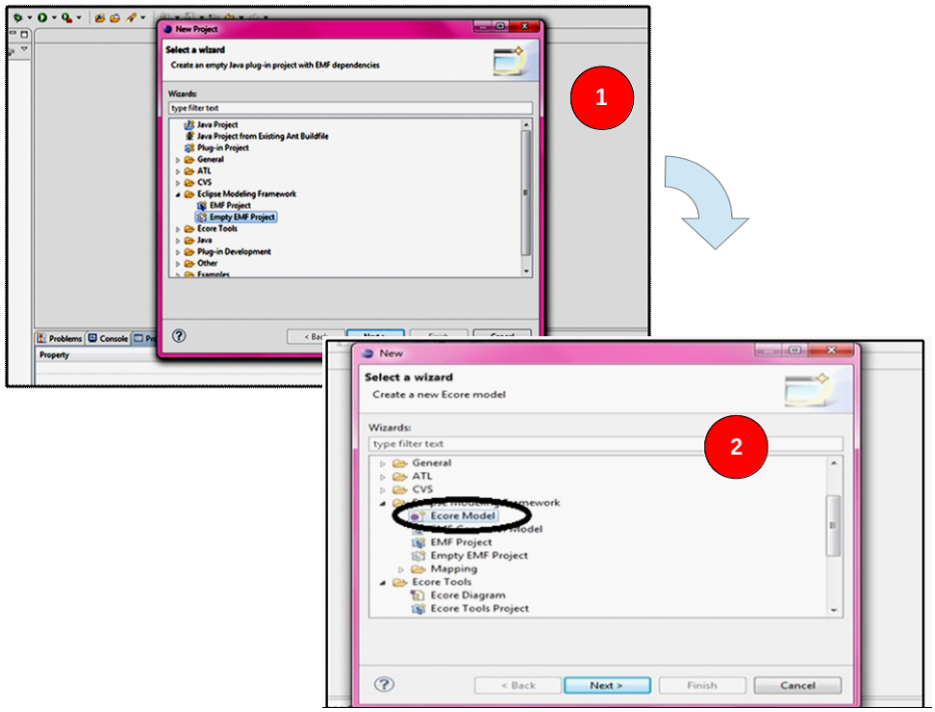


FIGURE 4.6 – Premières étapes pour créer un méta-modèle avec Ecore

4. Pour créer un méta-modèle Ecore, sélectionner le projet que vous avez créé précédemment. Et sélectionner en suite : new ! other,

5. Lorsque vous cliquez sur Other une fenêtre s'affiche. Elle sert à sélectionner le type de fichier à créer : choisir Eclipse Modeling Framework, puis Ecore model. La figure 4.7 montre le résultat de cette action.

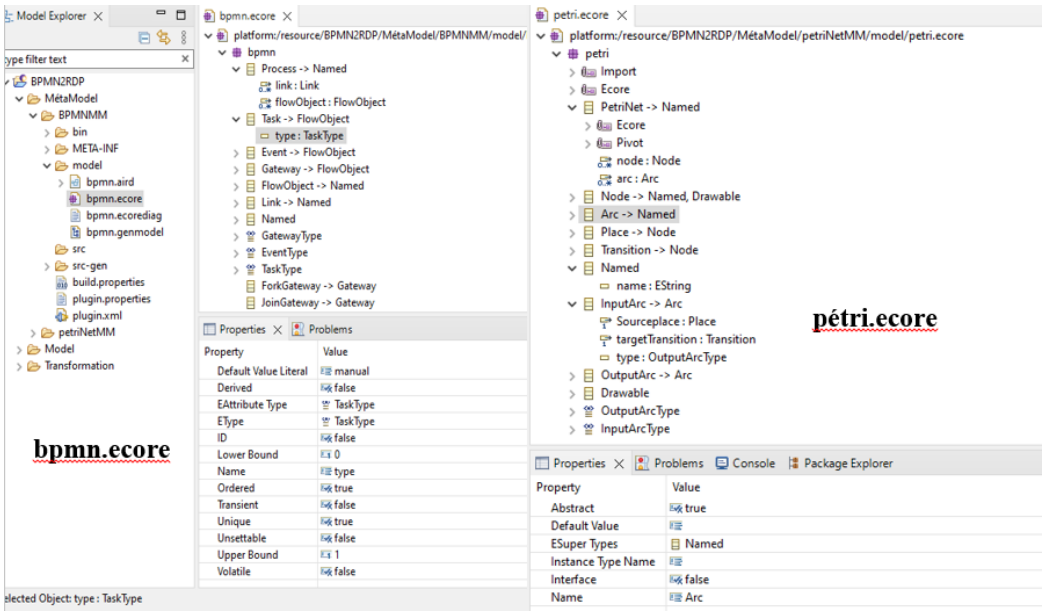


FIGURE 4.7 – capture d'écran des Méta-modèle Ecore BPMN et RdP

6. Pour créer les classes du méta-modèle, les attributs de ces classes ainsi que les références entre ces dernières, suivez les étapes suivantes :

(a) Sélectionner la racine du fichier Ecore (dans notre exemple My.ecore) puis cliquer avec bouton droit et choisir new childEclass

(b) De la même façon on peut créer autant de classe selon les besoins.

(c) Chaque classe est caractérisée par un ensemble d'attributs, on peut ajouter un attribut par la sélection de la classecliquer bouton droit new childEAttribute. Lorsqu'on ajoute un attribut, un ensemble de propriétés lui a été lui associé, aller sur la propriété Name et donner un nom à ce dernier ainsi que son type.

(d) Pour ajouter une référence entre les classes, commencer par le choix de la classe source$\rightarrow$ $boutondroit \rightarrow newchild \rightarrow EReference$.

Puis donner un nom à la référence, un type (le nom de la classe cible), la propriété Iscontainement reçoit la valeur True. Préciser les cardinalités (exemple de 1 à *). La Figure 4.8 résume ces étapes.

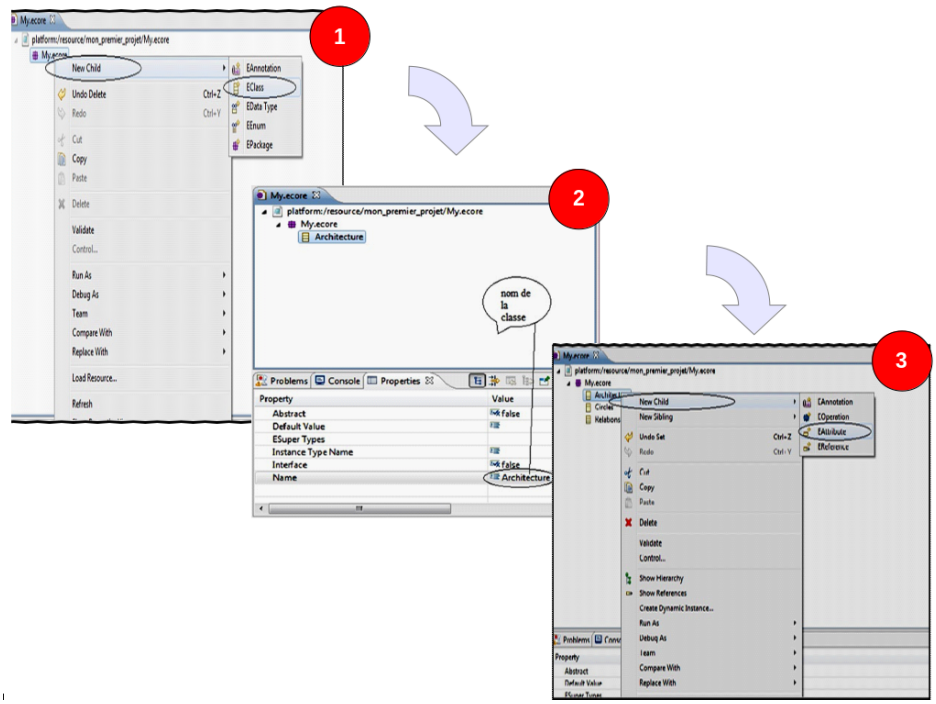


FIGURE 4.8 – Création de classe, Attributs et références dans Ecore

4.5.4 ATL (Atlas Transformation Language)

ATL est un langage de transformation de modèle. Il permet à la fois des constructions déclaratives et impératives. Toutefois, les auteurs d'ATL recommandent l'utilisation du style déclaratif qui simplifie l'écriture des transformations.

Les transformations dans ATL se composent de trois modules : Header, Helpers et Rules.

1. Header est utilisé pour déclarer des informations générales tels que les noms du module et de la transformation, des métamodèles source et cible et des bibliothèques utilisées dans la transformation.
2. Helpers sont des instructions génériques basées sur l'OCL pour éviter la redondance du code de transformation.
3. Rules sont des expressions qui décrivent la manière de générer les éléments du modèle cible (basé sur le métamodèle cible) à partir des éléments du modèle source (basé sur le métamodèle source). Il existe une implémentation d'ATL

dans l'IDE d'Eclipse dotée d'un moteur de règles responsable de la compilation et l'exécution des règles de transformation [9].

Mécanisme de transformation des modèles

Une transformation de modèles se réalise en général selon trois étapes :

1. Définition des règles de transformation :Cette étape consiste à mettre en correspondance les éléments du modèle source et les éléments du modèle cible.
2. Expression des règles de transformation :Il faut un langage de transformation (exemple QVT ou ATL) capable de spécifier, filtrer et sélectionner les éléments de chacun des modèles source et cible pour ensuite les écrire dans un format interprétable par un moteur d'exécution de règles de transformation.
3. Exécution des règles de transformation :Les règles de transformation, une fois spécifiées et exprimées, requièrent un moteur d'exécution. Celui-ci prend en entrée le modèle et son métamodèle source, les règles de transformation ainsi que le métamodèle du modèle cible. Il produit en sortie le modèle cible [8].

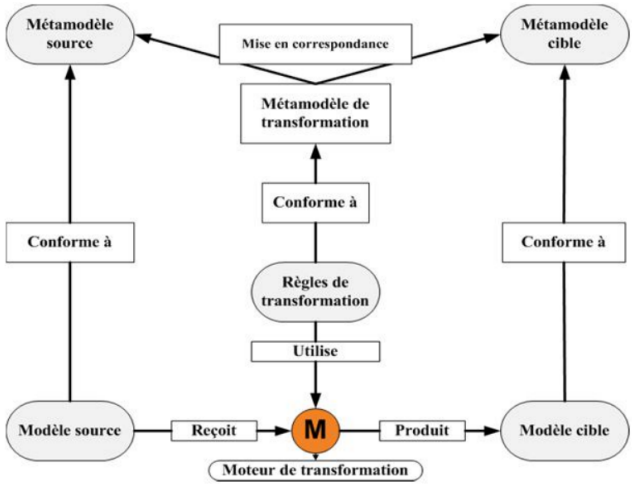


FIGURE 4.9 – Mécanisme de transformation des modèles[9]

4.5.5 Règles de la transformation des BPMN Vers RdP

La figure 4.10 résume les règles de transformation qu'on propose pour passer d'un modèle BPMN vers un réseau de petri .

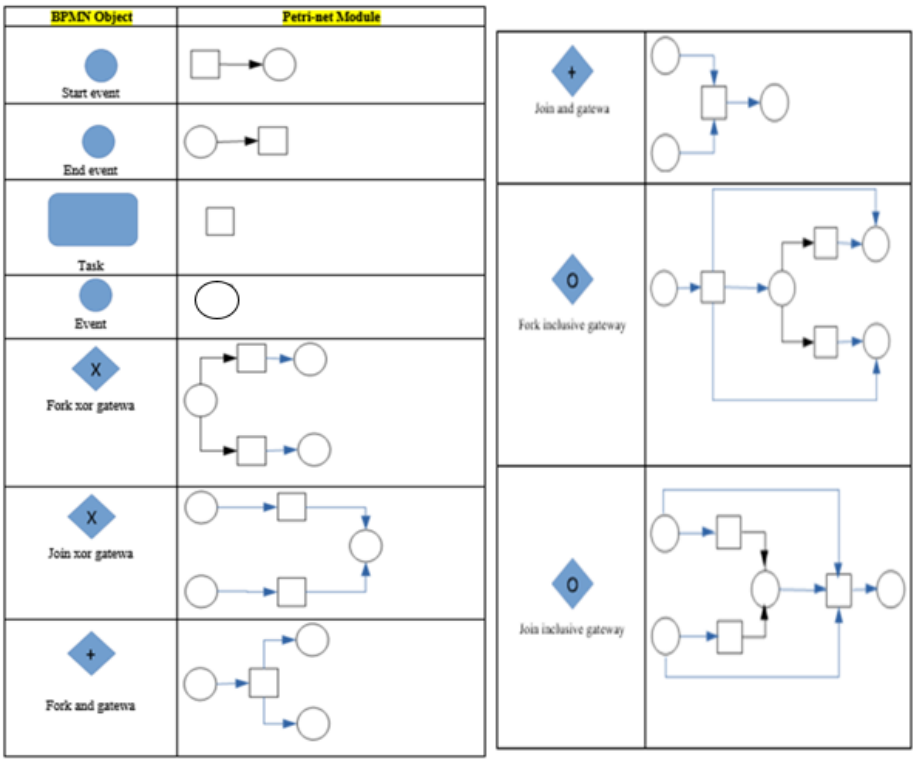


FIGURE 4.10 – Mapping BPMN vers Petri-net modules

Nous établissons les correspondance entre les modèles BPMN et les réseaux de Petri de la façon suivante :

1. **Tâche, événement et passerelles**

   (a) Une tâche(Task) est mappé sur une transition.

   La transition, étiquetée avec le nom de cette tâche ou de cet événement, modélise l'exécution de la tâche ou de l'événement.

   (b) Un événement de début (Start) est mappé sur une transition, arc input, place.

   (c) Un événement de fin (End) est mappé sur une place, arc output, transition.

(d) Un évènement de type (message) est mappé vers une place

2. **Les branchements de condition sont transformés en une structure en réseau de Pétri :**

   (a) Pour le branchement parallèle *Fork and Gateway* : il est représenté par une transition, sa place d'entrée, et un nombre de places de sortie selon le nombre d'arcs sortants de ce branchement.

   (b) Pour le branchement parallèle *Join and Gateway* : il est représenté par une transition, un nombre de places d'entrée selon le nombre d'arcs entrants, et une seule place de sortie.

   (c) Pour le branchement conditionnel *exclusif fork xOr Gatway* : il est représenté par un nombre de transitions selon le nombre de ses arcs sortants, une seule place d'entrée, et un nombre de places de sortie selon le nombre de transitions.

   (d) Pour le branchement conditionnel exclusif *Join xOr Gatway* : il est représenté par un nombre de transitions selon le nombre de ses arcs entrants, le même nombre de places d'entrée, et une seule place de sortie.

   (e) pour le branchement de type conditionnel *inclusif (or)Fork ou join*. Nous devons les remplacer par une combinaison de branchement de type *xo*r et *and* équivalentes comme montré dans la figure 4.11

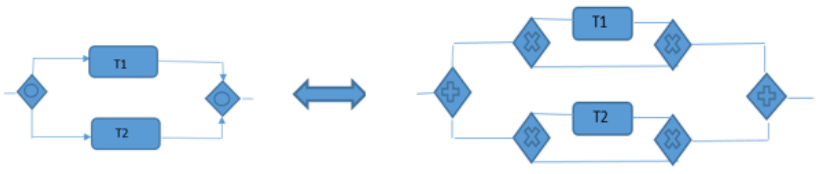


FIGURE 4.11 – Combinaison de branchement inclusive(OR)

Ceci veut dire qu'on est obligé d'enlever tous les branchements de type inclusive (or) et les remplacer par leurs équivalents avant de faire le mappage vers les réseau de petri.

3. **lien ou Link :** un lien est transforme en un Arc en réseau de petri. la nature Input ou output de l'arc est déterminé en fonction de la source et de lcible du lien.

4.5.6 Les contraintes OCL

Afin de vérifier la conformité des modèles, les contraintes OCL (Object Constraint Langage) sont considérées comme un moyen efficace. Cette conformité permet d'exprimer toutes les questions relatives à la préservation de la sémantique du modèle. Il faut noter, pour toute transformation ; c'est au concepteur des règles de transformation d'imposer n'importe quelle condition qui doit être respectée. Pratiquement, le langage OCL est disponible dans la plateforme EMF/Eclipse à travers l'outil OclInEcore.

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore';

package petri : petri = 'http://www.univ-guelma.dz/petri'
{
	class PetriNet extends Named
	{
		property node#petrinet : Node[*|1] { ordered composes };
		property arc#petrinet : Arc[*|1] { ordered composes };
		invariant uniquePlacesNames:  self.node->selectByType(Place)->
forAll(e1,e2|(e1<>e2)implies (e1.name<>e2.name));
		invariant
		uniqueTransitionNames:  self.node->selectByType(Transition)-
>forAll(e1,e2|(e1<>e2)implies (e1.name<>e2.name));
	}
	abstract class Node extends Named,Drawable
	{
		property petrinet#node : PetriNet[1];
	}
	abstract class Arc extends Named
	{
		property petrinet#arc : PetriNet[1];
		attribute weight : ecore::EInt[1];
		invariant positiveWeight: self.weight >= 1;
	}
	class Place extends Node
	{
		operation addToken() : Boolean[1];
		operation deleteToken() : Boolean[1];
		attribute tokenNumber : ecore::EInt[1];
		property inputarc#Sourceplace : InputArc[*|1] { ordered };
		property outputarc#targetPlace : OutputArc[*|1] { ordered };
		attribute tokenColor : ecore::EInt[1];
		invariant positiveTokenNumber: self.tokenNumber >= 0;
	}
```

FIGURE 4.12 – les contraintes OCL

— Classe Arc : Elle possède une contrainte OCL : WeightError : self.weight > 1, cela indique que les arcs doivent toujours avoir un nombre positif de weight.

— Classe Node : Elle possède une contrainte OCL : EmptyNameError : self.name.size() > 0,cela indique que les places et les transitions ne peuvent pas avoir de noms vides.

— Classe Place : Elle possède également une contrainteOCL : tokenNumberError : self.tokenNumber> 0, cela indique que les places ne peuvent pas avoir un nombre négatif de jetons.

4.5.7 Syntaxe d'une règle ATL

L'opération élémentaire dans une transformation ATL est nommée « règle ATL ». Un programme de transformation écrit en ATL est composé de règles qui spécifient comment les éléments du modèle source sont reconnus et parcourus pour créer et initialiser les éléments du modèle cible. Ces règles sont de la forme générale décrite par la figure suivante

```
61 -- R7 ---Fork And Gateway rule--------------------------------
62
63 rule ForkParallelGateway2Place extends FlowObject2Node{
64     from
65         f:MM!ForkGateway (f.type=#parallel)
66     to
67         n:MM1!Place(name <-f.name),
68         p1:MM1!Place(name <-'output1',petrinet <- f.process),
69         p2:MM1!Place(name <-'output2',petrinet <- f.process),
70         t:MM1!Transition(name <- 't',petrinet <- f.process),
71         a1:MM1!InputArc (sourcePlace <- n,targetTransition <- t,petrinet <- f.process),
72         a2:MM1!OutputArc(sourceTransition <- t,targetPlace <- p1,petrinet <- f.process),
73         a3:MM1!OutputArc(sourceTransition <- t,targetPlace <- p2,petrinet <- f.process)
74 }
75
76
```

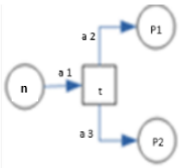


FIGURE 4.13 – syntaxe d'une règle de transformation en ATL

4.5.8 Exécution de la transformation avec un exemple réél (Processus de Recrutement)

Nous avons vu précédemment la méthode pour créer un méta-modèle avec le framework EMF , on va maintenant voir comment créer un modèle à partir d'un méta-modèle. Pour créer un modèle (instance de méta-modèle) il faut suivre les étapes suivantes :

- Aller sur le projet qui contient le méta-modèle a instancié (dans cet exemple "BPMN2RDP").

- Ouvrir le méta-modèle se format Ecore (le méta-modèle source "process").

- Cliquer droit sur la racine du méta-modèle puis choisir l'option "Create Dynamic Instance" (dans cet exemple la racine est nommée process). En suite, donner un nom à votre modèle, préciser et valider le chemin pour enregistrer ce modèle (enregistre le dans le dossier "Models").

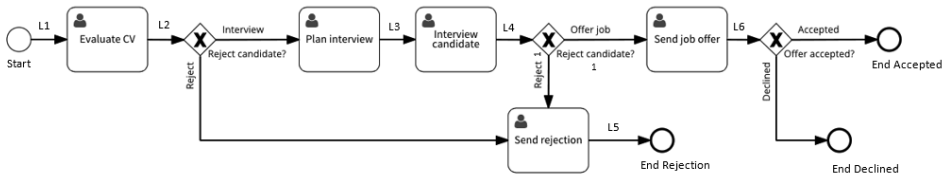


FIGURE 4.14 – Exemple Processus de Recrutement

• Pour exécuter la transformation, il faut ouvrir le fichier ATL qui contient les règles de transformation : aller dans la barre d'outil ! run ! une fenêtre suivante s'affiche qui permet de configurer la compilation du fichier ATL 4.16

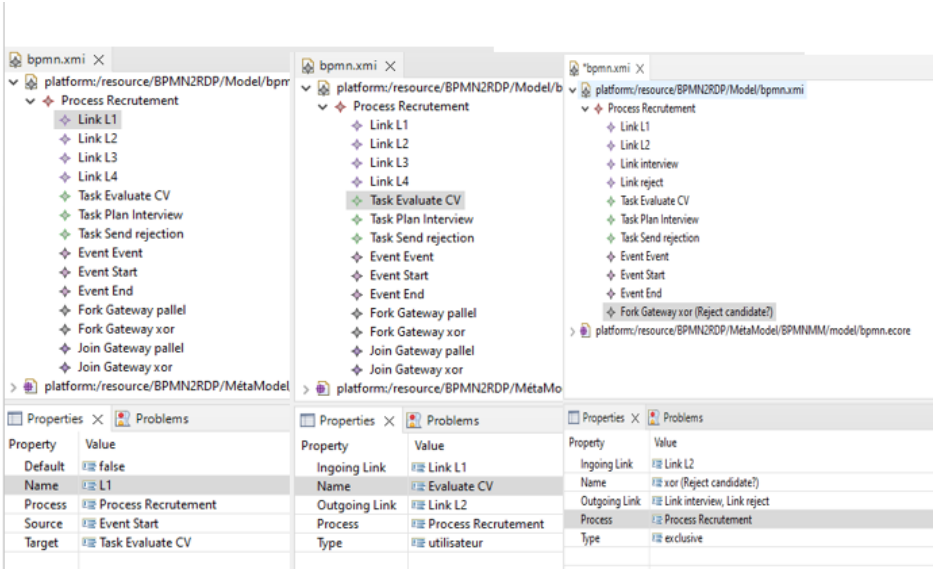


FIGURE 4.15 – Instanciation de modèle (Processus de Recrutement)

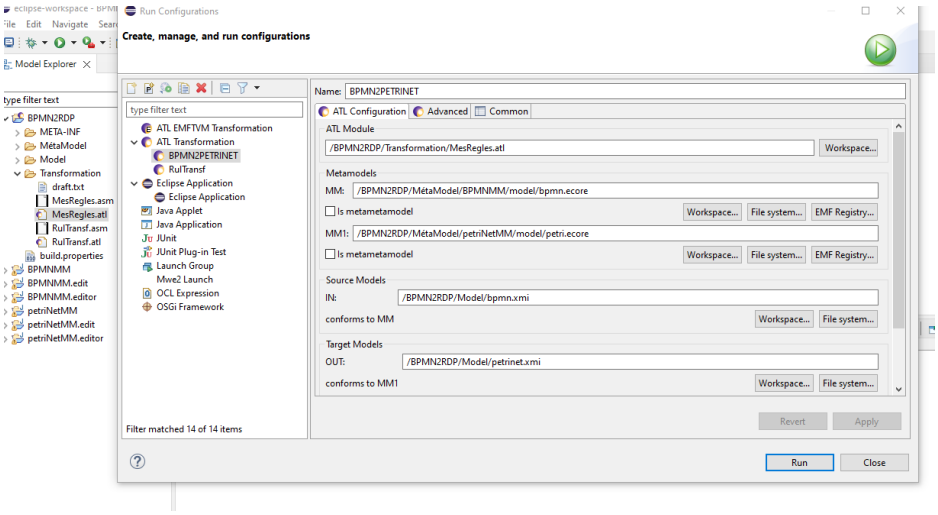


FIGURE 4.16 – Configuration de Run pour ATL

• Après avoir exécuter notre moteur de transformation on a eu un réseau de Pétri . La figure 4.17 donne le modèle BPMN d'input après préparation.

• Nous remarquons maintenant que le résultat obtenu est conforme à méta-modèle des RDP et respect ses contraintes.

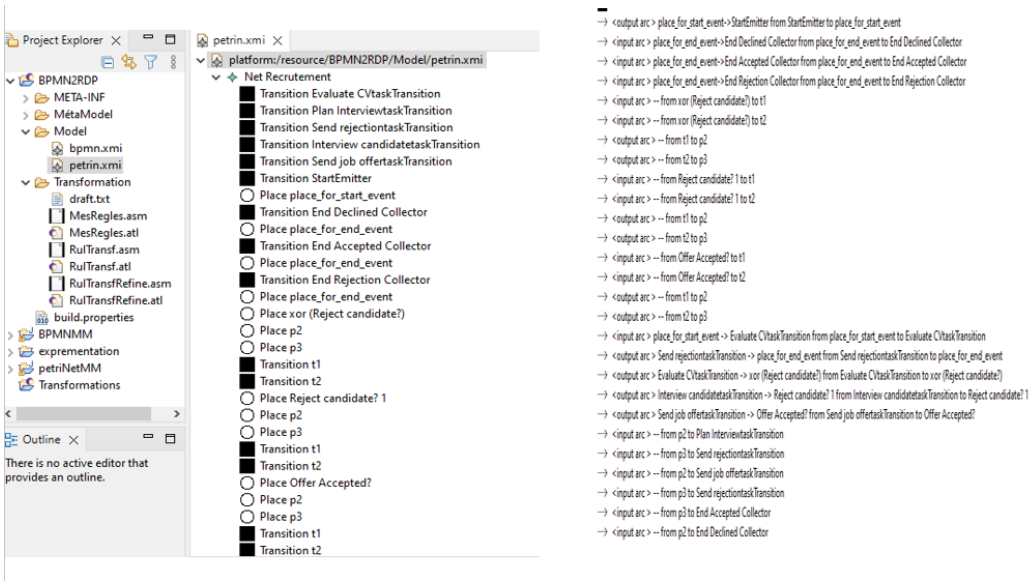


FIGURE 4.17 – PétriNet.xmi

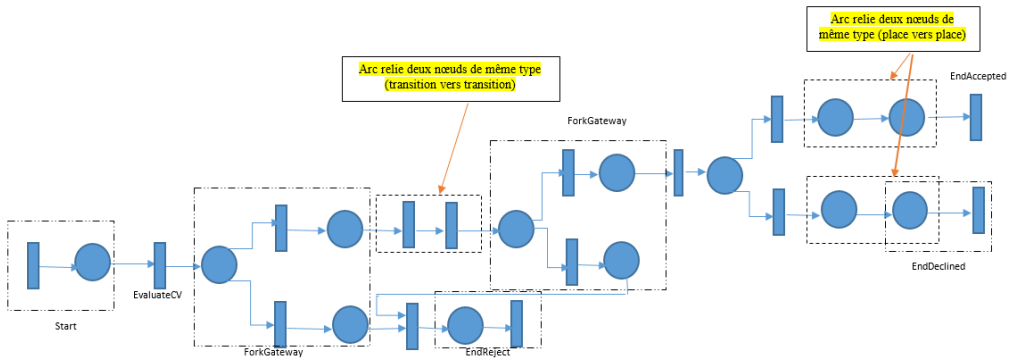


FIGURE 4.18 – le Modèle de sortie

Discussion du résultat

Nous constatons que le réseau de pétri généré contient des incohérences sémantiques

1. l'existence d'un arc qui relie deux nœuds de même type (place vers place)
2. l'existence d'un arc relie deux nœuds de même type (transition vers transition)

Pour lever ces incohérences. Nous devons faire une préparation du modèle BPMN avant de le transformer vers le réseau de pétri.la figure 4.19 illustre un extrait de ces règles de préparation du modèle BPMN.

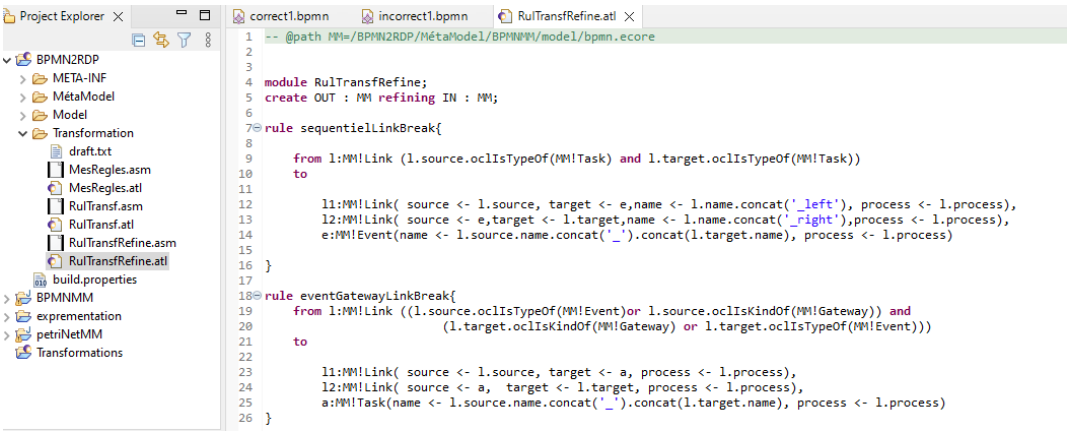


FIGURE 4.19 – Règles raffinement de BPMN

Cette préparation s'effectue à travers une transformation ATL endogène (une raffinement du modèle BPMN dans lui même) décrite comme suite :

1. Insérer un évènement de type message (avec le nom fictif) entre chaque deux taches consécutives.
2. Insérer une tache fictif (avec le nom tache fictif) entre chaque deux évènements/ branchement vers évènement/évènement vers branchement/branchement vers branchement consécutives

Ces transformations endogènes (raffinements) n'apportent aucune modification sur la sémantique du modèles BPMN. La figure 4.20 donne une exemple de résultat incorrect d'évènement(e1 vers e2) et résultat correct après raffinements de BPMN.

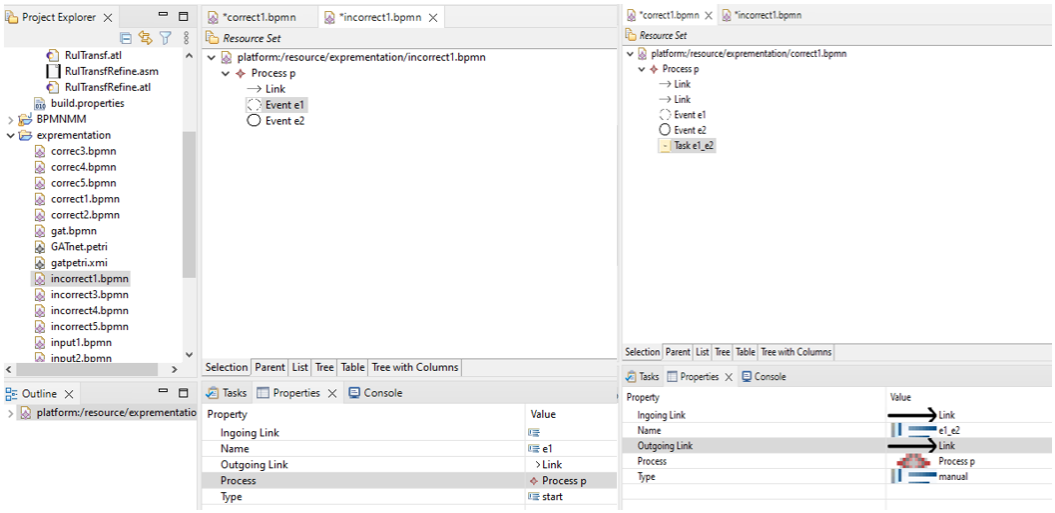


FIGURE 4.20 – Exemple de résultat après raffinement de BPMN

4.6 Conclusion

Nous avons parcouru durant ce chapitre la mise en ouvre de notre transformation en suivant l'approche de l'ingénierie dirigée par les modèles. A prés avoir présenté les deux méta modèles sources et cible et la manière de leurs élaboration sous la plate-formes EMF. Nous avons exposé la mise en oeuvre de la transformation en utilisant le puissant outil ATL avec son langage déclaratif. nos résultats sont sont confirmé après l'expérimentation de notre transformation sur un exemple réel.

CONCLUSION GÉNÉRALE

Nous arrivons à terme dans ce rapport. Nous avons voulu à travers ce modeste travail de proposer une contribution qui aide les concepteurs et les analystes des processus métier à détenir entre leurs mains un outil automatique qui leurs permet de passer automatiquement d'un modèle informel d'un processus métier vers un modèle formel équivalent en réseau de petri sur lequel ils peuvent effectuer des analyses. Pour bien mener notre travail. nous avons appuyé sur l'approche MDE qui englobe aujourd'hui une large communauté. l'approche MDE préconise essentiellement l'utilisation massives des modèles et des transformation des modèles. Par conséquent, nous avons travailler à fournir des environnement pour la modélisation en BPMN et en réseau de petri avant de commencer à penser aux règles de correspondances entre les concepts des ces deux formalismes afin de réaliser les transformations. Évidement. la mise en oeuvre de nos idée nécessite des outils. A ce propos, nous avons pensé à des outils existants qui offrent un degré important de maturité et de fiabilité et nous avons trouvé dans la plateforme EMF pour la modélisation et la méta modélisation et ATL pour la transformation des modèles la meilleure combinaison.

Il reste beaucoup d'étapes avant de le qualifier ce travail d'un produit final ou livrable. Nous comptons de continuer à l'améliorer dans plusieurs axes notamment dans la puissance d'expressivité des méta modèles ainsi que dans la consistance des règles de transformation et la prise en charge de toutes les situations possibles.

BIBLIOGRAPHIE

[1] Farida ALI GUECHI. « Une approche de test des systemes multiagents basee sur des modeles formwls ». Thèse de doct. Université de Batna 2, 2014.

[2] Lamia Gaouar BEKKARA. *Décrire son interface avec le langage HCIDL : Approche MDA pour la construction d'interfaces homme-machine : cas de la multimodalité et de la plasticité*. Éditions universitaires européennes, 2020.

[3] Jean BÉZIVIN et Xavier BLANC. « MDA : Vers un important changement de paradigme en génie logiciel ». In : *Développeur référence v2* 16 (2002), p. 15.

[4] Mouna BOUARIOUA. « Une approche basée transformation de graphes pour la génération de modèles de réseaux de Petri analysables à partir de diagrammes UML ». In : *UNIVERSITÉ CONSTANTINE* 2 (2013).

[5] GW BRAMS, Charles ANDRÉ et Gérard BERTHELOT. *Réseaux de Petri : théorie et pratique*. Masson, 1983.

[6] Beno
it COMBEMALE. « Ingénierie Dirigée par les Modèles (IDM)–État de l'art ». In : (2008).

[7] Remco M DIJKMAN, Marlon DUMAS et Chun OUYANG. « Formal semantics and analysis of BPMN process models using Petri nets ». In : *Queensland University of Technology, Tech. Rep* (2007), p. 1-30.

[8] Olfa DJEBBI et Marie Pierre GERVAIS. « Mda : Vers l'industrialisation de la construction d'applications réparties ». In : *Mémoire de DEA* (2004).

[9] Nassima DJEMA. « Contribution à la maintenance des applications distribuées : un outil pour la restructration et la maintenance des bases de données ». Thèse de doct. Université Mouloud Mammeri, 2015.

[10] Mohamed Naoufel KHOLLADI. « Une approche de transformation de la notation BPMN vers BPEL basée sur la transformation de graphe ». In : (2009).

[11] Karim LABADI. « Contribution à la modélisation et à l'évaluation de performances des systèmes logistiques à l'aide d'un nouveau modèle de réseaux de Petri stochastiques ». Thèse de doct. Université de Technologie de Troyes, 2005.

[12] SPEM OMG et O NOTATION. « Software systems process engineering metamodel specification ». In : *OMG Std., Rev* 2 (2008), p. 18-71.

[13] Murata TADAO. « Petri nets : properties, analysis and applications ». In : *Proceedings of the IEEE* 77.4 (1990).

[14] Eric THIERRY et Christophe CRESPELLE. « Réseaux de Pétri et modeles proies-prédateurs ». In : ().

[15] WEB1. `https://www.modeliosoft.com/fr/technologies/mda.html`. (consulter le 17/03/2022).

[16] WEB10. : `https://www.iterop.com/comprendre-le-bpm/`. (consulter le 19/03/2022).

[17] WEB11. `https://asana.com/fr/resources/business-process-management-bpm`. (consulter le 19/03/2022).

[18] WEB13. `https://fr.myservername.com/top-10-best-business-process-management-software`. (consulter le 27/03/2022).

[19] WEB14. `https://www.edrawsoft.com/fr/what-is-bpmn.html`. (consulter le 19/03/2022).

[20] WEB15. https://www.microsoft.com/fr-fr/microsoft-365/business-insights-ideas/resources/the-guide-to-using-bpmn-in-your-business. (consulter le 27/02/2022).

[21] WEB18. https://www.Lucidchart.com. (consulter le 02/06/2022).

[22] WEB19. https://www.manager-go.com. (consulter le 04/06/2022).

[23] WEB2. https://www.omg.org/mda/. (consulter le 01/06/2022).

[24] WEB3. https://www.01net.com/article/245351.htm. (consulter le 01/06/2022).

[25] WEB5. https://www.ibm.com/docs/fr/SSQ2R2_9.5.1/org.eclipse.emf.doc/references/overview/EMF.html. (consulter le 17/03/2022).

[26] WEB6. https://www.eclipse.org/modeling/emf/. (consulter le 20/03/2022).

[27] WEB8. https://www.eclipse.org/atl/atlTransformations. (consulter le 15/04/2022).

[28] WEB9. :https://www.majyori.com/ooad/object-constraint-language. (consulter le 15/04/2022).