République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed El Bachir El Ibrahimi de Bordj Bou Arreridj
Faculté des Mathématiques et d'informatique

MEMOIRE

Présente en vue de l'obtention du diplôme
Master en informatique
Spécialité : Technologies de l'information et de la communication

THEME

La transformation automatique des diagrammes d'états-transitions vers les réseaux de Petri

Présenté par :
HADDOUCHE MEBARKA
DAHAMNA NOURA ELALDJA

Soutenu publiquement le :

Devant le jury composé :

Président :
Examinateur :
Encadreur :

Promotion : 2021/2022

Dédicace

Je dédie ce modeste travail et ma profonde gratitude :
À mes très chers parents qui m'ont apporté au quotidien un soutien et une confiance sans faille et de ce fait, Je ne peux pas exprimer ma gratitude seulement par des mots.
Que dieu vous protège et vous garde pour moi.
À mes chers frères et mes précieuses sœurs, les mots ne peuvent résumer ma reconnaissance et mon amour à votre égard.
À tous les membres de ma famille
À mes adorables amies, pour votre fidélité et votre soutien avec lesquels nous avons partagé nos moments de joie et de bonheur
À tout mes enseignants pour votre soutien, votre enseignement et vos conseils tout au long de notre parcoure éducatif et professionnel

Remerciement

Nous remercions Allah de nous avoir donné la santé, la volonté, la patience et la Persévérance pour terminer ce travail.
Avant de vous convier à la présentation de ce travail, l'opportunité nous est donnée de Témoigner notre gratitude et notre reconnaissance à toutes les personnes qui par leurs aides et
Leurs encouragements nous ont permis de réaliser ce mémoire.
J'exprime toute ma gratitude à l'Encadreur : **BENDIAF MESSAOUD**
Qui a su nous conseiller par ses critiques constructives et nous guider dans notre travail et toute la période de mémoire.
Nos remerciements s'adressent à tous les membres du jury pour l'honneur d'accepter de Juger notre travail.
Nos remerciements vont aussi à tous ceux et celles qui ont participé deprès ou de loin à L'élaboration du présent travail.

Enfin, je tiens à remercier toute la promotion 2021-2022 Master informatique. Ainsi que tous mes enseignants et les membres du département informatique d'Université Bordj Bou-Arreridj

RÉSUMÉ :

Les triples grammaires de graphes nous permettent de définir la relation entre différents types de modèles et de transformer un type de modèle en un autre, et La présentation graphique joue un rôle très important dans le développement logiciel qui nous ramène à l'intégration des graphes dans les transformations de modèles et la transformation de graphes. A cet effet, il est important de proposer des solutions pour intégrer les graphes dans le processus de transformation.

Notre approche est basée sur les méta-modèles, l'idée de base de notre travail est de créer un pont technologique entre l'environnement de l'IDM et l'environnement de grammaires de Graphes et de faire une transformation d'un diagramme d'état transition vers les réseaux de petri (RDP), en utilisant l'outil triple graph grammar (TGG). Pour cela il nous faudra définir un ensemble de règles de transformation qui vont permettre de réaliser automatiquement ce processus.

Mots-clés : transformationde modèles, Transformation de graphe, grammaires, méta-modèles, diagramme d'état transition
, [(RDP) les réseaux de petri], [(TGG) triple graph grammar]

ABSTRACT

Triple graph grammars allow us to define the relationship between different types of models and to transform one type of model into another, and Graphical presentation plays a very important role in software development which brings us back to the integration of graphs in model transformations and graph transformation. To this end, it is important to propose solutions to integrate graphs into the transformation process.

Our approach is based on meta-models, the basic idea of our work is to create a technological bridge between the environment of the IDM and the environment of grammars of Graphs and to make a transformation of a diagram of state transition to petri nets, using the TGG interpreter tool. To do this, we will need to define a set of transformation rules that will allow this process to be carried out automatically.

Keywords:, Model, model transformation, Graph transformation, grammars, meta-models, state transition diagram
[(RDP) petri nets], [(TGG) triple graph grammar]

الملخص :

تتيح لنا قواعد الرسوم البيانية الثلاثية تحديد العلاقة بين أنواع مختلفة من النماذج وتحويل نوع واحد من النماذج إلى آخر ، كما يلعب العرض الرسومي دورًا مهمًا للغاية في تطوير البرامج مما يعيدنا إلى تكامل الرسوم البيانية في تحويلات النموذج وتحويل الرسم البياني. تحقيقا لهذه الغاية ، من المهم اقتراح حلول لدمج الرسوم البيانية في عملية التحول.

يعتمد نهجنا على النماذج الوصفية ، والفكرة الأساسية لعملنا هي إنشاء جسر تكنولوجي بين بيئة IDM وبيئة القواعد النحوية للرسوم البيانية وإجراء تحول في رسم تخطيطي لانتقال الحالة إلى شبكات بتري (PDR) ، باستخدام أداة قواعد الرسم البياني الثلاثي (TGG). للقيام بذلك ، سنحتاج إلى تحديد مجموعة من قواعد التحويل التي ستسمح بتنفيذ هذه العملية تلقائيًا..

الكلمات الرئيسية: النموذج، تحويل النموذج، تحويل الرسم البياني ، القواعد النحوية ، النماذج الوصفية ، مخطط انتقال الحالة [قواعد الرسم البياني الثلاثي (TGG)]، [شبكات بتري (RDP)] ،

Table des matières

Table des figures

Liste des tableaux

Acronymes

UML : Unified **M**odeling **L**anguage**.**
RDP : Réseau **d**e **p**etri**.**
OPN: Oracle **P**artner **N**etwork
IDM: Ingénierie **D**irigée par les **M**odèles.
OMG : Object **M**anagement **G**roup
MOF : Meta-**O**bject **F**acility.
MDA : Model **D**riven **A**rchitecture.
OCL : Object **C**onstraint **L**anguage**.**
IBM : International **B**usiness **M**achines
XML: Extensible **M**arkup **L**anguage.
CIM: Computation **I**ndependent **M**odel.
PIM : Platform **I**ndependent **M**odel .
PDM: Platform **D**escription **M**odel
PSM: Platform **S**pecific **M**odel
M2T : Model to **T**ext.
M2M : Model to **M**odel.
LHS : left **H**and **S**ide
RHS : Right **H**and **S**ide
TGG : Triple **G**raph **G**rammar
VPM : Volume **P**laquettaire **M**oyen
EMF: Eclipse **M**odeling **F**ramework**.**
SysML: Système **M**odeling **L**anguage
GMF: Graphical **M**odeling **F**ramework
ATL: Atlas **T**ransforming **L**anguage
JDK : Java **D**éveloppement **K**it
XMI : XML **M**etadata **I**nterchange
JDT : Java **D**evelopment **T**ooling
J2SE : Java **2 S**tandard **E**dition
J2EE : Java **2 E**nterprise **E**dition
JME : Java **M**icro **E**dition
QVT : (**Q**uery/**V**iew/**T**ransformation)

Introduction générale

Contexte :

UML est un langage de modélisation graphique à base de pictogrammes conçu comme une méthode normalisée de visualisation dans les domaines du développement logiciel et en conception orientée objet. Aujourd'hui, les systèmes modélisés sont de plus en plus complexe et leur taille ne cesse d'augmenter. Le langage UML est constitué de diagrammes. Ces diagrammes sont tous réalisés à partir du besoin des utilisateurs. Le diagramme d'états-transitions est parmi les diagrammes, de la norme UML, à offrir une vision complète de l'ensemble des comportements de l'élément auquel il est attaché. Les diagrammes d'états-transitions (statecharts diagramme), concept utilisé par David Harel pour son extension de notation de machine d'état à plat comprend des états imbriqués et concurrents. Cette notation a servi de base à la notation de la machine UML. UML étant un langage à caractère plutôt visuel. En effet, les notations semi-formelles et visuelles d'ULM peuvent provoquer des inconsistances au niveau des modèles développés

Le diagramme états-transitions est un schéma utilisé en génie logiciel pour représenter des automates déterministes. Il fait partie du modèle UML et s'inspire principalement du formalisme des statecharts et rappelle les grafcets des automates

La transformation de modèles est une opération fondamentale dans l'ingénierie dirigée par les Modèles (IDM). Elle peut être manuelle ou automatisée, mais dans ce dernier cas elle nécessite de la part du développeur qui la conçoit la maîtrise des méta-modèles impliqués dans la transformation. Les modèles sont considérés comme des éléments de base. L'une des devises les plus populaires de l'IDM prend tout son sens « Model once, gênerait every where » (« Modéliser une fois, générer partout »). La transformation de modèles est un processus de conversion d'un ensemble de modèles d'une application donnée à d'autres modèles de la même application. Une transformation de modèles définit un ensemble de règles pour passer d'un modèle source conforme à un méta-modèle source à un modèle cible conforme à un métamodèle cible.

Problématique

La transformation de modèles est une opération fondamentale dans l'ingénierie dirigée par les modèles. Elle peut être manuelle ou automatisée. Si il est le dernier cas elle nécessite le développeur qui la conçoit la maitrise des méta-modales impliqués dans la transformation donc comment maitrises le passage des « diagrammes d'état transition » ver « réseau de petri » ?

Objectif

Notre objectif est de proposer une approche basée sur la transformation de graphes en utilisant le TGG (Triple Graph Grammars) pour maitriser le passage des modèles de diagrammes d'états-transitions UML vers les Réseaux de Petri. Du fait que notre approche est dirigée par les modèles, nous avons utilisé un environnement technique adapté à la modélisation, la métamodélisation et la transformation des modèles. Nous avons choisi l'espace technologique très répandu Eclipse Modeling Framework, qui utilise Ecore pour créer et manipuler les modèles

Contribution

Dans le cadre de notre Projet de fin d'étude, nous proposons une approche basée sur la transformation de graphes en utilisant le TGG (Triple Graph Grammar) pour maîtriser La transformation automatique des diagrammes d'états-transitions vers les réseaux de Petri.

Plan de mémoire

Á la suite de ce préambule, notre mémoire se divise principalement en quatre chapitres.
Le premier chapitre : nous présentons Les diagrammes d'états-transitions, Présentation de diagramme d'états transition, Les différents types, Les concepts avancés, Les états…..
Le deuxième chapitre : nous présentons l'IDM Ingénieries dirigée par les modèles parles sur les méta-modélisations et les Notions générales de l'IDM, les Architecture et les phrase de modélisation ….
Le troisième chapitre : nous présentons Les réseaux de Petri, Définition de réseau de pétri, La description d'un RdP algébriquement, Concepts de base, Réseaux de petri particuliers….
Le quatrième chapitre : nous présentons l'Implémentation et mise en œuvre nous y détaillerons la partie réalisation : les outils, logiciels et environnement de développement et des prises d'écran de l'application, là où nous entamerons la modélisation, nous introduisons la méthode utilisée et ses différentes étapes. (Notre choix se portera l'outil TGG, suivant une démarche du processus de transformation de graphes). Enfin nous allons terminer par une conclusion générale et perspective

.

Chapitre 1

Diagramme d'état-transition

1.1 Introduction

Un diagramme d'états-transitions rassemble et organise les états et les transitions d'un classeur donné. Et décrivent le comportement interne d'un objet à l'aide d'un automate à états finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets (de type signaux, invocations de méthode).

Ils spécifient habituellement le comportement d'une instance de classeur (classe ou composant), mais parfois aussi le comportement interne d'autres éléments tels que les cas d'utilisation, les sous-systèmes, les méthodes.

Le diagramme d'états-transitions est le seul diagramme, de la norme UML, à offrir une vision complète et non ambiguë de l'ensemble des comportements de l'élément auquel il est attaché. En effet, un diagramme d'interaction n'offre qu'une vue partielle correspondant à un scénario sans spécifier comment les différents scénarii interagissent entre eux.

1.2 Diagramme d'états-transitions

1.2.1 Présentation

Un diagramme d'états-transitions est un graphe qui représente un *automate à états finis*, c'est-à-dire une machine dont le comportement des sorties ne dépend pas seulement de l'état de ses entrées, mais aussi d'un historique des sollicitations passées, comme la figure (1.1).

Un diagramme d'états-transitions rassemble et organise les états et les transitions d'un classeur donné. Bien entendu, le modèle dynamique du système comprend plusieurs diagrammes d'états-transitions. Il est souhaitable de construire un diagramme d'états-transitions pour chaque classeur (qui, le plus souvent, est une classe) possédant un comportement dynamique important. Un diagramme d'états-transitions ne peut être associé qu'à un seul classeur. Tous les automates à états finis des diagrammes d'états-transitions d'un système s'exécutent concurremment et peuvent donc changer d'état de façon indépendante. [9]

Les diagrammes d'état transition (statecharts diagramme), concept utilisé par David Harel. Pour son extension de notation de machine d'état à plat comprend des états imbriqués et concurrents. Cette notation a servi de base à la notation de la machine UML.

Les diagrammes d'états-transitions UML représentent en réalité des automates à états finis, mathématiquement parlant, ils représentent des graphes orientés.

Un diagramme d'état transition commence généralement par un rond noir qui indique l'état initial et se termine par un rond cerclé indiquant l'état final. Toutefois, bien qu'ils aient des points de départ et des extrémités bien définis, les diagrammes d'états-transitions ne sont pas forcément le meilleur outil pour représenter la progression d'une série d'événements. Ils sont plutôt indiqués pour illustrer des types de comportements spécifiques, notamment les changements d'état [10]

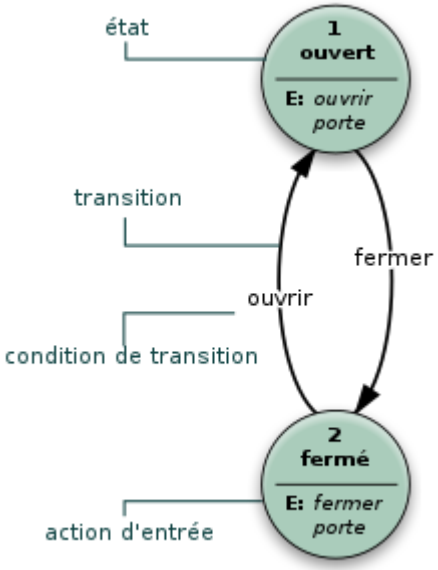


Figure 1.1 : Diagramme d'état transition

1.2.2 Symboles et composants des diagrammes d'états-transitions

Vous pouvez inclure de nombreuses formes différentes dans un diagramme états-transitions, surtout si vous décidez de l'associer à un autre diagramme. Voici la liste des formes les plus courantes que vous trouverez : [10]

- **État composite :**

État qui intègre des sous-états. Voir l'exemple de diagramme états-transitions d'une université ci-dessous. Dans cet exemple, « Inscriptions » représente l'état composite, car il englobe plusieurs sous-états dans le processus d'inscription [10]

- **Pseudo-état choix :**

Losange qui indique un état dynamique avec des résultats potentiels variables

- **Événement :**

Instance qui déclenche une transition. Son nom figure au-dessus de la flèche de transition applicable. Dans le cas présent, « fin des cours » est l'événement qui déclenche la fin de l'état « Enseigné actuellement » et le début de l'état « Examens finaux » [10]

Figure 1.2 : exemple de diagramme d'évènement

- **Point de sortie :**

Point auquel un objet quitte l'état composite ou l'automate, symbolisé par un cercle barré d'une croix. En règle générale, on l'utilise si le processus n'est pas terminé mais doit être quitté en raison d'une erreur ou d'un autre problème [10]

- **Premier état :**

Marqueur du premier état du processus, représenté par un cercle noir avec une flèche de transition [10]

- **Garde :**

Condition booléenne qui autorise ou bloque une transition, inscrite au-dessus de la flèche de transition [10]

- **État :**

Rectangle aux coins arrondis qui indique la nature actuelle d'un objet**.**

- **Sous-état :**

État contenu dans la zone d'un état composite. Dans le diagramme états-transitions de l'université ci-dessous, « Ouvert aux inscriptions » est un sous-état du plus grand état composite intitulé « Inscriptions ». [10]

- **Terminateur :**

Flèche allant d'un état à un autre et indiquant un changement d'état.

- **Transition :**

Flèche allant d'un état à un autre et indiquant un changement d'état. Comme la figure (1.3)

Figure1.**3 :** exemple de transition

- **Comportement de transition :**

Comportement résultant de la transition d'un état, inscrit au-dessus de la flèche de transition.

- **Déclencheur :**

Type de message qui déplace activement un objet d'un état à un autre, inscrit au-dessus de la flèche de transition. Dans cet exemple, « Problème avec la réservation » est l'élément déclencheur qui enverrait la personne à l'agence de voyage de l'aéroport au lieu de l'acheminer vers l'étape suivante du processus, comme la figure (1.4)

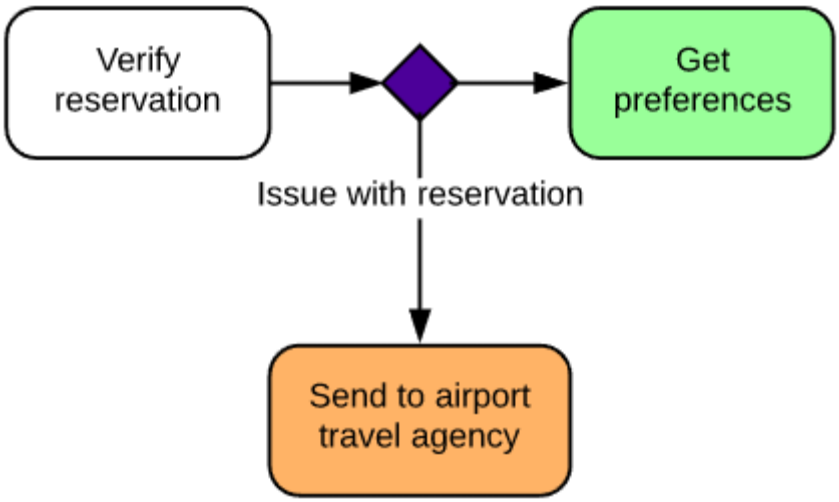


Figure1.4 : exemple pour Déclencheur

1.3 Les états :

• Un état correspond à la manière d'être d'un objet pendant un intervalle de temps.

• Un état se définit par : un nom, les actions d'E/S, les activités, les actions liées aux transitions internes (elles n'occasionnent aucun changement d'état).

• Un diagramme d'états a toujours un et un seul état initial pour un niveau hiérarchique donné. Il peut n'avoir aucun état final ou plusieurs. [11]

1.3.1 Les Types d'états :

- **L'état initial :**

Est un pseudo état qui indique l'état de départ, par défaut, lorsque le diagramme d'états-transitions, ou l'état enveloppant, est invoqué. Lorsqu'un objet est créé, il entre dans l'état initial État initial Initialisation du système, exécution du constructeur de l'objet [9]

- **L'état final :**

Est un pseudo état qui indique que le diagramme d'états-transitions, ou l'état enveloppant, est terminé. État final Fin de vie du système, destruction de l'objet [9]

- **États intermédiaires :**

Étapes de la vie du système, de l'objet [9]

1.4 Événement

Un événement est quelque chose qui se produit pendant l'exécution d'un système et qui mérite d'être modélisé. Les diagrammes d'états-transitions permettent justement de spécifier les réactions d'une partie du système à des événements discrets. Un événement se produit à un instant précis et est dépourvu de durée. Quand un événement est reçu, une

transition peut être déclenchée et faire basculer l'objet dans un nouvel état. On peut diviser Les événements en plusieurs types explicites et implicites : signal, appel, changement et temporel. Fait instantané venant de l'extérieur du système et survenant à un instant donné. [12]

1.4.1 Types d'événements:

1.4.1.1 Signal : Un signal est un type de classeur destiné explicitement à véhiculer une communication asynchrone à sens unique entre deux objets. L'objet expéditeur crée et initialise explicitement une instance de signal et l'envoi à un objet explicite ou à tout un groupe d'objets. Il n'attend pas que le destinataire traite le signal pour poursuivre son déroulement. La réception d'un signal est un événement pour l'objet destinataire. Un même objet peut être à la fois expéditeur et destinataire, réception d'un message asynchrone [13]

1.4.1.2 Appel d'une opération (synchrone) : liée aux cas d'utilisation, opération du diagramme de classes, Un événement de changement est généré par la satisfaction (i.e. passage de faux à vrai) d'une expression booléenne sur des valeurs d'attributs. Il s'agit d'une manière déclarative d'attendre qu'une condition soit satisfaite.
When (<condition booléenne>) évaluée continuellement jusqu'à ce qu'elle soit vraie [13]

1.4.1.3 Satisfaction d'une condition booléenne : Un événement de changement est généré par la satisfaction (*i.e.* passage de faux à vrai) d'une expression booléenne sur des valeurs d'attributs. Il s'agit d'une manière déclarative d'attendre qu'une condition soit satisfaite. La syntaxe d'un événement de changement est la suivante [13]

1.4.1.4 Temps - Date relative : Les événements temporels sont générés par le passage du temps. Ils sont spécifiés soit de manière absolue (date précise), soit de manière relative (temps écoulé). Par défaut, le temps commence à s'écouler dès l'entrée dans l'état courant. [13]

La syntaxe d'un événement temporel spécifié de manière relative est la suivante :

- Date relative When (date = date)

- Date absolue : after (durée)

1.5 Transition :

Une transition définit la réponse d'un objet à l'occurrence d'un événement. Elle lie, généralement, deux états *E*1 et *E*2 et indique qu'un objet dans un état *E*1 peut entrer dans l'état *E*2 et exécuter certaines activités, si un événement déclencheur se produit et que la condition de garde est vérifiée. La syntaxe d'une transition est la suivante : [9]

[< événement>] ['[' <Garde > '] '] ['/' <activité>]

1.5.1 Les différents types de transition :

ce tableau présente différents type de transition [13]

Type de transition	description	Syntaxe
Transition entry	Spécification d'une activité d'entrée qui s'exécute lorsqu'on saisit un état	Entry/activity
Transition exit	Spécification d'une activité de sortie qui s'exécute lorsqu'on quitte un état	Exit /activity
Transition externe	Une transition externe est une transition qui modifie l'état actif. Il s'agit du type de transition le plus répandu. Elle est représentée par une flèche allant de l'état source vers l'état cible	(a : T)[guard]/activity
Transition interne	Les transitions internes possèdent des noms d'événement prédéfinis correspondant à des déclencheurs particuliers : entry, exit, do et include. Ces mots-clefs réservés viennent prendre la place du nom de l'événement dans la syntaxe d'une transition interne.	e (a:T)[guard]/activity

Tableau 1.1 : Les différents types de transition

1.6 Les concepts avancés :

1.6.1 Point choix :

Il est possible de représenter des alternatives pour le franchissement d'une transition. On utilise pour cela des pseudos états particuliers : les points de jonction et les points de décision [14]

1.6.1.1 Point de jonction :

Les points de jonction sont un artefact graphique qui permet de partager des segments de transition, l'objectif étant d'aboutir à une notation plus compacte ou plus lisible des chemins alternatifs.

Un point de jonction peut avoir plusieurs segments de transition entrante et plusieurs segments de transition sortante. Par contre, il ne peut avoir d'activité interne ni des transitions sortantes dotées de déclencheurs d'événements. [14]

1.6.1.2 Point de décision :

Un point de décision possède une entrée et au moins deux sorties. Contrairement à un point de jonction, les gardes situées après le point de décision sont évaluées au moment où il est atteint. Cela permet de baser le choix sur des résultats obtenus en franchissant le segment avant le point de choix .Si, quand le point de décision est atteint, aucun segment en aval n'est franchissable, c'est que le modèle est mal formé**.** [14]

1.6.2 Etats composites :

- **Présentation :**

Un état simple ne possède pas de sous-structure, mais uniquement, le cas échéant, un jeu de transitions internes. Un état composite est un état décomposé en régions contenant chacune un ou plusieurs sous-états. [12]

- **États historiques :**

Un état historique, également qualifié d'état historique plat, est un pseudo état qui mémorise le dernier sous-état actif d'un état composite. Graphiquement, il est représenté par un cercle contenant un *H*.

Une transition ayant pour cible l'état historique est équivalente à une transition qui a pour cible le dernier état visité de l'état englobant. Un état historique peut avoir une transition sortante non étiquetée indiquant l'état à exécuter si la région n'a pas encore été visitée.

Il est également possible de définir un état historique profond représenté graphiquement par un cercle contenant un *H**. Cet état historique profond permet d'atteindre le dernier état visité dans la région, quel que soit son niveau d'imbrication, alors que le l'état historique plat limite l'accès aux états de son niveau d'imbrication. [12]

- **Les points de connexion :**

Les points de connexion sont des points d'entrée et de sortie portant un nom, et situés sur la frontière d'un état composite. Ils sont respectivement représentés par un cercle vide et un cercle barré d'une croix .Il ne s'agit que de références à un état défini dans l'état composite. Une unique transition d'achèvement, dépourvue de garde, relie le pseudo état source à l'état référencé. Cette transition d'achèvement n'est que le prolongement de la transition qui vise le point de connexion. Les points de connexions offrent ainsi une façon de représenter l'interface d'un état composite en masquant l'implémentation de son comportement. [12]

1.7 Conclusion :

Dans ce chapitre nous avons exposés le diagramme d'états-transition**,** Les diagrammes d'états-transitions permettent justement de spécifier les réactions d'une partie du système à des événements discrets. Un événement se produit à un instant précis et est dépourvu de durée. Quand un événement est reçu, une transition peut être déclenchée et faire basculer l'objet dans un nouvel état.

Chapitre 2
Réseaux de Petri

2.1 Introduction

Les réseaux de Petri est un modèle mathématique permettant la représentation de systèmes distribues discrets (informatique, industriel), introduit par Petri (1962). Est également un langage de modélisation, représente sous forme d'un graphe biparti oriente. Et Ils définissent une notation formelle pour la modélisation et la vérification de systèmes concurrents. La manipulation de tels modèles par des outils d'analyse permet une vérification automatique de propriétés structurelles, ainsi que la vérification des systèmes considérés suivant des formules de logique temporelle. Il est ainsi possible de vérifier l'absence d'un état interdit du système, la causalité entre états ou de caractériser les flux d'exécution dans l'architecture.

2.2 Définition de réseau de petri :

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 franchissabilité et de franchissement **[1].**

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

- 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.
- 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.
- Un arc : relie alternativement une place à une transition et une transition à une place

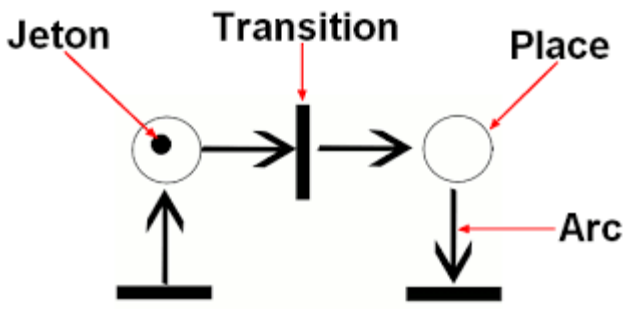


Figure 2.1 : Le réseau de Petri (RdP)

2.3 La description d'un RdP algébriquement :

Un RdP est un graphe composé de 2 types de nœuds :

- Les places (Pi) qui permettent de décrire les états du système modélisé. L'ensemble de ces places est noté P= {P1, P2, ...}.
- Les transitions (Ti) qui représentent les changements d'états. L'ensemble de ces transitions est noté T= {T1, T2, ...}. **[2]**

Un RdP est un quadruplet Q = tel que :
P = {Pi}, i ∈ {1,..., n} est appelé ensemble de places
T = {Tj}, j ∈ {1,..., m} est appelé ensemble de transitions avec P∩T = ∅
Pré est une application de P□T → ℕ dite d'incidence avant.
Post est une application de P□T → ℕ dite d'incidence arrière.
Pré (Pi, Tj) est appelé poids de l'arc reliant Pi et Tj.
Post (Pi ,Tj) est appelé poids de l'arc reliant Tj et Pi **[1]**

2.4 Marquage d'un Réseau de Petri :

Un marquage est dénoté par un vecteur du nombre de jetons dans chaque place : la i ieme composante correspond au nombre de jetons dans la i ieme ` place. **[5]**

M = (1, 0, 1, 0, 0, 2, 0)

Un réseau de Petri N = (P, T, F, W) est un réseau de Petri sans marquage initial et un réseau de Petri Rdp avec marquage initial M0 est Rdp = (N, M0). **[5]**

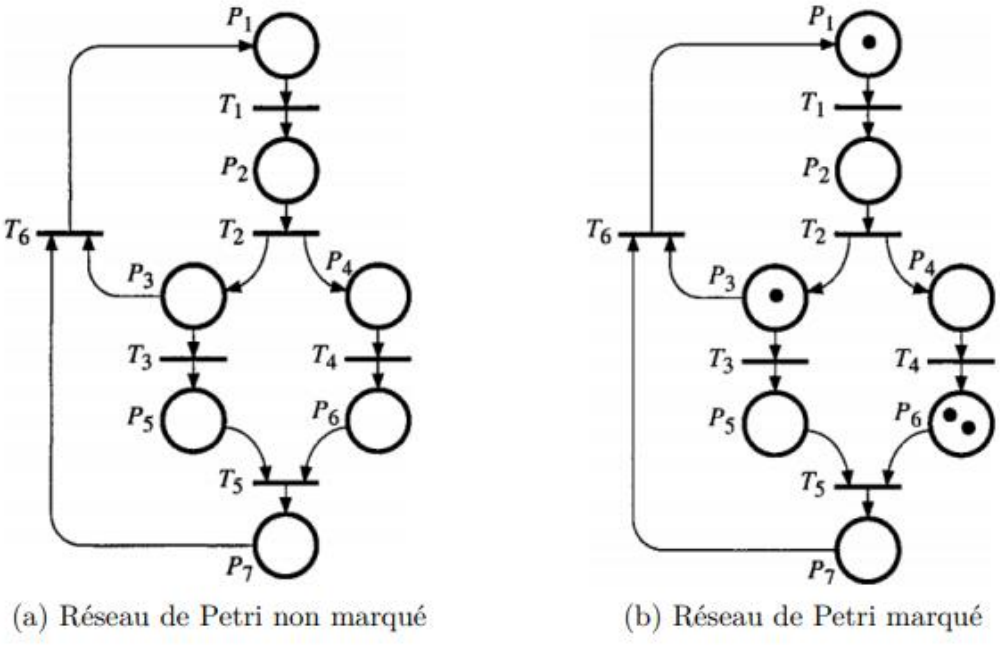


Figure 2 .2 : exemple de réseau de petri marqué et non marqué

2.5 Évolution d'un Réseau de Petri :

L'évolution d'un Réseau de Petri correspond à l'évolution de son marquage au fil du temps (évolution de l'état du système) : il se traduit par un déplacement des jetons pour une transition t de l'ensemble des places d'entrée vers l'ensemble des places de sortie de cette transitin. [5]

2.5.1 Transition validée :

On dit qu'une transition est validée si toutes les places en entrée de celle-ci possèdent au moins une marque. Une transition source est par définition toujours validée. [5]

2.5.2 Règle de Franchissement :

Si la transition est validée, on peut effectuer le franchissement de cette transition : on dit alors que la transition est franchissable [5]

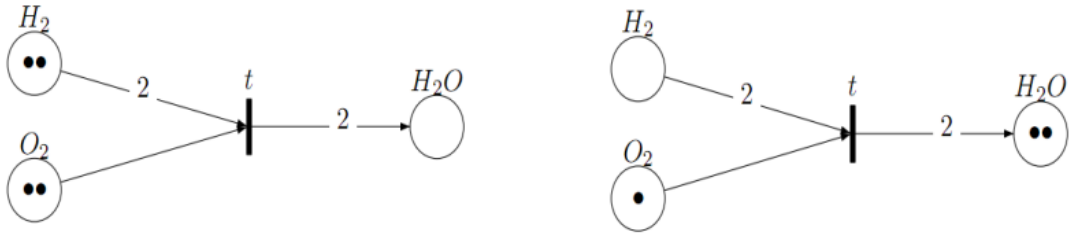
Le franchissement consiste à :

- retirer W (p, t) jetons dans chacune des places en entrée p de la transition t.
- ajouter W (t, p) jetons à chacune des places en sortie p de la transition t.

La règle de franchissement est illustrée par la figure 2.3 en utilisant la réaction chimique connue:

$$2H2 + O2 \rightarrow 2H2O$$

La présence des deux jetons dans chaque place d'entrée (figure 2.3a), indique que 2 unités de H2 et 2 unités d'O2 sont disponibles, donc la transition t est franchissable. Après avoir franchi t, le marquage va changer et on obtient le réseau ayant le marquage comme celui de la figure 2.3b, maintenant t n'est plus franchissable.



(a) Marquage avant de franchir la transition t.

(b) Marquage après avoir franchi t

Figure 2.3 : exemple de règle de franchissement de transition

2.6 Réseaux de petri particuliers :

Le graphe associé à un réseau de Petri peut être très complexe. Un certain nombre de situations présente un intérêt particulier : **[3]**

2.6.1 Les graphes d'états:

Dans ce cas chaque transition ne dispose que d'une place en entrée et une place en sortie.

2.6.2 Les réseaux sans conflits :

Dans lesquels chaque place n'a qu'une transition en sortie

2.6.3 Les réseaux dits simples :

Réseaux avec conflits mais dans lequel chaque transition n'intervient au plus que dans une situation de conflit.

2.6.4 Les réseaux purs

Dans cette situation aucune place n'est à la fois en entrée et en sortie de la même transition

La figure suivante illustre les définitions précédentes : **[3]**

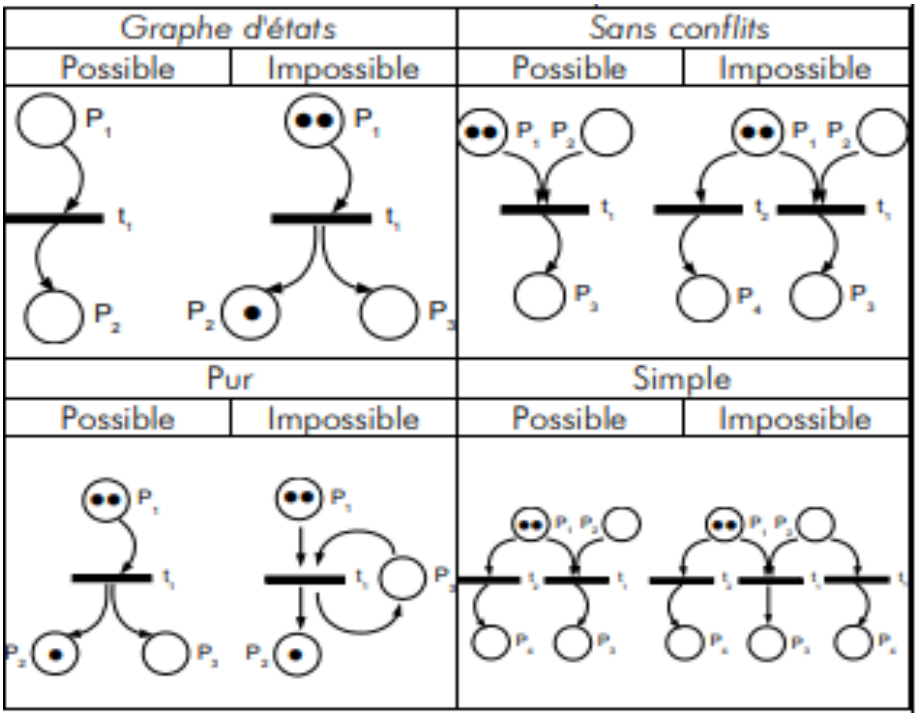


Figure 2.4 : les définitions précédentes

2.7 Propriétés comportementales des RdP

L'analyse d'une fonction ou d'un système séquentiel passe par l'étude des propriétés du RdP qui le représente. Parmi ces propriétés, nous citerons celles qui permettent d'affirmer que les spécifications incluses dans le modèle RdP sont correctes. C'est ainsi que nous pourrons démontrer que le nombre d'états pouvant être atteints est fini (RdP borné) ou qu'un réseau (donc le système représenté) est sans blocage (RdP pseudo-vivant, RdP vivant, RdP propre). On mettra également en évidence les conflits entre plusieurs évolutions possibles (et qui sont autant d'ambiguïtés à lever) (Conflits structurel et effectif), ou encore la gestion du partage des ressources (Exclusion mutuelle) **[7]**

2.7.1 RdP borné et non borné

Un RdP est borné pour un marquage initial donné si, quel que soit le marquage accessible atteint M et quelle que soit la place p considérée, le nombre de jetons contenus dans cette place est inférieur à une borne k : ∀M et ∀p : M (p) ≤ k (2.3) On dira également que le nombre d'états accessibles à partir de l'état initial est fini, le graphe d'états équivalent peut donc être construit. Interprétation : un système physique présente toujours un nombre d'états fini ; il en est ainsi, par exemple, d'un stock dont la capacité est toujours limitée. Toute spécification d'un système réel par RdP doit présenter un graphe borné. Lorsque la borne est égale à 1, on dit que le RdP est sauf ou binaire. **[8]**

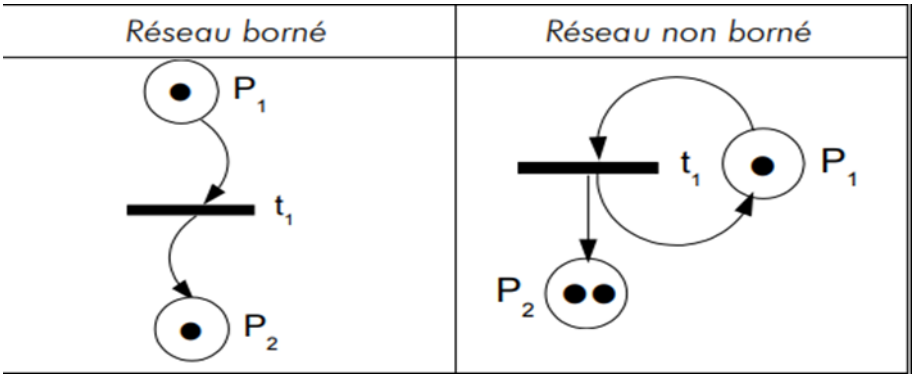


Figure 2.5 : Réseau borné et Réseau non borné

2.7.2 RdP vivant

Un RdP est vivant pour un marquage initial donné si, pour tout marquage M accessible à partir du marquage initial, il existe une transition t, il existe une séquence S de franchissements qui inclut la transition t : ∀M et ∀p ∈ P, ∃S : M M' tel que t ⊂ S (2.4) Interprétation : la vivacité indique que le système représenté est sans blocage, mais également qu'il n'existe pas de branche morte dans le modèle graphique, donc de spécification incomplète. **[8]**

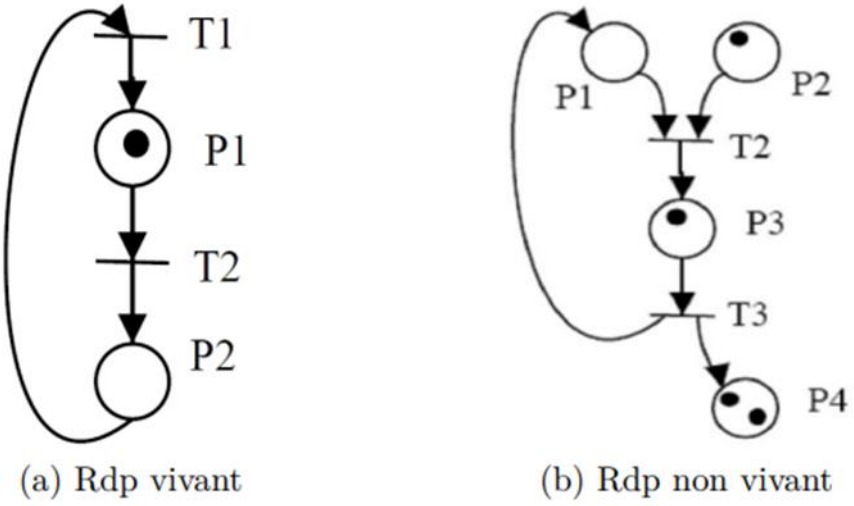


Figure 2.6 : Exemple de vivacité des Réseau de Petri

2.7.3 RdP pseudo-vivant

Un RdP est pseudo-vivant pour un marquage initial donné si, pour tout marquage M accessible à partir du marquage initial, il existe une transition t sensibilisée : ∀M et ∀p ∈ P, ∃t ∈ T : M (p) ≥ I (p, t) (2.5) Interprétation : cette propriété traduit l'absence de blocage total dans le système spécifié. 2 Spécification et vérification des Systèmes Embarqués Temps Réel 27. [8]

2.7.4 RdP réinitialisable

Un RdP est réinitialisable si, pour tout marquage M accessible à partir du marquage initial, il existe une séquence S de franchissements qui ramène au marquage initial : ∀M, ∃S : M M0 Interprétation : la plupart des processus industriels ont un fonctionnement répétitif. Il est donc très important de vérifier si les RdP qui les représentent sont réinitialisables. [8]

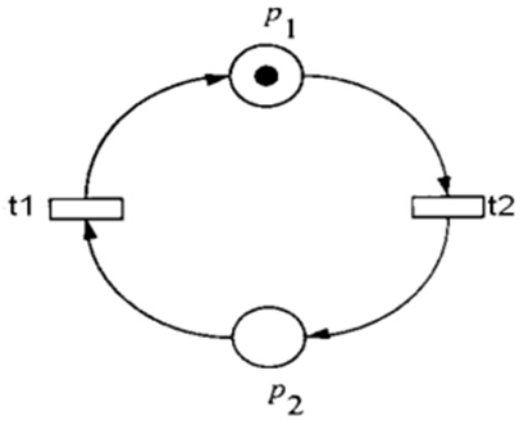


Figure 2.7 : Exemple d'un réseau de Petri réinitialisable

2.7.5 Conflits structurel et effectif

Deux transitions sont en conflit structurel lorsqu'elles possèdent une place d'entrée commune. Le conflit devient effectif si le marquage de la place commune sensibilise les deux transitions. Dans ce cas, le franchissement d'une transition empêche le franchissement de l'autre. Une seule transition sera franchie, mais rien dans le réseau ne permet de prévoir laquelle. Interprétation : un conflit effectif signifie qu'il y a non-déterminisme du réseau, donc que l'évolution du système décrit présente une partie aléatoire. [8]

2.7.6 Exclusion mutuelle

Deux places sont en exclusion mutuelle ou mutuellement exclusives si pour un marquage initial M0 donné, elles ne peuvent être simultanément marquées quel que soit le marquage M atteint à partir de M0. Interprétation : on rencontre l'exclusion mutuelle dans tout système comprenant un partage de ressource. [8]

2.8 Différents types de réseaux de Petri : [4]

Réseau **temporises**	• Les transitions instantanées n'ont pas de sens physique… • De même, deux évènements ne peuvent être simultanés • La notion de temps est alors introduite • Souvent utilisé pour la robotique industrielle
Réseau **interprétés**	• A chaque état on associe une loi de commande qui peut dépendre de l'environnement
Réseau **stochastiques**	• Font apparaître des processus non déterministes
Réseau **continus**	• Les transitions peuvent consommer et générer des fractions de jetons

Tableau 2.1 : Différents types de réseaux de Petri

2.9 Les Réseaux de Petri de Haut Niveau

Pour l'utilisation des réseaux de Petri dans la modélisation des systèmes réels, plusieurs auteurs ont trouvé qu'il est convenable d'étendre le formalisme de réseau de Petri pour compacter la représentation de modèle ou pour étendre le pouvoir de modélisation du formalisme de réseau de Petri. Ce qui a donné naissance aux réseaux de Petri de haut niveau. **[7]**

2.9.1 Réseaux de Petri colorés

Les réseaux de Petri colorés (CP-nets3 ou CPN) ajoutent aux réseaux de Petri, des primitives pour la définition de types de données et la manipulation de valeurs de données. Nous ne les utilisons pas dans la représentation interne du modèle formel mais uniquement pour représenter certains des réseaux de Petri génériques que nous produisons par

Transformation, de manière plus compacte. Dans un réseau de Petri coloré, un jeton a une valeur appartenant à un type.

• Un type de donnée est associé à une place indiquant qu'elle ne peut contenir que des jetons appartenant à ce type. Un type peut être le produit cartésien de types existants. Une place peut contenir un multiset de jetons (i.e. contenir des jetons de même valeur).

• Un arc entrant d'une transition peut avoir des gardes sur la sensibilisation de la transition. Une transition n'est franchissable que lorsqu'un sous ensemble des jetons dans les places sources satisfait les gardes sur les arcs entrants de la transition.

• Les arcs sortants peuvent contenir des expressions arithmétiques spécifiant comment calculer les valeurs produites par la transition.

• les gardes et les expressions sont exprimées à l'aide de paramètres (ou variables) et sont évaluées en associant la valeur d'un jeton à un paramètre (binding). Le développement des réseaux de Petri coloré a été motivé par le désir de développer un langage de modélisation bien-fondé théoriquement et assez versatile en même temps. Ils sont utilisés pour les systèmes de taille et de complexité qu'on retrouve dans les projets industriels. **[7]**

2.9.2 Réseau de Petri Objet :

Les réseaux de Petri Objet (OPN) étendent le formalisme des réseaux de Petri colorés avec une intégration complète des propriétés orientées objet y compris l'héritage, le polymorphisme et la liaison dynamique. L'orientation objet fournit des primitives de structuration puissante permettant la modélisation des systèmes complexes **[7]**

2.10 Conclusion

Cet article a introduit l'utilisation des réseaux de Petri dans la production manufacturière l'étude d'un cas concret a montré comment modéliser une ligne de fabrication simple. Le dimensionnement et l'évaluation des performances du système ont été étudiés à partir du graphe d'événements temporisés obtenu lors de la phase de modélisation. Cet exemple a permis de mettre en évidence, au moins partiellement, quelques avantages de l'approche par réseaux de Petri. Nous avons ainsi montré succinctement qu'une même famille d'outils est utilisée pour la spécification, la modélisation, l'analyse qualitative, le dimensionnement et l'évaluation des performances du système étudié.

Chapitre 3

Ingénierie dirigée par les modèles (IDM)

3.1 Introduction

Les fondements de cette thèse reposent d'un côté sur la théorie de la modélisation, la méta-modélisation, la manipulation et la transformation de modèles et de l'autre côté sur les concepts de conception et de développement d'outils de modélisation. Tous ces concepts sont issus du contexte de l'Ingénierie Dirigée par les Modèles (IDM). En effet, ce dernier est caractérisé par sa capacité à accélérer les développements et à faciliter la réutilisation de la conception des systèmes. Ce chapitre fournit donc une base théorique et méthodique solide sur les concepts clés et la terminologie respective couramment impliqués dans l'IDM. Un aperçu de quelques outils supports de l'IDM est également fourni.

3.2 INGENIERIE DIRIGEE PAR LES MODELES :

L'ingénierie dirigée par les modèles IDM (ou aussi Model Drivent Engineering MDE) est une pratique d'ingénierie des systèmes utilisant les capacités des technologies informatiques pour décrire à travers des modèles, concepts, et langages, à la fois le problème posé (besoin) et sa solution. Cette pratique est construite autour de deux notions fondamentales : les modèles et les transformations). En focalisant le raisonnement sur les modèles, l'IDM permet de travailler à un niveau d'abstraction supérieur et de vérifier sur une maquette numérique (ensemble de modèles qui traitent divers aspects d'un système) un ensemble de propriétés que l'on ne pouvait aborder auparavant qu'en fin de cycle de développement. L'un des apports supplémentaires du travail sur maquette numérique, en lieu et place du traditionnel code logiciel, est de fournir un référentiel central pour l'étude des différentes problématiques d'un système permettant à différents acteurs de s'intéresser à différents aspects du système à développer. [19]

3.2.1 La métamodélisation

La méta-modélisation est une activité qui consiste à définir le méta-modèle d'un langage de modélisation. Il s'agit non seulement de produire des méta-modèles mais aussi de définir la sémantique du langage. De mettre en œuvre des analyseurs, des compilateurs, des générateurs de code et plus généralement, à construire un ensemble d'outils exploitant les méta-modèles. Ainsi méta-modélisation est l'analyse, la construction et le développement des cadres, règles, contraintes, modèles et théories applicables et utiles pour modéliser une classe prédéfinie des problèmes. Comme son nom l'indique, ce concept applique les notions de méta-modèle et de modélisation en génie logiciel et génie système. Les méta-modèles sont de nombreux types et ont des applications diverses. [17]

3.2.2 Notions générales de l'IDM

Cette section vise à donner une vue simplifiée des concepts clés et de la terminologie respective couramment impliqués dans l'IDM, en répondant à des questions courantes comme: Qu'est-ce qu'un modèle, un méta-modèle et un méta-méta-modèle ? Quelle est la relation entre ces concepts ? Quelles sont les facettes clés d'un langage de modélisation et d'une méthode de modélisation, Et qu'est-ce qu'un point de vue ?

3.2.2.1 Système:

Un système est un ensemble d'éléments liés entre eux par des relations dont si un

élément soit modifié tout l'ensemble d'éléments sera modifié. Pour mieux connaitre et étudier un système, on a besoin de la simplifier et le modéliser et avoir une abstraction du système qui est un modèle. [17]

3.2.2.2 Modèle:

Un « modèle » est une représentation simplifiée d'un système où il est construit avec l'intention de décrire le système et répondre aux questions que l'on se pose sur lui. Donc un système est modélisé avec un ensemble de modèles ou chacun capture un aspect particulier. Par analogie avec les langages de programmation, un programme exécutable représente le système alors que le code source de ce programme représente le modèle. [17]

Chaque modèle est exprimé à partir d'un langage de modélisation qui doit être clairement défini, et comme l'IDM prend la définition de tout est modèle donc le langage de modélisation prend la forme d'un modèle, appelé méta-modèle,

3.2.2.3 Méta-modèle :

Un méta-modèle est un langage d'expression (langage de modélisation) d'un modèle, ou un méta-modèle représente une spécification formelle d'une abstraction. Le concept de méta-modèle permet également d'aider, pour un système donné, a la construction d'un modèle. Par conséquent, un modèle est une instance d'un méta-modèle et un modèle est lié à son méta-modèle par une relation de conformité.

Un méta-modèle est exprimé à son tour avec un langage de méta modélisation spécifié par un méta-méta-modèle. [17]

3.2.2.4 Méta-métamodèle :

Comme l'IDM prend la définition de "tout est modèle", un méta-métamodèle permet de décrire un modèle de méta-modèles. Il est un langage de description de méta-modèles. Il permet d'exprimer les règles de conformité qui lient les entités du niveau modèle à celles du niveau métamodèle. Est conçu avec la propriété de méta-circularité, c'est-à-dire la capacité de se décrire lui-même. [17]

3.3 Architecture de méta modélisation

3.3.1 Architecture MDA à quatre niveaux :

Dans l'IDM, l'acte de modélisation peut se situer à différents niveaux. Dans ce contexte, l'OMG propose une architecture de ces niveaux de modélisation et de méta-modélisation (figure 3.1). Cette architecture pose les bases des relations qui existent entre entités à modéliser, modèles, méta-modèles et méta-méta-modèles [18]

Au sommet de cette hiérarchie se trouve la couche de méta-modélisation (désignée M3) qui est principalement chargée de fournir un langage pour spécifier les méta-modèles. MOF est une couche méta-méta-modèle unique car elle est instanciée à partir de son propre modèle, c'est-à-dire que le MOF est défini dans MOF.

Dans la couche ci-dessous (désignée par M2), les méta-modèles sont définis par instanciation du méta-méta-modèle (c'est-à-dire que chaque élément du métamodèle est une instance d'un élément défini dans le méta-méta-modèle). Le langage UML est un exemple, une forme d'instance MOF.

Dans la couche en dessous de M2 (désignée par M1), les modèles sont définis en fonction de l'intérêt et des besoins de ses utilisateurs: généralement pour différents domaines d'application et différents niveaux d'abstraction, par ex. au niveau de la définition de l'entreprise, des exigences techniques ou de la conception de logiciels.

Enfin, le niveau le plus bas de la hiérarchie (la couche M0) contient des instances réelles d'éléments définis dans le modèle qui existent réellement dans le contexte d'un environnement informatique ou même dans le monde réel. [18]

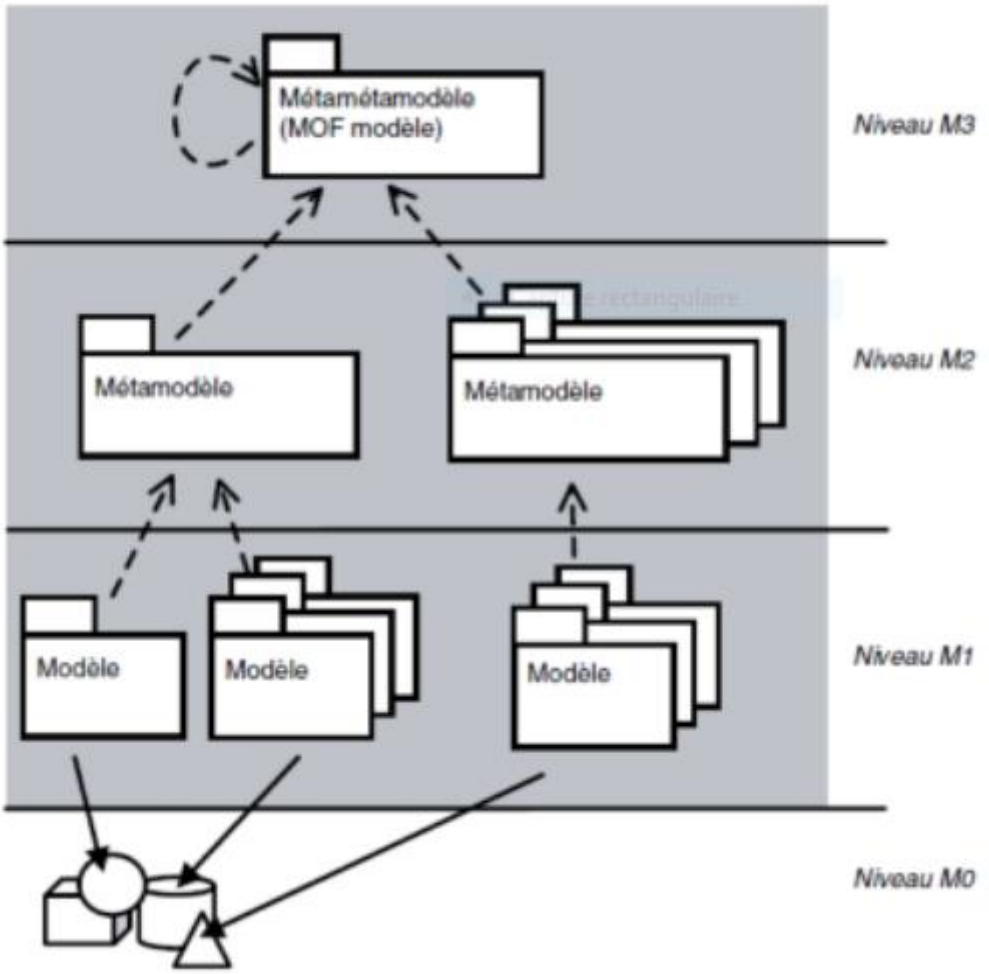


Figure 3.1 : Architecture à quatre niveaux.

3.3.2 Les standards de L'OMG

L'OMG a défini plusieurs standards pour le MDA, Nous mentionnerons les plus importants d'entre eux [21]

- **Meta-Object Facility (MOF)**

Le MOF est une norme OMG de méta modélisation et de référentiel de métadonnées. Il s'agit d'un cadre d'intégration extensible piloté par un modèle pour définir, manipuler et

intégrer des métadonnées et des données d'une manière indépendante de la plateforme. Des normes basées sur MOF sont utilisées pour intégrer des outils, des applications et des données. [21]

- **UML : Unified Modeling Language**

Étant UML est un langage de modélisation graphique à base de pictogrammes conçus comme une méthode normalisée de visualisation dans les domaines du développement logiciel et en conception orientée objet. Il est destiné à l'architecture, la conception et la mise en œuvre de systèmes logiciels complexes par leur structure aussi bien que leur comportement. L'UML a des applications qui vont au-delà du développement logiciel, notamment pour les flux de processus dans l'industrie. Il ressemble aux plans utilisés dans d'autres domaines et se compose de différents types de diagrammes. Dans l'ensemble, les diagrammes UML décrivent la limite, la structure et le comportement du système et des objets qui s'y trouvent. L'UML n'est pas un langage de programmation, mais il existe des outils qui peuvent être utilisés pour générer du code en plusieurs langages à partir de diagrammes UML. [21]

- **Object Constraint Language**

OCL est un langage informatique d'expression des contraintes utilisé par UML. Développé pour la première fois par IBM en 1995 puis standardisé par l'object management group, OCL permet d'effectuer des requêtes sur des métamodèles. OCL peut s'appliquer sur la plupart des diagrammes d'UML et permet de spécifier des contraintes sur l'état d'un objet ou d'un ensemble d'objets. [21]

- **Xml Meta data Interchange**

XMI est un cadre d'intégration basé sur XML pour l'échange de modèles et, plus généralement, de tout type de données XML. XMI est utilisé dans l'intégration d'outils, de référentiels, d'application et d'entrepôts de données. Le cadre définit des règles pour générer des schémas XML à partir d'un métamodèle basé sur la Meta object Facility (MOF). XMI est le plus souvent utilisé comme format d'échange pour UML, bien qu'il puisse être utilisé avec n'importe quel langage compatible MOF. [21]

3.4 PHASES DE MODELISATION (CIM, PIM, PSM ET CODE)

Le principe de base de l'IDM est l'élaboration de différents modèles. En partant d'un modèle métier dans lequel aucune considération informatique n'apparait (Computation Independent Model, CIM), la transformation de celui-ci en modèle d'analyse et de conception » (Platform Independent Model, PIM) et enfin la transformation de ce dernier en modèle de code (Platform Specific Model, PSM) pour l'implémentation concrète du système (figure 3.2). [18]

3.4.1 Le CIM (Computation Independent Model)

Est le modèle d'analyse de base du métier ou du domaine d'application. Ce modèle est

indépendant de tout système informatique, il décrit les concepts de l'activité métier, le savoir-faire, les processus, la terminologie et les règles de gestion (de haut niveau), il décrit aussi la situation dans laquelle le système est utilisé. C'est un modèle de très longue durée de vie, il n'est modifié uniquement que si les connaissances ou les besoins métier changent. [18]

3.4.2 Le PIM (Platform Independent Model)

Est un modèle de conception, il décrit le système informatique indépendamment de toute plate-forme technique et de toute technologie utilisée pour déployer l'application. Il représente le logique métier spécifique au système (fonctionnement des entités et des services). [18]

3.4.3 Le PDM (Platform Description Model)

Pour les modèles de description de la plate-forme sur laquelle le système va s'exécuter. Il définit les différentes fonctionnalités de la plate-forme et précise comment les utiliser.[18]

3.4.4 Le PSM (Platform Specific Model) :

Sert à la génération du code exécutable pour les plates-formes techniques particulières.
D'une manière générale on peut distinguer quatre catégories d'outils

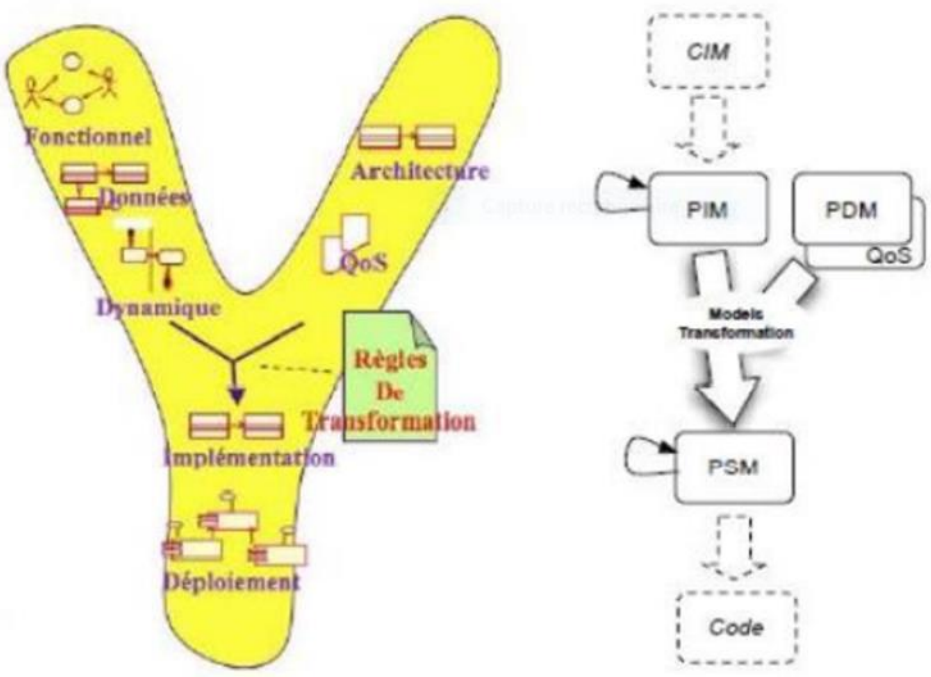


Figure 3.2 : Un processus MDA en Y dirigé par les modèles.

3.5 La transformation de modèle :

Les transformations de modèles sont au cœur de l'approche de l'ingénierie dirigée par les modèles. Mais il n'existe pas encore de consensus sur la définition et la mise en œuvre d'une transformation. Dans la littérature, de multiples approches sont proposées. Pour réaliser des transformations de modèles, ces derniers doivent être exprimés dans un certain langage de modélisation ou méta-modèle.

.

3.5.1 principe général d'une transformation :

On La définition la plus générale et qui fait l'unanimité au sein de la communauté IDM consiste à dire qu'une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources. [20]

En réalité, la transformation se situe entre les méta-modèles source et cible qui décrivent la structure des modèles cible et source. Le moteur de transformation de modèles prend en entrée un ou plusieurs modèles sources et crée en sortie un ou plusieurs modèles cibles.

Une transformation des entités du modèle source met en jeu deux étapes : [20]

- **Première étape :** permet d'identifier les correspondances entre les concepts Des modèles source et cible au niveau de leurs méta-modèles, ce qui induit l'existence d'une fonction de transformation applicable à toutes les instances du méta-modèle source.
- **Deuxième étape :** consiste à appliquer la transformation du modèle source afin de générer automatiquement le modèle cible par un programme appelé moteur de transformation ou d'exécution.

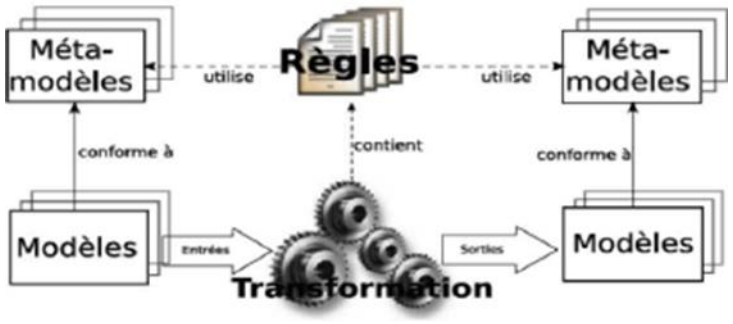


Figure 3.3 : transformation de modèles.

3.5.2 types de transformation :

Une transformation de modèles et en correspondances des éléments des modèles cibles et sources. On distingue les types de transformation suivant : [20]

3.5.2.1 Les transformations simples (1 vers 1) :

Qui associent à tout élément du modèle source au plus un élément du modèle cible.

3.5.2.2 Les transformations multiples (M vers 1) :

Qui prennent en entrée un ensemble d'éléments du modèle source et produisent en sortie un ensemble d'éléments (généralement différent) du modèle cible.

3.5.2.3 Les transformations de mis à jour

ces transformations sont caractérisées par l'absence de modèle cible, et agissent donc directement sur le modèle source. Les transformations de restructuration sont des exemples typiques de ce type de transformation.

3.5.3 Typologie de transformation

En partant des méta-modèles sources et cibles de la transformation, on distingue deux types

de transformations : les transformations endogènes et exogènes.
Une transformation est dite endogène si les modèles impliqués sont issus du même méta-modèle. Mais lorsque les modèles sources et cibles sont de différents méta-modèles, la transformation est dite exogène ou encore translation. [20]

- **Transformations endogènes**

Se situe dans le même espace technologique et les modèles source et cible sont conformes au même méta-modèle. Par exemple transformation d'un modèle UML en un autre modèle UML.

- **Transformations exogènes**

Se situe entre é espaces technologique différents et les modèles source et cible sont conformes à des méta-modèle différents.

- **Verticalité et horizontalité**

Une transformation de modèle peut aussi être classé selon un autre critère qui est le niveau d'abstraction si les modèles cible et source appartiennent au même niveau d'abstraction donc on a une transformation horizontale et si les modèles cible et source appartiennent à deux niveaux d'abstractions différents donc on a une transformation verticale. Par exemple, une génération de code est une transformation verticale

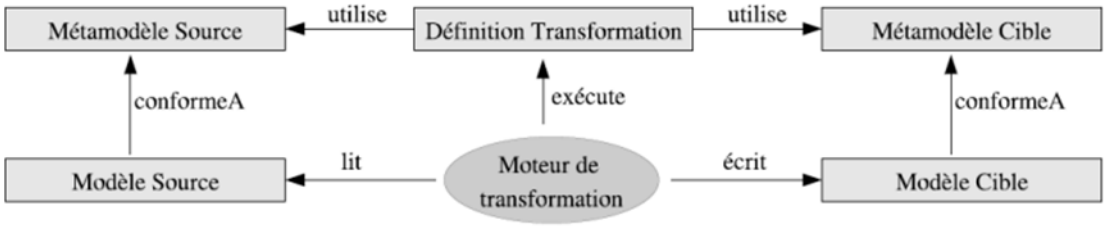


Figure 3.4 : Concepts de base des transformations de modèles.

3.5.4 Approches de transformation de modèles

La prise en compte des différents critères de classification tel que spécification, règles de transformation, relation entre source et cible, incrémentation, traçabilité, etc. permet d'identifier plusieurs approches de transformation dont les principales catégories sont présentées ci-dessous [Figure 3.]. Au plus haut niveau de la classification, on distingue deux approches de transformations : les transformations de « modèles à modèles » et les transformations de «modèles vers du code source» (génération de code). Dans chacune de ces deux catégories, on distingue plusieurs sous-catégories comme le montre la figure 3-5.

3.5.4.1 Les approches de modèle au code (Model To Text M2T)

La génération de code peut être considérée comme un cas particulier de transformation de type modèle vers texte. Pour cela, il faut définir un méta-modèle correspondant au langage de programmation cible.

Dans le contexte de la génération de code, l'utilisation d'une transformation aurait pour objectif de transformer un modèle de haut-niveau d'abstraction en un second modèle de plus bas niveau d'abstraction (proche du code exécuté). De cette manière, on facilite la conception en épargnant à l'utilisateur les détails d'implémentation, en traduisant automatiquement ce modèle. Nous faisons appel pour cela à des transformations de type M2T que nous appliquons à des modèles.

Diagramme d'état transition afin de générer du code compatible avec l'outil RDP. Dans ce cas, on parle d'une transformation M2T endogène (le modèle et le code généré sont conforme au même méta- modèle RDP) et verticale (le modèle et le code généré ne sont pas dans le même niveau d'abstraction). Le projet Model to Text (M2T) se concentre sur la génération d'artefacts textuels à partir de modèles, Une grande classe de transformations traduit les modèles en texte. Le texte peut être du code généré, d'autres modèles dans la syntaxe textuelle ou d'autres objets textuels tels que des rapports ou des documentations. Le texte est généralement généré à l'aide de modèles, une technique extrêmement bien établie (par exemple dans le développement web). [20]

3.5.4.2 Les approches de modèle au modèle (Model To Model M2M)

Ces transformations ont beaucoup évolué depuis l'apparition de MDA. Ce type de transformations permet la génération de plusieurs modèles intermédiaires avant d'atteindre le modèle de code, afin d'étudier les différentes vues du système, son optimisation, la vérification de ses propriétés et sa validation. [20]

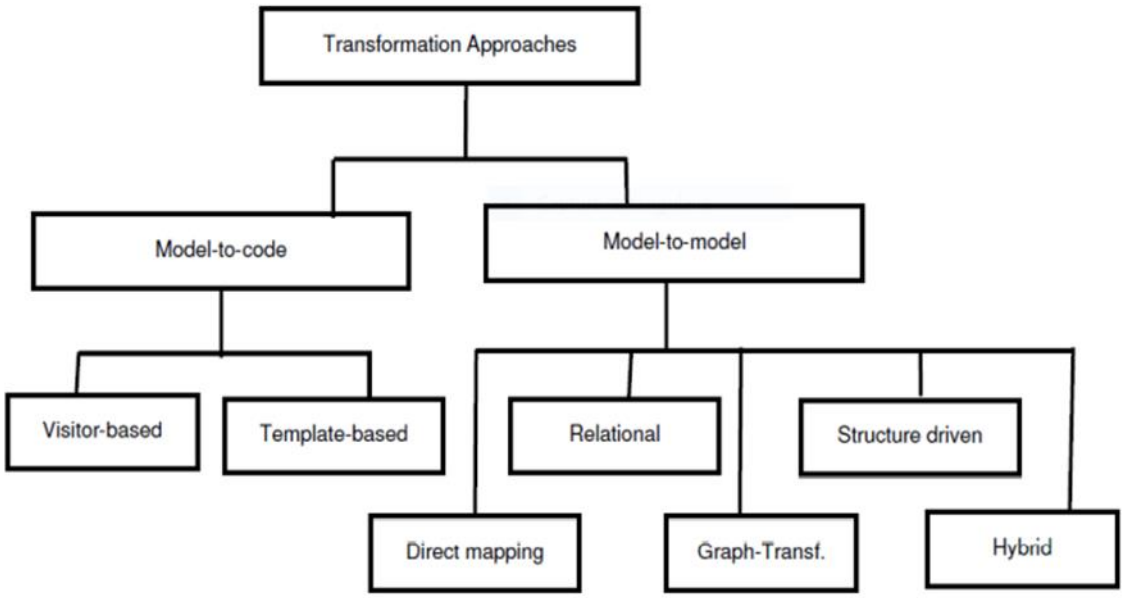


Figure 3.5 : Les approches de transformation de modèles.

3.6 Grammaire de Graphe

Une grammaire de graphe est un ensemble P de règles muni d'un graphe initial S et d'un ensemble T de symboles terminaux. [16]

3.6.1 Principe de transformation de graphe

Une transformation de graphe consiste en l'application d'une règle à un graphe et itérer ce processus. Chaque application de règle transforme un graphe par le remplacement d'une de ses parties par un autre graphe. Donc, la transformation de graphe est le processus de choisir une règle d'un ensemble indiqué, appliquer cette règle à un graphe et réitérer le processus jusqu'à ce qu'aucune règle ne puisse être appliquée. La transformation de graphe

est spécifiée sous forme d'un modèle de grammaires de graphes, ces dernières sont une généralisation, pour les graphes, des grammaires de Chomsky. Elles sont composées de règles dont chacune est composée d'un graphe de côté gauche (LHS) et d'un graphe de côté droit (RHS). [16]

3.6.2 Les outils de transformations de graphes

Il existe plusieurs outils implantant les systèmes de réécriture de graphes. Les plus intéressants sont : [22]

- **TGG "Triple Graph Grammar"**

L'approche des grammaires de graphes triples (Triple Graph Grammar : TGG), introduit par Andy Schürr est une tentative de créer une méthode pour connecter différents systèmes/modèles par rapport à certains règles/critères prédéfinis, de sort que les changements dans un système/modèle conduiraient inévitablement à des changements dans l'autre. TGG peut être utilisé dans différents transformation de modèles et les scénarios de synchronisation. TGG est spécifié pour les transformations bidirectionnelles (transformation à l'avant et en arrière). [22]

- **Viatra**

Un plugin eclipse développé en 2005 à l'université de Budapest. Le logiciel utilise le langage VPM pour la modélisation. Pour définir les règles de transformations, l'utilisateur peut utiliser des motifs récursifs et des motifs de négations représentant les conditions d'application négatives. L'ordonnancement dans l'application des règles est basé sur une machine à états abstraite. [22]

- **Eclipse Modeling Galileo**

Un environnement spécifique aux transformations de graphes intégré à la plateforme eclipse. Le Framework principal est celui utilisé pour la modélisation. Il est appelé Eclipse Modeling Framework (EMF). Les modèles y sont décrits en XML, mais peuvent être spécifiés dans des documents UML/SysML. L'interopérabilité avec d'autres outils basés sur Eclipse est l'un des points forts de ce Framework. Un de ces outils appelé Graphical Modeling Framework (GMF) permet le développement d'éditeur graphique de modèles. Les transformations de modèles sont exprimées en langage ATL (Atlas Transforming Language). [22]

- **Fujaba "From UML to Java and back again"**

Fujaba est un projet d'outil open source UML CASE fournissant aux développeurs un support pour l'ingénierie et la réingénierie logicielles basées sur des modèles. Le projet Fujaba vise à développer et étendre la suite d'outils Fujaba et ainsi offrir une plate-forme extensible pour les chercheurs en génie logiciel. Le groupe de développement Fujaba développe et étend en permanence Fujaba et de nombreux outils connexes. A l'origine, Fujaba visait prendre en charge l'ingénierie avant et arrière des logiciels. C'est pourquoi Fujaba est l'acronyme de « From UML to Java and back again ». [22]

- **AGG "Attributed Graph Grammar"**

Est un environnement de développement pour les systèmes de transformation de graphes attribués prenant en charge une approche algébrique de la transformation de graphe. Il vise à spécifier et prototyper rapidement des applications avec des données complexes structurées en graphes. [22]

3.7 Conclusion

Dans ce chapitre, nous avons présenté les notions de base de l'ingénierie dirigée par les modèles ou nous avons mis l'accent sur la transformation de modèles. Nous avons aussi décrit le rôle principal de la transformation dans le cadre de l'approche IDM et ses activités. L'objectif visé dans notre projet est d'intégrer les notions des grammaires de graphes dans le processus de transformation de modèles. Parmi ces types de transformations, la transformation de modèle vers modèle. En utilisant une grammaire de transformation TGG (triple graph grammar). Avec TGG on peut définir, déclarer une transformation bidirectionnelle. Un modèle peut être transformé en un autre modèle en se basant sur un modèle de correspondance. Le chapitre suivant explicitera plus particulièrement le formalisme TGG.

Chapitre 4

Implémentation et mise en œuvre

4.1. Introduction

Dans ce chapitre nous présentons l'environnement d'implémentation que nous avons utilisé pour notre recherche et le processus de transformation de modèles avec TGG. L'objectif principal de ce travail est d'utiliser l'outil TGG-interpreter pour maîtriser le passage des modèles de diagrammes d'états-transitions UML vers les Réseaux de Petri, cette migration consiste à la traduction du schéma source en un schéma cible.

4.2. Environnement d'implémentation :

Nous avons utilisé TGG et JDK pour transformer et manipuler les modèles dans un environnement technique adapté à la modélisation et la transformation des modèles, Pour transformer les diagrammes d'état transition vers les réseaux de Petri, on propose une méthode qui s'appuie sur les trois étapes suivantes :

- ✧ Construction de méta-modèle des diagrammes d'état transition, permet la transformation des métamodèles de l'espace technique EMF en grammaires de graphes dans l'espace technique TGG.
- ✧ Construction de méta-modèle des réseaux de Petri. et consiste à spécifier les règles de transformation et la grammaire de graphe de correspondance.
- ✧ Définition des règles de la transformation. sert à générer le moteur de transformation et exécuter ce dernier sur un modèle source pour avoir un modèle cible

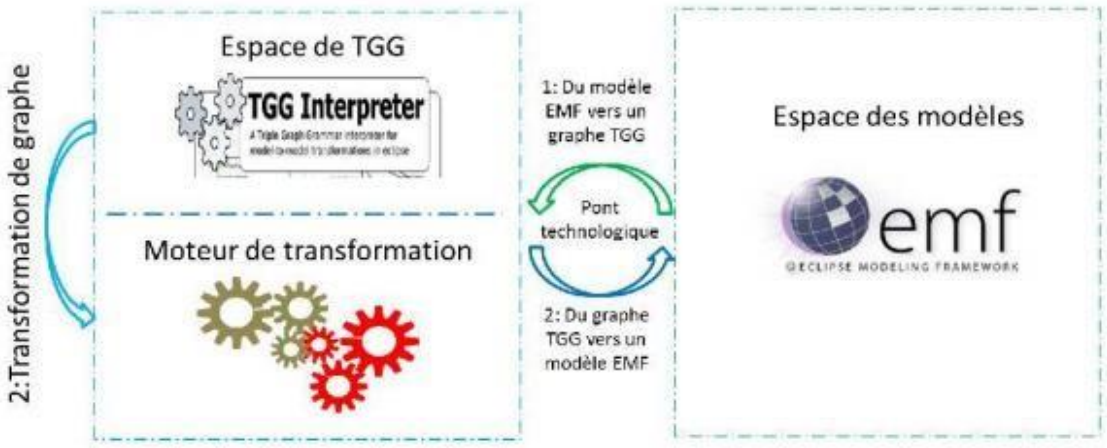


Figure 4.1 : Principe de mise en œuvre de transformation de modèles.

4.2.1. Eclipse

Eclipse Est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la fondation Eclipse visant à développer un environnement de production de logiciels libre qui soit extensible, universel et polyvalent, en s'appuyant principalement sur Java, Son objectif est de produire et fournir des outils pour la réalisation de logiciels, englobant les activités de programmation (notamment environnement de développement intégré et Framework)

4.2.2. Eclipse Modeling Framework

Eclipe Le projet EMF est un cadre de modélisation et une installation 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. [23]
EMF (noyau) 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. Veuillez consulter le projet de modélisation pour un aperçu des technologies EMF. . [23]

EMF se compose de trois éléments fondamentaux :

- EMF - Le framework : EMF principal comprend un méta-modèle (Ecore) pour décrire les modèles et la prise en charge de l'exécution des modèles, y compris la notification des modifications, la prise en charge de la persistance avec la sérialisation XMI par défaut et une API réfléchissante très efficace pour manipuler les objets EMF de manière générique.
- EMF.Edit - Le framework : EMF.Edit inclut des classes réutilisables génériques pour la création d'éditeurs pour les modèles EMF. Il offre Classes de fournisseur de contenu et d'étiquettes, prise en charge des sources de propriétés et autres classes pratiques qui permettent aux modèles EMF d'être affichés à l'aide de visualiseurs et de feuilles de propriétés de bureau standard (JFace).Une structure de commandes, comprenant un ensemble de classes d'implémentation de commandes génériques pour la création d'éditeurs prenant en charge l'annulation et le rétablissement entièrement automatiques.
- EMF.Codegen - La fonction de génération de code EMF est capable de générer tout ce qui est nécessaire pour créer un éditeur complet pour un modèle EMF. Il comprend une interface graphique à partir de laquelle les options de génération peuvent être spécifiées et les générateurs peuvent être invoqués. L'outil de génération s'appuie sur le composant JDT (Java Développement Tooling) d'Eclipse. [23]

4.2.3. Le Java Développement Kit (JDK)

Ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi

que les outils avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la machine virtuelle Java. [24]

Le Java Développement Kit (JDK) désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé, transformé en byte code destiné à la machine virtuelle Java.

Il existe plusieurs éditions de JDK, selon la plate-forme Java considérée (et bien évidemment la version de Java ciblée) :

- JSE pour la Java 2 Standard Edition également désignée J2SE ;
- JEE, sigle de Java Enterprise Edition également désignée J2EE ;
- JME 'Micro Edition', destinée au marché mobiles ;

À chacune de ces plateformes correspond une base commune de développement Kits, plus des bibliothèques additionnelles spécifiques selon la plate-forme Java que le JDK cible, mais le terme de JDK est appliqué indistinctement à n'importe laquelle de ces plates-formes.

4.2.4. TGG interpreter

Le TGG Interpréter a été développé pour la transformation de modèles TGG et est un outil de mise à jour incrémentale résultant de la comparaison de TGG et de la norme de l'OMG bidirectionnel QVT (Query/View/Transformation) pour les transformations de modèles. [17]

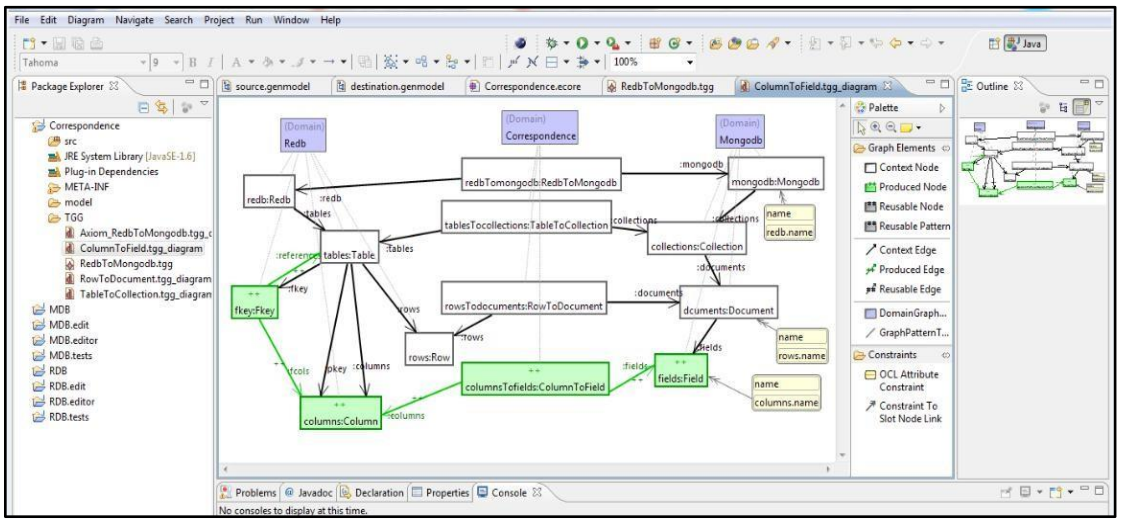


Figure 4.2 : Éditeur de Règle TGG de TGG Interpreter.

Une bonne transformation de modèles dépend d'une bonne définition des règles de transformation. TGG se compose de trois graphes, chaque graphe appartient à un domaine et chaque domaine contient une règle de grammaire de graphe. Le domaine sur la gauche est appelé source et celui sur la droite est appelé cible, le domaine de correspondance relie les deux (figure 4.2), chaque domaine est attribué à un métamodèle. Les nœuds de ce domaine sont les classes de ce métamodèle, il existe différents nœuds de règle de transformation : nœud de contexte, nœud de production, les contraintes et les nœuds réutilisables. [25]

❖ **nœud de contexte :** influencent la transformation, ils peuvent apparaître comme des nœuds de contexte dans les règles de TGG. La présentation graphique d'un nœud de contexte est présentée d'une case avec un contour noire. Les nœuds de contexte doivent être appariés avec les objets du modèle précédemment traités. Cela signifie que le TGG, tel qu'il est présenté à ce jour, besoin de spécifier une grammaire complète pourtoutes les parties de modèle. En revanche, TGG permet également de formuler une grammaire partielle sur un modèle. Là, les nœuds de contexte ne doivent pas nécessairement être préalablement traitées par une autre règle.

❖ **Nœud de production** : Les nœuds de production sont affichés sous forme de boîtes vertes avec une bordure verte et une label "++". Les arrêtes de production sont affichés sous forme de flèches de couleur vert foncé avec une étiquette "++". Les nœuds de production ne doivent pas correspondre à nœud qui existe déjà. Si on trouve ces éléments dans le domaine source, les éléments de modèle cible et de modèle de correspondance peuvent être créés selon la règle. Tous les objets créés sont liés aux nœuds de contexte de la règle. En conséquence, un objet de modèle ne peut être présenté comme un nœud de production qu'une seule fois. Nous appelons cela la sémantique bind-only- once pour les de nœuds de productions.

❖ **Nœud réutilisable :** A noter que les types de composants ne doivent pas être générés, mais peuvent être réutilisés encore et encore depuis des nœuds déjà existants avec les propriétés requises. Par conséquent, nous appelons ces nœuds "les nœuds réutilisables". Graphiquement sont représentés en gris et avec []. La sémantique des nœuds réutilisables est qu'ils peuvent être nouvellement générées ou réutilisés de façon arbitraire. La couleur grise reflète le fait que, chaque nœud réutilisable pourrait être soit en noir (un nœud du côté gauche de la règle de TGG, à savoir qu'il est réutilisé) ou en vert (un nœud du côté droit de la règle de TGG, à savoir qu'il est nouvellement généré), qui peut être choisi à chaque fois que la règle est appliquée. Nous pourrions remplacer par un nombre exponentiel de règles de TGG. Mais, le concept de nœuds réutilisables permet de réduire le nombre et la complexité des règles de TGG. En plus si l'exemple est complexe, les règles sans nœuds réutilisables peuvent générer un désordre. Les nœuds réutilisables nous permettre d'avoir plus de règles simples et de concentrer sur l'essentiel des modèles pertinents.

❖ **Les contraintes :** la contrainte réelle dans un nœud de contrainte peut être toute expression OCL qui se réfère à des objets qu'il est attaché. Les restrictions sont seulement nécessaires pour les mettre en œuvre des transformations ou des synchronisations plus efficace. Une contrainte OCL représentée avec la couleur jaune et doit être attachée avec un autre nœud.

4.3. Étude de cas

Afin de bien comprendre notre contribution sur la transformation de modèles basée sur les grammaires de graphes, nous présentons une étude de cas qui illustre le processus de spécification de transformation et leur mise en œuvre. Nous utilisons une étude de cas de transformation des diagrammes d'état transition en un réseau de petri.

4.3.1. Les métamodèles

Dans ce qui suit, nous présentons les métamodèles source et cible utilisés ainsi celui de correspondance.

4.3.1.1. Métamodèle de diagramme d'état transition :

La figure 4.3 montre le métamodèle d'un diagramme d'état transition. Le métamodèle indique qu'un diagramme d'état transition se compose de transition et State qui contient des initialState et SimpleState et finalState et compositionState qui contient (région). La transition contient les évent et les action et les condition qui compose des variables et Operator et datavalue.

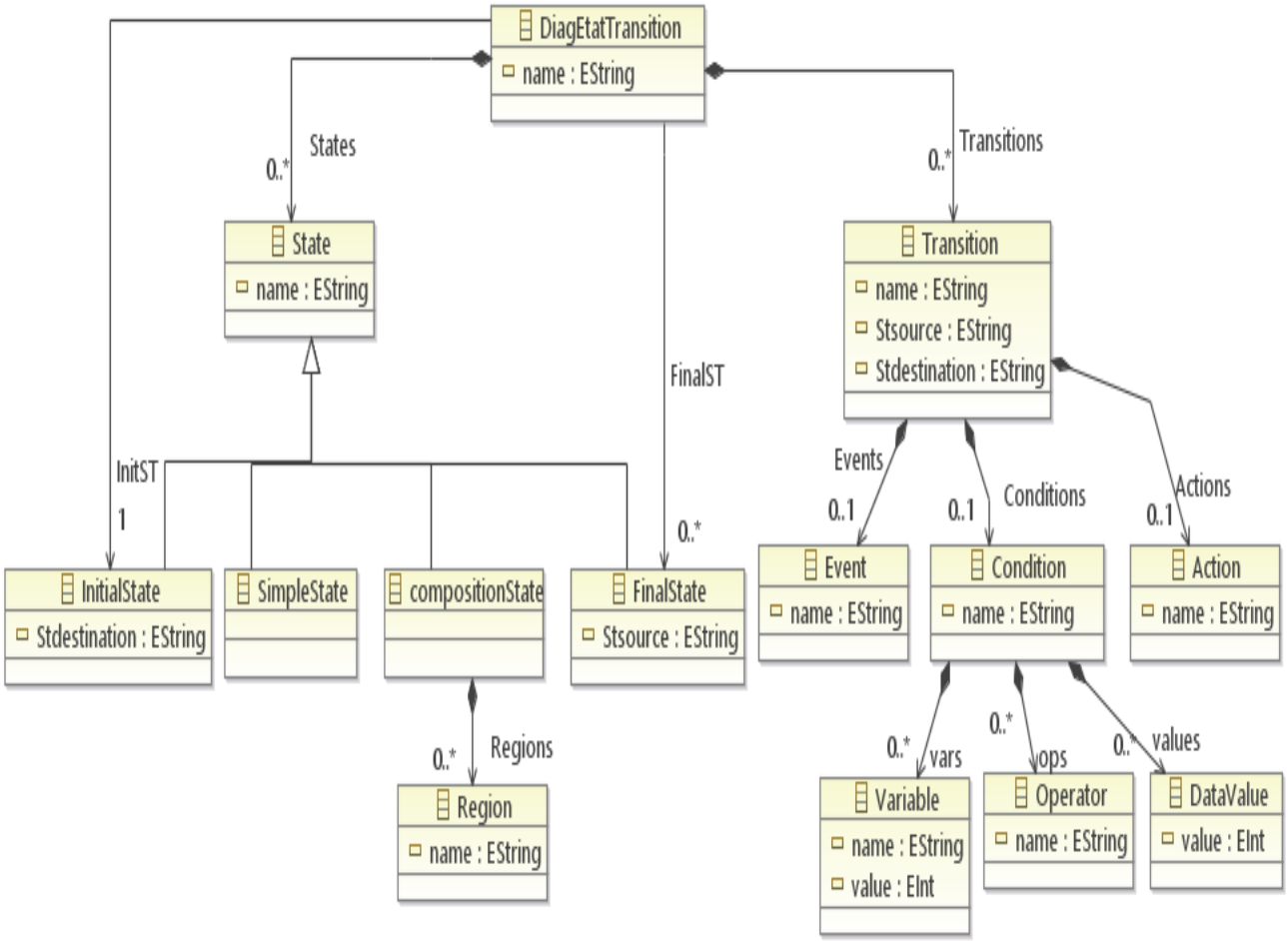


Figure 4.3 : Métamodèle de diagramme d'état transition

4.3.1.2. Métamodèle de réseau de petri :

La figure 4.4 montre le métamodèle de réseau de petri. Le métamodèle se compose Des places qui obtient token et la RTransition qui contient une condition (variable, operator, datavalue)

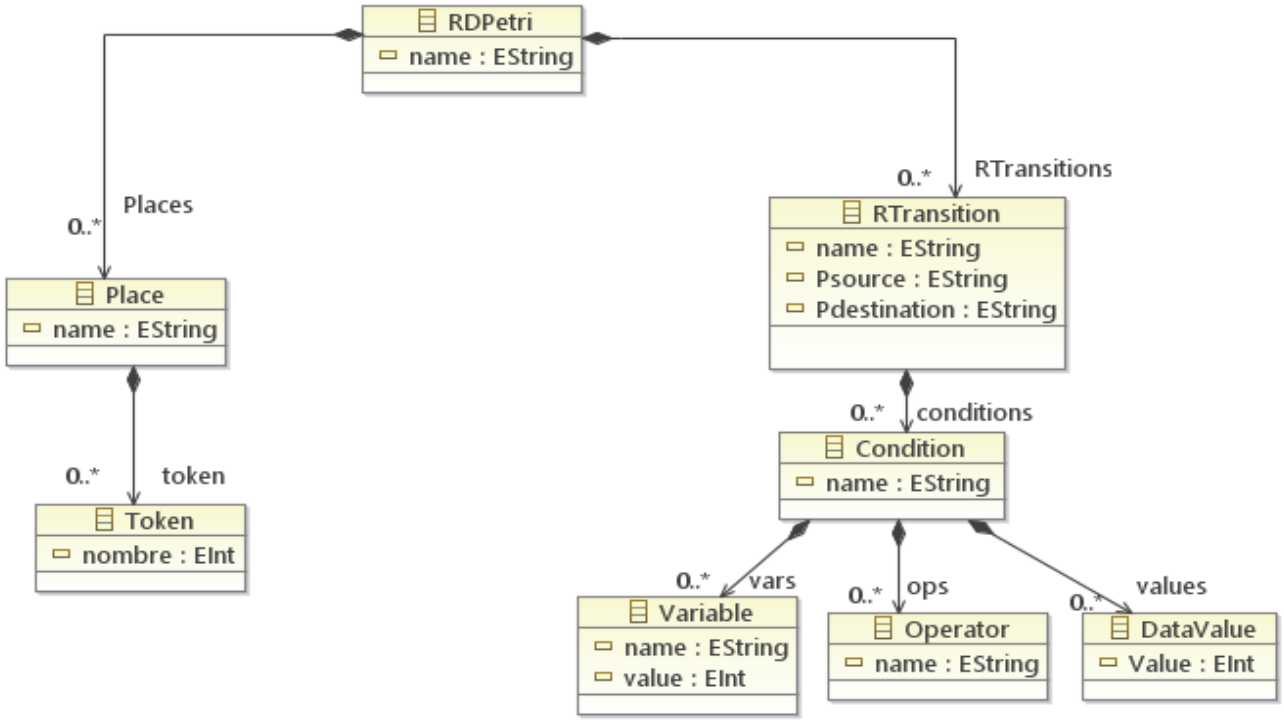


Figure 4.4 : Métamodèle réseau de petri

4.3.1.3. Métamodèle de correspondance :

Dans le projet EMF nommé (Correspondance), nous allons créer un dossier (TGG) pour définir toutes les règles de transformation**,** La figure (4.5) illustre le métamodèle de correspondance de l'exemple étudié où DiagEtatTransition2RDPetri présente la correspondance entre un diagramme d'état transition et un réseau de pitre.

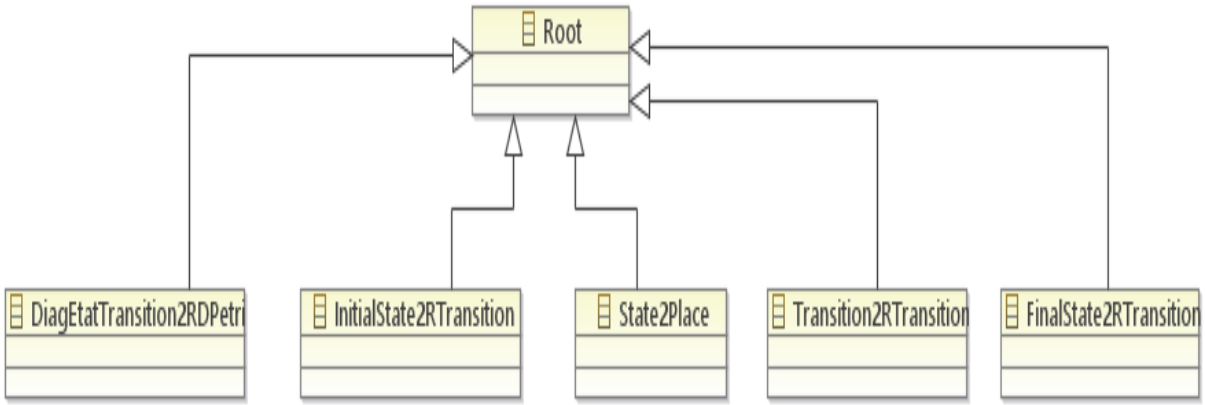


Figure 4.5 : Métamodèle de correspondance.

4.3.2. Génération des outils pour la transformation

En se basant sur le méta-modèle (voir la figure 4.6) on peut générer un outil qui nous Permettra de créer des exemples de diagramme d'état transition avec les étapes suivantes :

- Nous allons créer un projet EMF vide, puis un diagramme Ecore avec leur nom (DiagEtatTransition) dans le dossier (Modelé)

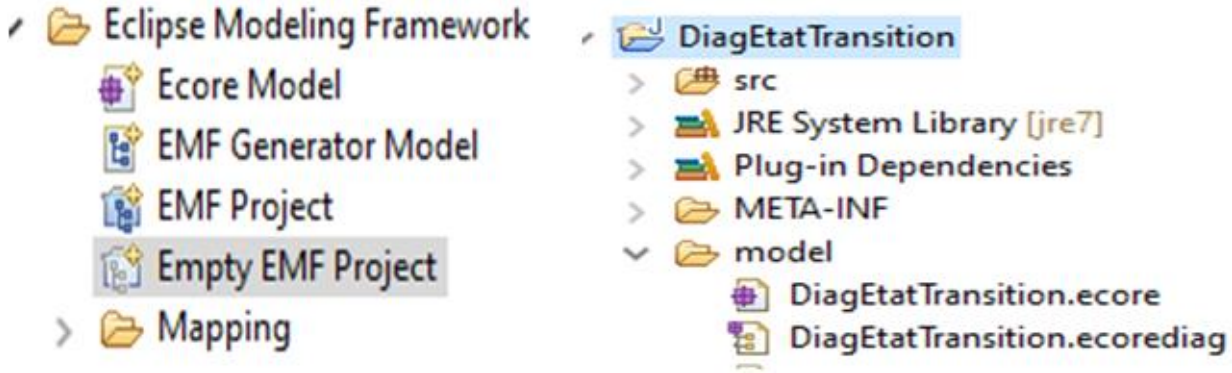


Figure 4.6 : Création d'un projet EMF et un diagramme Ecore.

Les éléments de notre méta-modèle de diagramme d'état transition sont affichés Dans le fichier (DiagEtatTransition. Ecore) (figure 4.7)

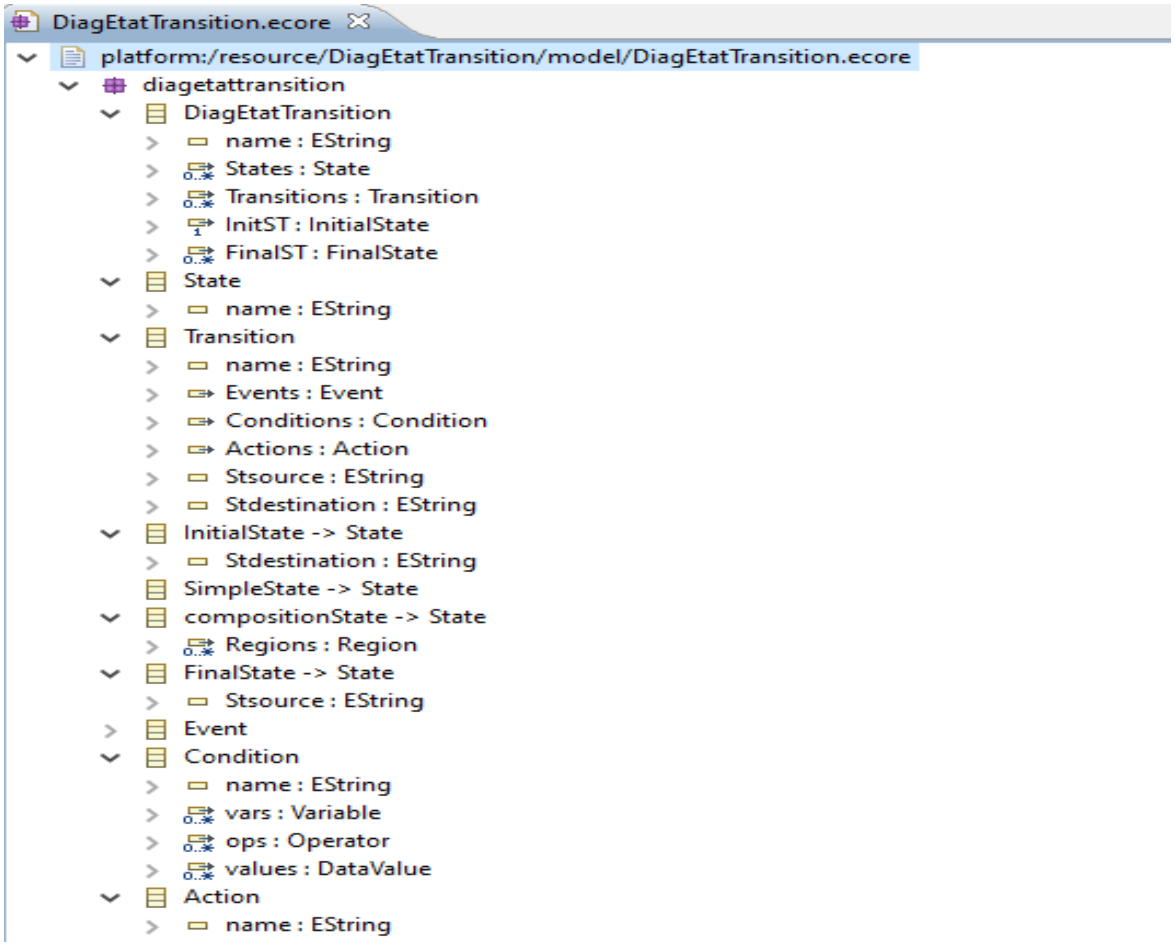


Figure 4.7 : Les éléments de diagramme d'état transition.ecor

Nous ferons de même pour la réseau de petri, nous créerons un projet (Empty EMF Project) , dans le dossier model nous représenter le schéma de méta-modele cible (RDP.ecorediag), puis le fichier (RDP.ecore) sera automatiquement affiché, et qui contient les éléments de réseau de petri . Dans le projet EMF nommé (Correspondance), dans le dossier model nous allons présenter le schéma de méta-model de correspondante (Correspondence.ecorediag), le fichier (Correspondence.ecore) s'affichera automatiquement, mais nous devrons saisir les attributs et leur type pour chaque règle de transformation (voir figure 4.8)

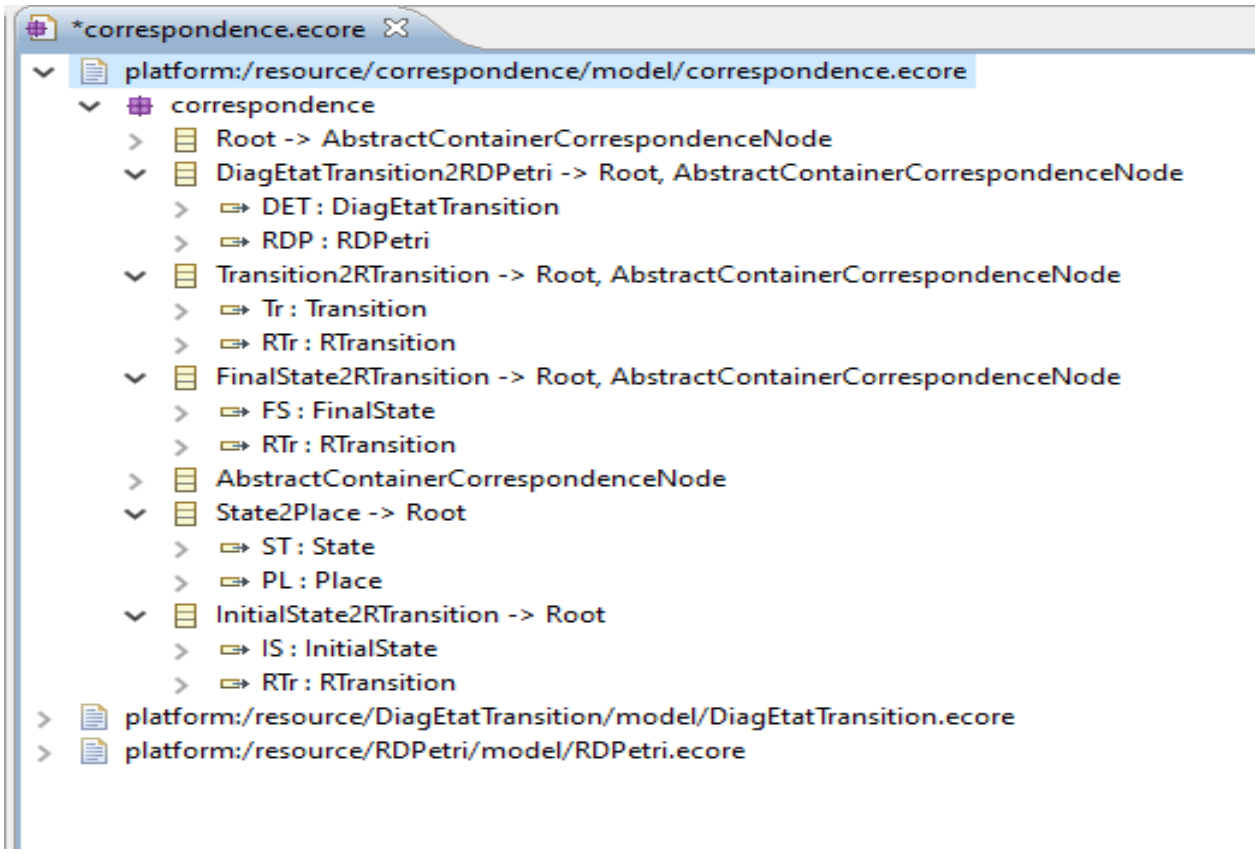


Figure 4.8 : Correspondence.ecore

4.3.3. Définition de règles de TGG :

L'idée générale dans la définition des règles de transformation est de spécifier des règles TGG qui transforment chaque classe du métamodèle source en une classe du métamodèle cible. **.** Chaque règle est composée de deux parties, la partie gauche (LHS) et la partie droite (RHS). Chaque partie peut être un sous graphe des formalismes considérés dans la transformation

4.3.2.1. Règle diagEtatTransition2RDPetri_daigram (Axiome) :

La figure (4.9) présente l'axiome de la transformation d'un diagramme d'état transition en réseau de Pétri, Sur le côté gauche, il montre un objet racine d'un diagramme d'état transition, sur le côté droit il montre un objet racine d'un réseau de Pétri, et dans la partie médiane, il montre un nœud de correspondance relative des deux

A partir de cet axiome, nous discutons maintenant les autres constructions qui se produisent dans le diagramme d'état transition et montrons comment les états correspondants sont créés dans le réseau de Pétri.

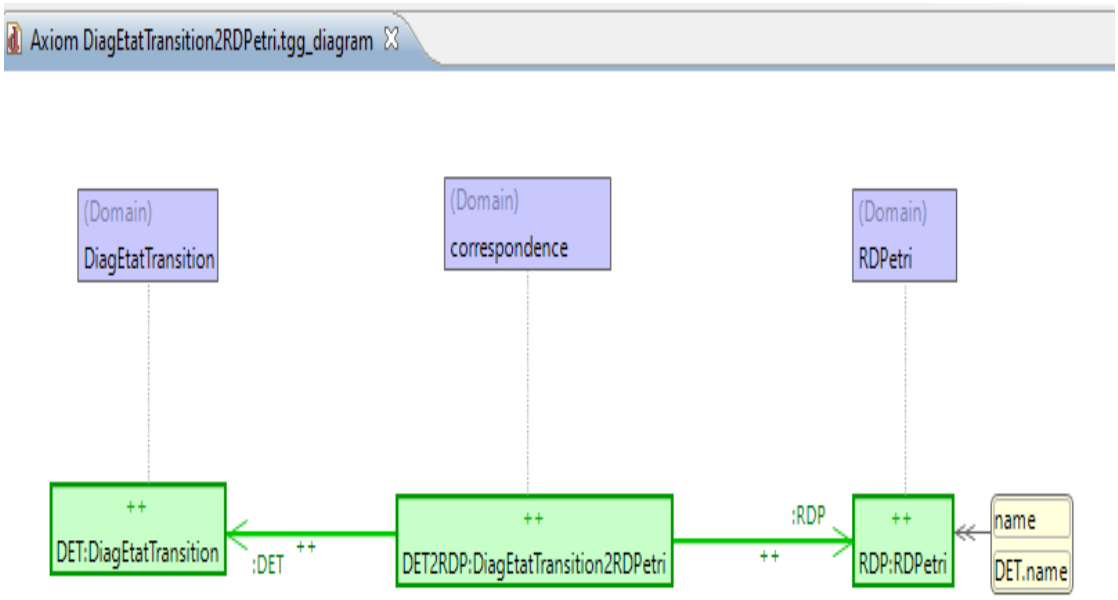


Figure 4.9 : Présentation graphique de l'axiome (diagEtatTransition2RDPetri).

4.3.2.2. Règle InitialState2RTransition :

Maintenant, on va commencer les règles pour les State, la première règle est pour le State initial. La figure 4.10 montre l'idée de cette transformation. Nous supposons que Règle diagEtatTransition2RDPetri existe déjà (représenté en noir) et maintenant le state initial est ajouté en tant que state de source en vert.
Sur le côté gauche, il montre l'initialState, sur le côté droit il montre un objet racine d'un Rtransition, et dans la partie médiane, il montre un nœud de correspondance relative des deux (**InitialState2RTransition**)

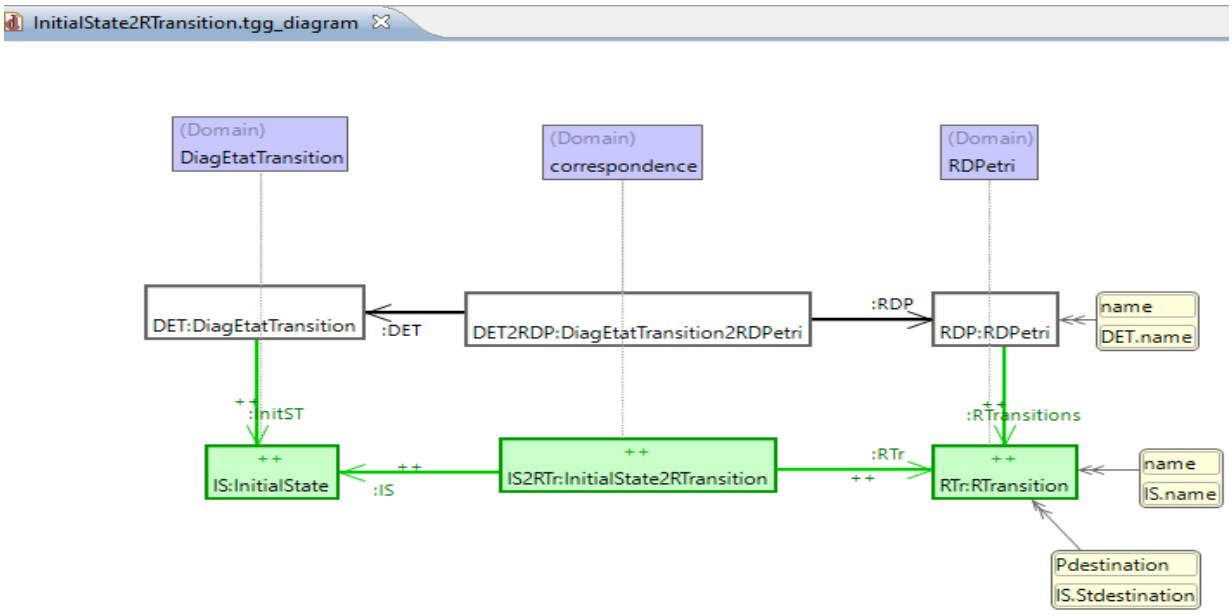


Figure 4.10 : Règle InitialState2RTransition

4.3.2.3. Règle State2Place :

La règle TGG correspondant est illustré à la figure 4.11 Sur le côté gauche (dans le domaine de diagramme d'état transition), nous voyons State ajoutée entre les deux edges. Sur le côté droit (dans le domaine de réseau de Pétri), nous voyons une place.

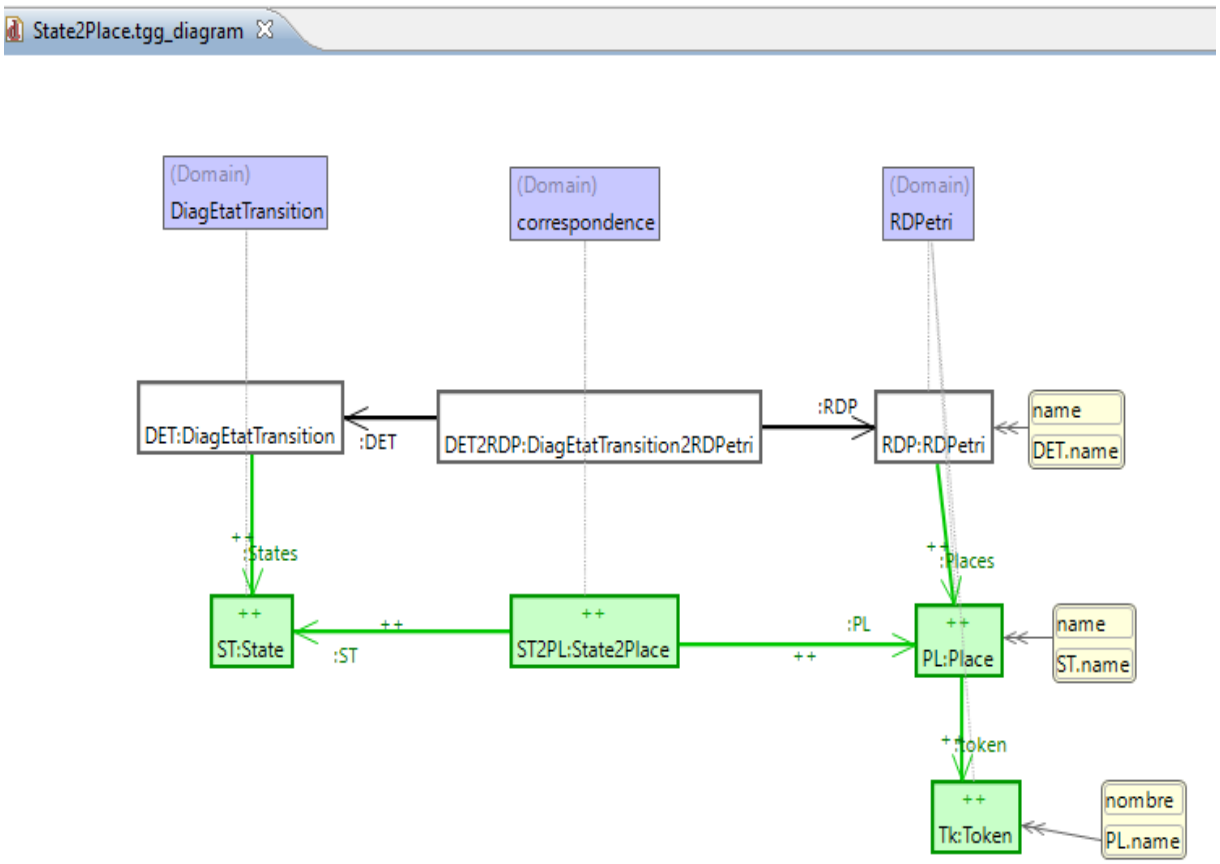


Figure 4.11 : Règle State2Place

4.3.2.4. Règle transition2Rtransition :

La règle TGG correspondant est illustrée à la figure 4.12 Sur le côté gauche (dans le domaine de diagramme d'état transition), nous voyons transition ajoutée entre les deux edges. Sur le côté droit (dans le domaine de réseau de Pétri), nous voyons une nouvelle Rtransition.
Et dans la partie médiane, il montre un nœud de correspondance relative des deux (transition2RTransition)

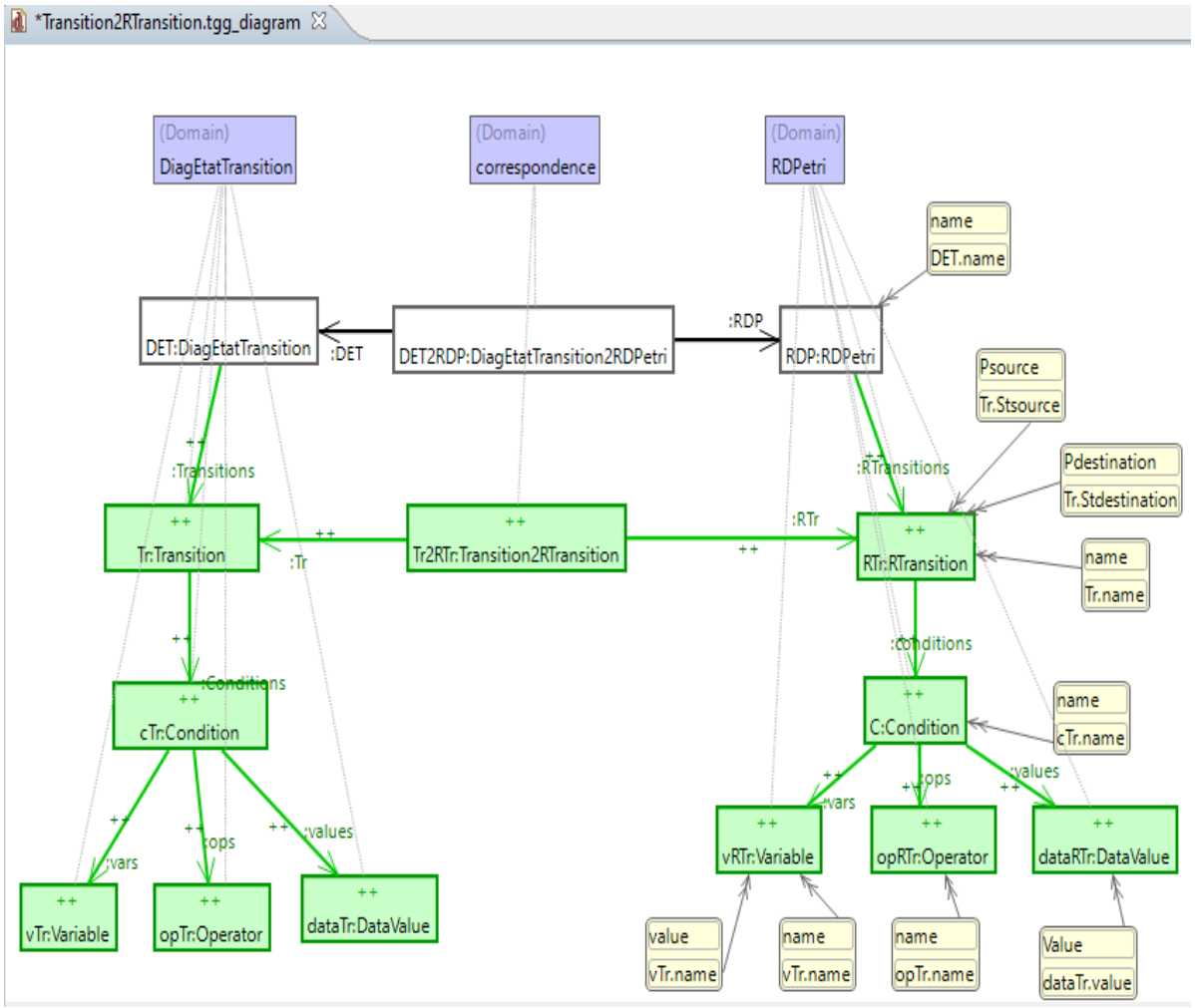


Figure 4.12 : Règle transition2RTransition

4.3.2.5. Règle FinalState2Rtransition :

L'idée de la transformation du State final est montré dans la figure 4.13 le State final sera remplacé par une transition qui n'a pas des arcs sortant.
Sur le côté gauche, il montre le finalState, sur le côté droit il montre Rtransition, et dans la partie médiane, il montre un nœud de correspondance relative des deux (**FinalState2RTransition**)

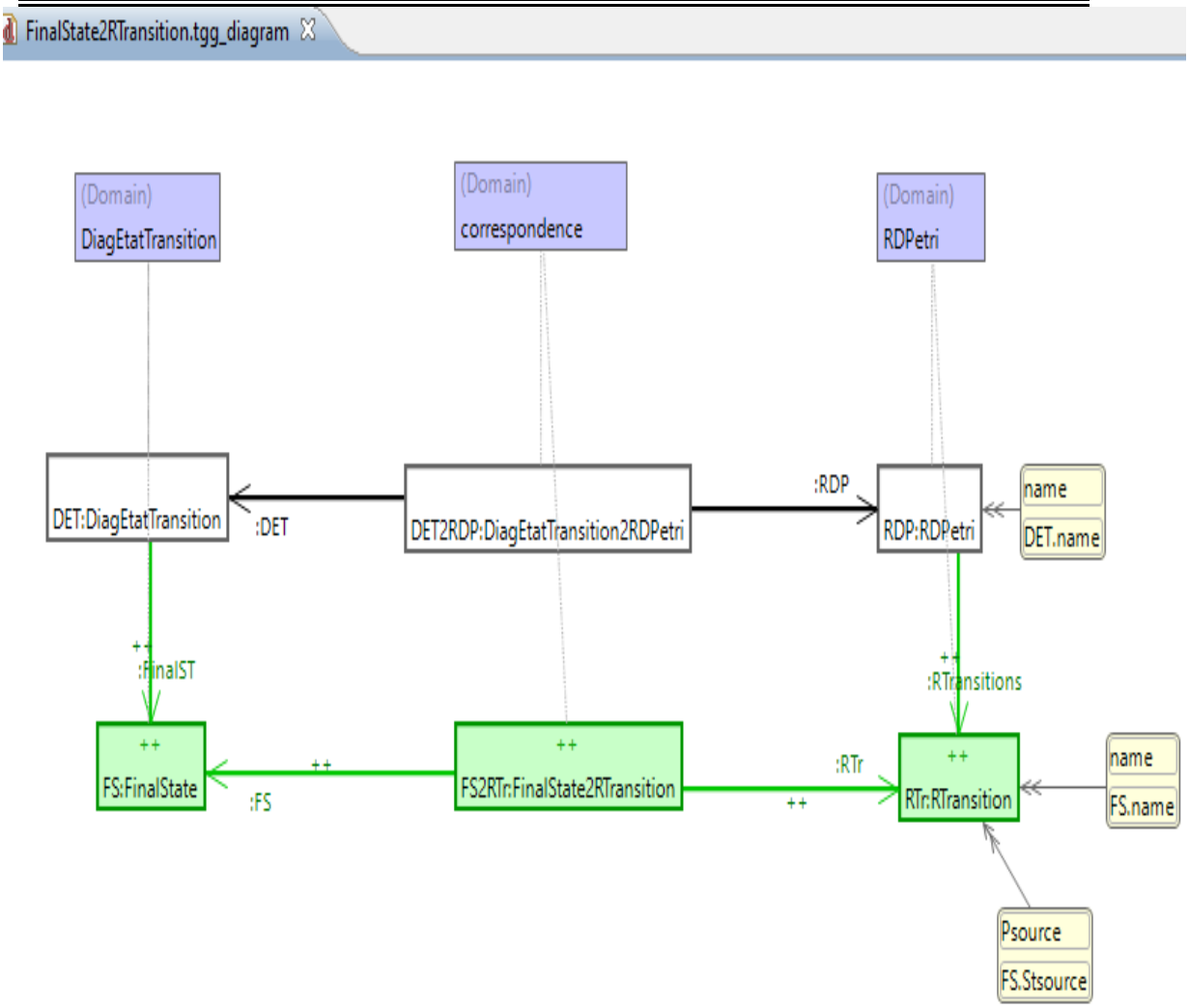


Figure 4.13 : Règle FinalState2Rtransition

4.3.4. Création de genmodel :

On passe à l'étape de création de genmodel à partir du modèle défini précédemment, il est possible de générer du code Java dédié à la création des instances de ce modèle. Nous allons créer un fichier EMF Générateur Model avec un nom (source.genmodel) dans le dossier model de projet (DiagEtatTransition), ensuite nous allons générer les projets (DiagEtatTransition.edit/) (DiagEtatTransition.editor/) (DiagEtatTransition.tests) voir figure (4.14)

Nous ferons de même pour le réseau de pitre

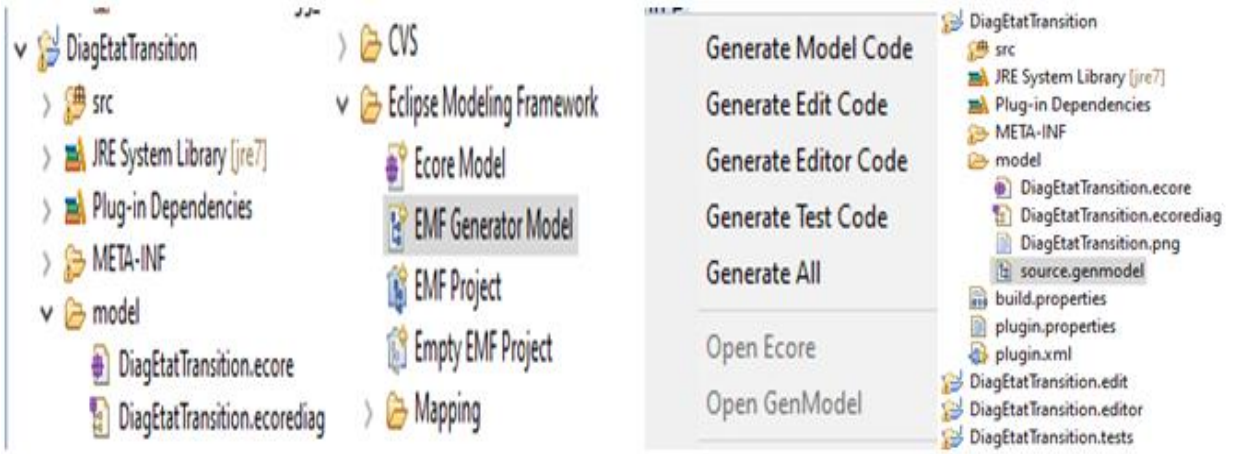


Figure 4.14 : génération des projets .édit/.editor/.tests

La génération de code nécessite la création d'un modèle de génération, Ce modèle contient des informations dédiées uniquement à la génération et qui ne pourraient pas être intégrées au modelé

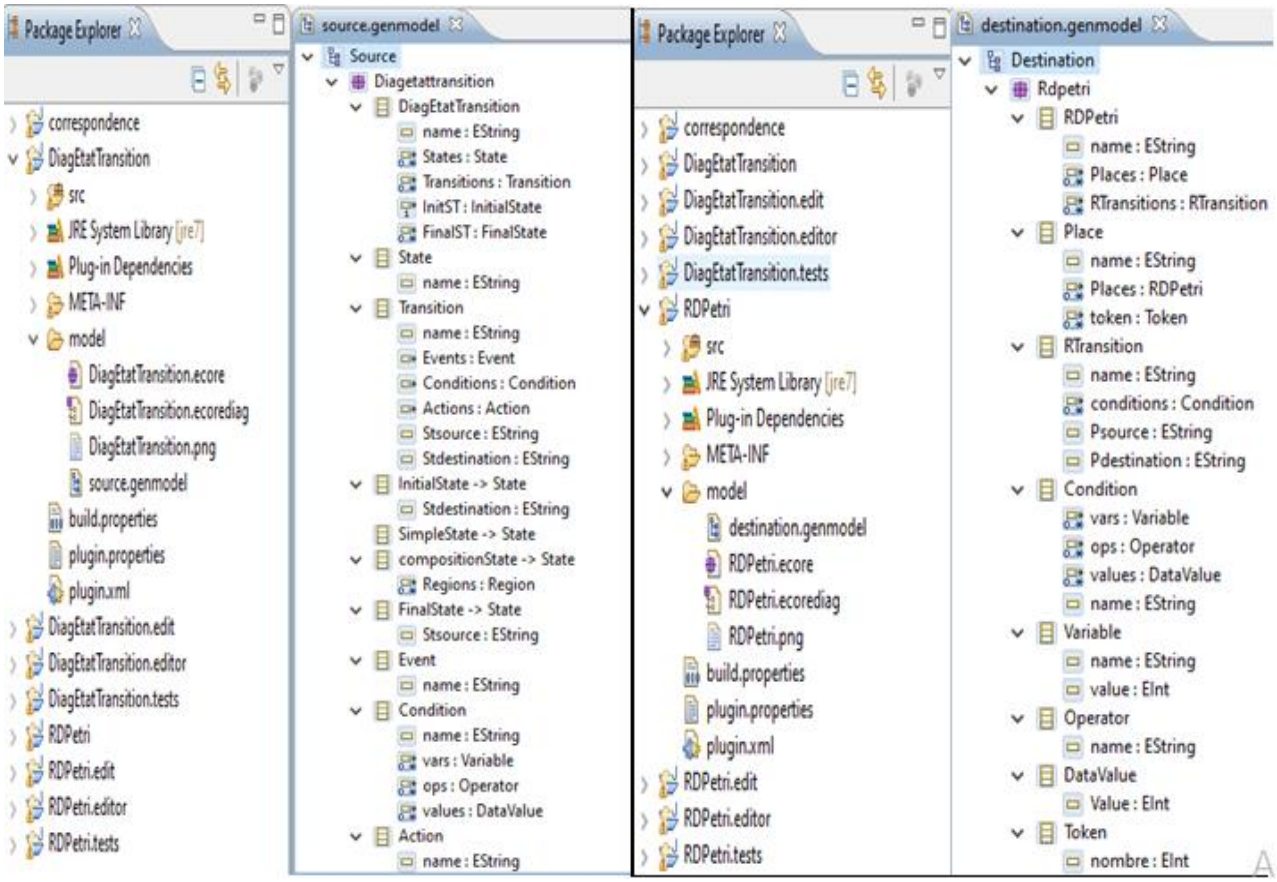


Figure 4.15 : Les éléments de source.genmodel / destination.genmodel

4.4. L'exécution :

Pour la phase d'exécution, une deuxième instance d'Eclipse doit être lancée, l'exécution dans TGG Interpreter se résume à crée une configuration du programme de transformation. Une fois ce dernier est créé, il est exécuté sur le modèle source afin d'obtenir le modèle cible et une trace de l'exécution des règles est génère

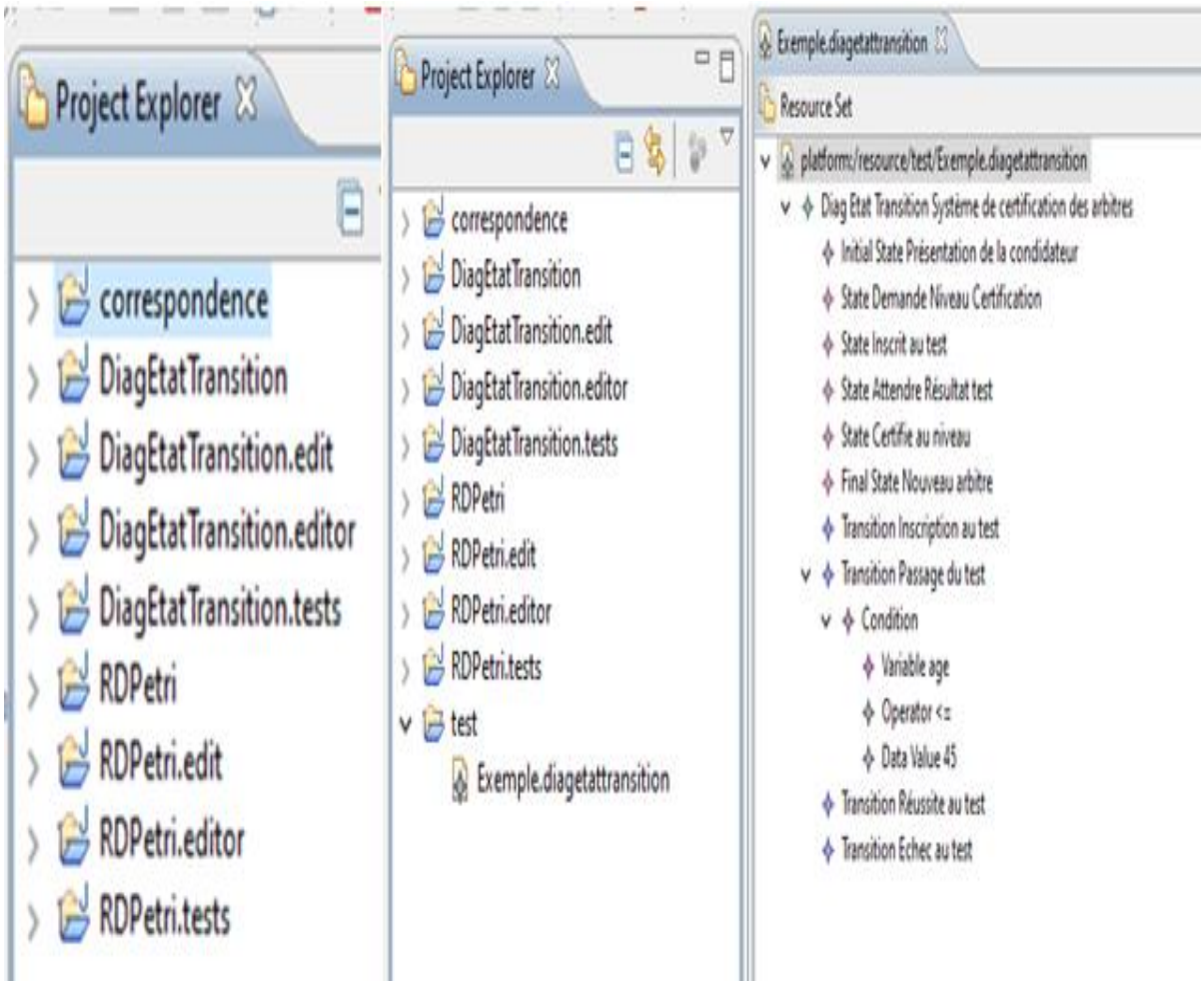


Figure 4.16 : Exemple de modèle de diagramme d'état transition

On va lancer une seconde instance d'Eclipse appliquée au projet source (Rtransition), puis on va créer un projet appelé (test) contenant le fichier (exemple1.Rtransition) pour réaliser un exemple du modèle source qui à son tour se transforme en modèle cible (voir figure 4.16), dans le dossier (test) on va créer un fichier (exemple1.interpreterconfiguration) avec lequel nous faisons le processus de transformation et une fenêtre de fin de transformation sera afficher (voir figure (4.17)

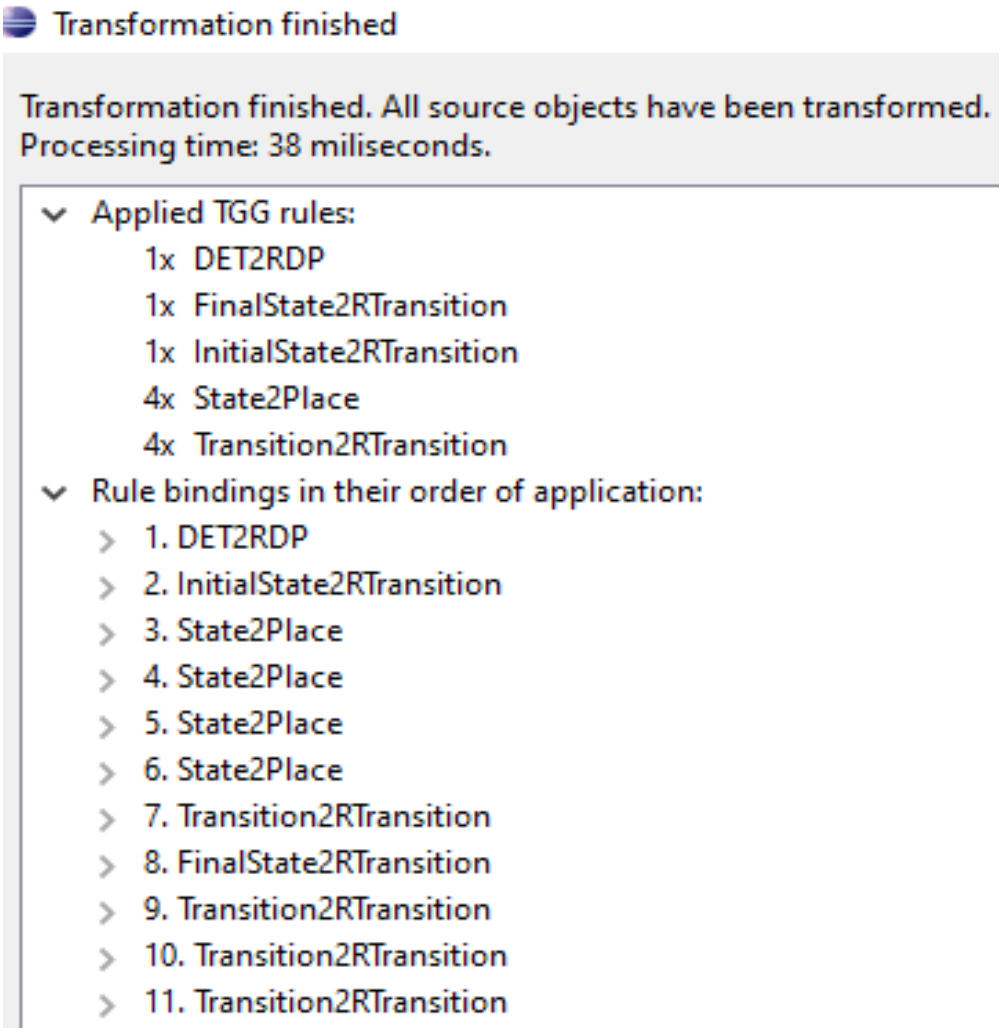


Figure 4.17 : Fin de transformation

Après avoir l'étape de fin de transformation, on va valider le fichier (exemple1.interpreterconfiguration) pour générer les fichiers (exemple1.xmi) et le fichier de modèle cible qui contient le résultat de transformation de modèle source (exemple1.RDP) (voir figure 4.18)

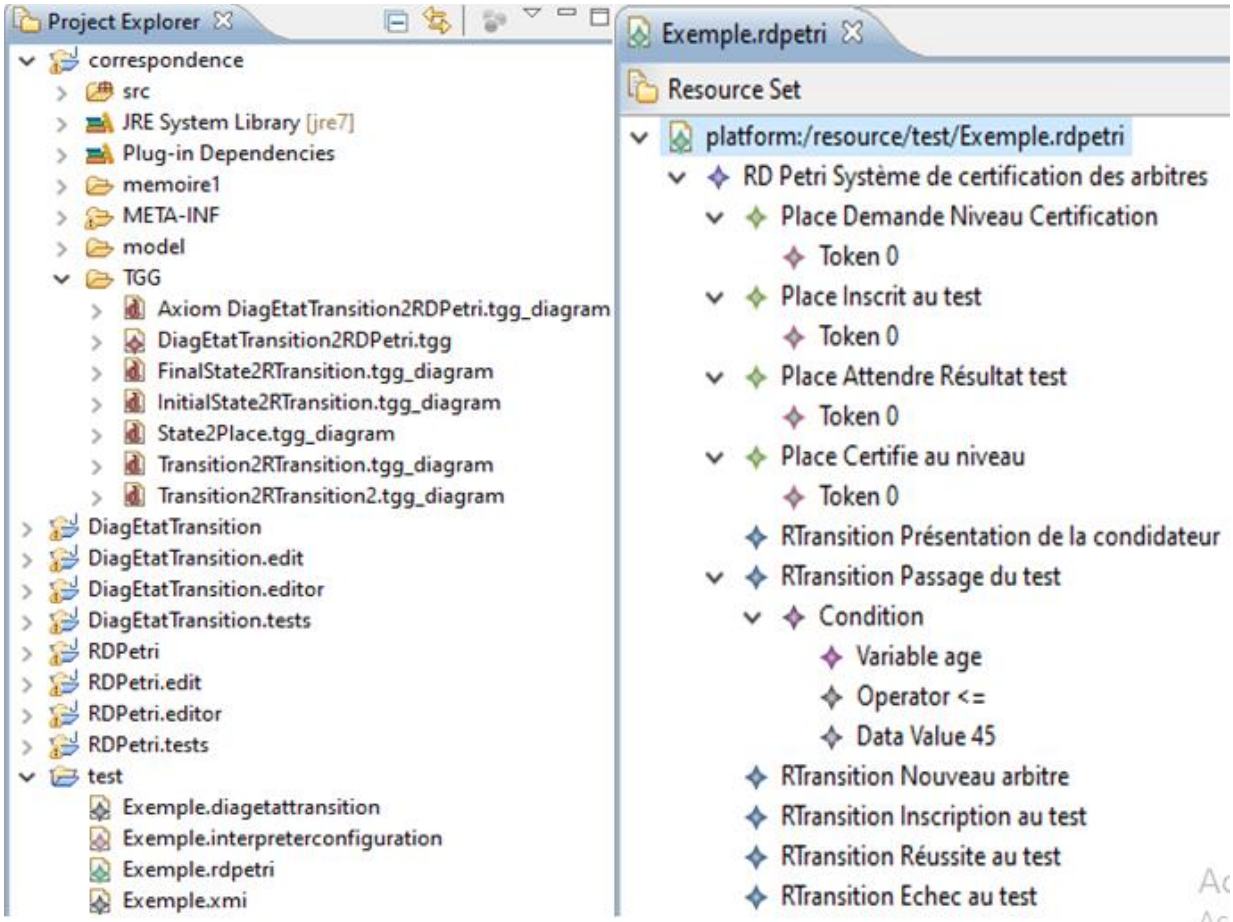


Figure 4.18 : Résultat de fin de transformation.

4.5 Génération de code :

La génération de code est l'étape du processus de compilation transformant l'arbre syntaxique abstrait enrichi d'informations sémantiques en code machine ou en bytecode spécialisé pour la plateforme cible. C'est l'avant-dernière étape du processus de compilation qui se situe avant l'édition des liens.

La deuxième transformation est une transformation Model-to-Text, qui prend un modèle RT-Maude et génère un code au format RT-Maude (inséré dans un module temporisé). Cette étape peut être accomplie en utilisant le Template Xpand.

4.5.1 Le langage de génération de code Xpand :

Le cadre de générateur Xpand fournit un langage textuels, qui sont utiles dans des contextes différents dans le processus de MDSD (par exemple, de validation, extensions de méta modèle, génération de code, la transformation du modèle).

Le langage Xpand est utilisé dans des modèles pour contrôler la génération de la sortie qui ce sont en code Java

Xpand est un langage spécialisé sur la génération de code à partir de modèles EMF c.-à-d. que on peut obtenir un code java à partir d'un model élaborer par Eclipse Mödling Framework (EMF) ;

Pour créer un projet Xpand on a besoin d'un modèle EMF, un modèle chek pour définit

Quelque contraintes qui a l'extension « **.**chk» et de trois packages essentielle ces packages Contiens des fichiers de différant extensions :

- le package méta model : contient un méta modèle de sortie (ex le méta modèle d'un langage ou d'un phrase simple…)
- le package Template : contient un fichier java qui a l'extension « .java », un fichier xpand qui a l'extension « **.**xpt »et un fichier xtend qui a l'extension « .ext »
- le package Workflow : contient le workflow qui permet d'appliquer le Template sur un modèle pour engendrer un ou plusieurs fichiers textes qui a l'extension « .mwe »

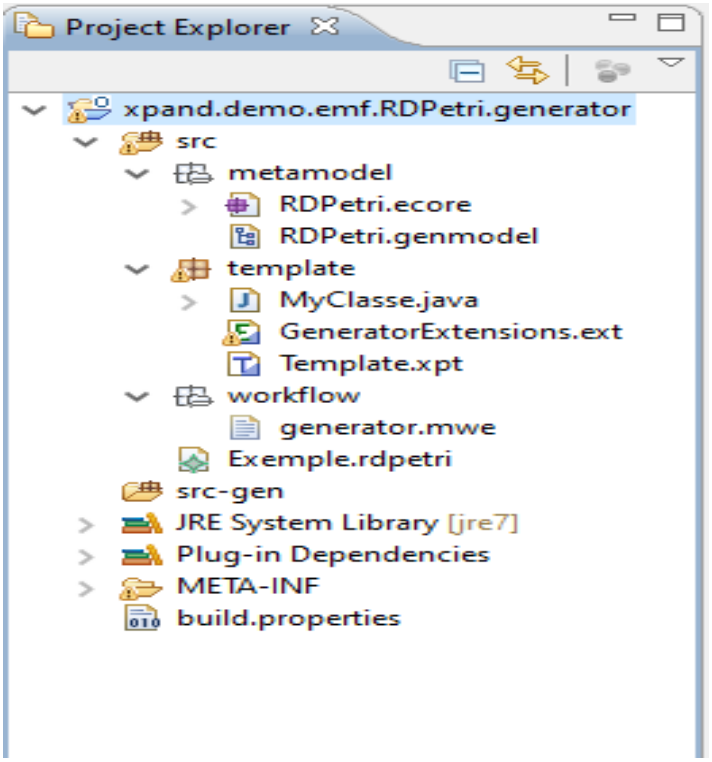


Figure 4.19 : les trois packagent (metamodel, template, workflow)

Le fichier Xpand permet le contrôle de la génération de code correspondant à un modèle, Le modèle doit être conforme à un méta-modèle donné. Le fichier d'extension « .xpt » se compose d'une ou plusieurs instructions IMPORT afin d'importer les méta-modèles, de zéro ou plusieurs EXTENSION avec le langage Xtend et d'un ou plusieurs blocks DEFINE. Les blocs DEFINE constituent le concept central du langage Xpand. La balise DEFINE se compose d'un nom, une liste optionnelle de paramètres et du nom de la méta-classe pour laquelle le Template est défini, Ils ont le format suivant dans la figure (4.20) :

*Template.xpt

```
«IMPORT rdpetri»
«IMPORT xpand2 »
«IMPORT emf »
«EXTENSION template::GeneratorExtensions»

"---------------------------------- main -----------------
«DEFINE main FOR RDPetri »
«file("Code")»
«EXPAND Element FOREACH eContents»
     «enregistrer("FIN)")»
«ENDDEFINE»
«DEFINE Element FOR EObject»
Not Defined: «metaType.name»
«ENDDEFINE»
«DEFINE Element FOR Place»
«enregistrer("    Place: "+this.name)»
«EXPAND Element FOREACH eContents»
«ENDDEFINE»
«DEFINE Element FOR Token»
      «enregistrer("       Jetons :"+this.nombre)»
«ENDDEFINE»
«DEFINE Element FOR RTransition»
      «enregistrer("     Transition: "+this.name)»
      «enregistrer("          place source: "+this.Psource)»
      «enregistrer("          place destination: "+this.Pdest
«EXPAND Element FOREACH eContents»
«ENDDEFINE»
«DEFINE Element FOR Condition»
      «enregistrer("          condition :")»
«EXPAND Element FOREACH eContents»
«ENDDEFINE»
«DEFINE Element FOR Variable»
      «enregistrer("          variable: "+this.name)»
«ENDDEFINE»

«DEFINE Element FOR Operator»
     «enregistrer("          operation: "+this.name)»
«ENDDEFINE»
«DEFINE Element FOR DataValue»
     «enregistrer("          valeur: "+this.Value)»
«ENDDEFINE»
```

Figure 4.20 : Le Template de génération de code

La prochaine étape de ce travail, nous voulons appeler des méthodes Java (Myclass.java) à partir d'une expression. Cela peut être fait en fournissant une extension Java via une classe Java (figure 4.21)

MyClasse.java

```
package template;
import java.io.*;
public class MyClasse {
    private static  String filename = "c://test/code.RDPetri";
    private static boolean existed = true;
    public static PrintWriter print1;


        public static void enregistrer(String aString) {
            try {
                 print1 = new PrintWriter( new BufferedWriter ( new FileWriter  (filename,existed)))
                 print1.println();
                 print1.println(aString);
                 print1.close();

            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();}
            System.out.println(aString);
          }


        public static void enregistrer1(String aString) {
            try {
                 print1 = new PrintWriter( new BufferedWriter ( new FileWriter  (filename,existed)))
                 print1.println("  "+aString);

                 print1.close();

            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();}
            System.out.println(aString);
          }
```

Figure 4.21 : Template Xpand pour la génération de code RDP (My class.java)

A la fin de cette étape, nous devrions obtenir une génération de code, qui sera applicable à tout modèle de sortie RDP de la réalisation de la transformation des règles TGG par l'interpréteur TGG de l'étape précédente. Une fois que nous aurons obtenu la spécification formelle de notre système temps réel dans le module temporisé RDP, le résultat dans la figure (4.22)

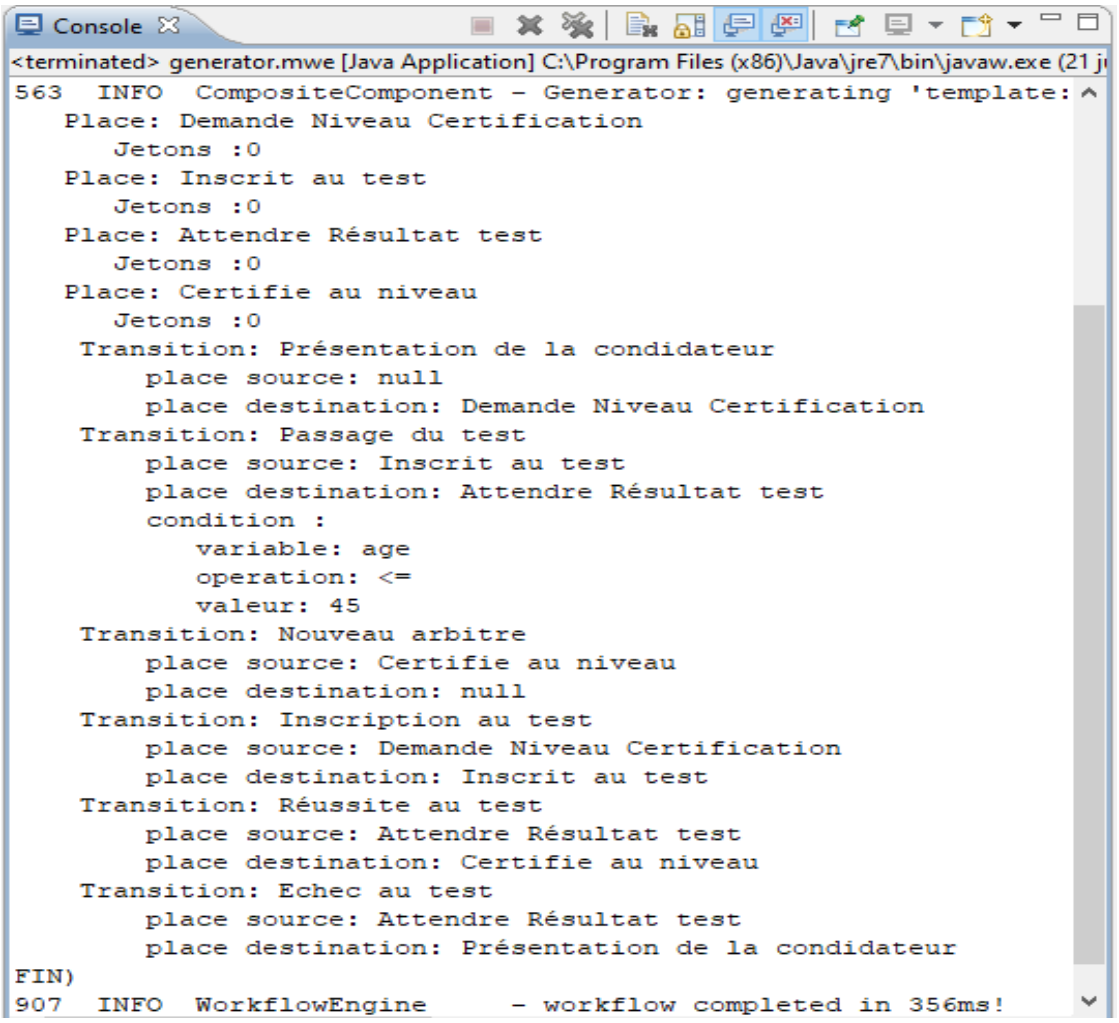


Figure 4.22 : résultat de la transformation M2T (generator)

4.6 Conclusion

Dans ce chapitre, Nous avons proposé une approche dirigée par les modèles pour la transformation de modèles basée sur l'utilisation des grammaires de graphes Nous avons réalisé une étude de cas de transformation de graphes en utilisant le TGG (Triple Graph Grammars) pour maitriser le passage des modèles de diagrammes d'états-transitions UML vers les Réseaux de Petri, l'implantation de cette approche nécessite l'utilisation d'environnements dédiés tels que la plateforme Eclipse Modeling Framework (EMF) qui nous a permis de mettre en œuvre les méta-modèles et les transformations. Notre travail s'est axé spécialement sur la spécification des transformations avec le formalisme TGG et leur implémentation avec l'outil TGG Interpreter intégré dans l'environnement EMF pour définir les règles de transformation TGG

Conclusion généraleet perspectives

A travers ce mémoire, notre objectif était de proposer une approche de transformation des diagrammes d'état transition vers les réseaux de Petri imbriqués Nous avons introduit dans le premier chapitre par une présentation des diagrammes d'état transition ; Le deuxième chapitre a été le sujet de détail de réseau de Petri. Les concepts de base et les principales propriétés des réseaux de Petri ordinaires ont été évoqués. Puis les réseaux de Petri imbriqués ont été présentés. Ces derniers sont des réseaux de Petri de haut niveau. Dans le troisième chapitre, nous avons présenté notre approche pour la transformation des diagrammes d'état transition vers les réseaux de Petri. Nous avons vu que cette approche se divise en trois étapes : la première consiste à proposer un métamodèle des diagrammes d'état transition. La deuxième étape consiste aussi à proposer un méta-modèle pour les réseaux de Petri. La troisième étape consiste à définir une grammaire de graphes qui permet de transformer un graphe source décrit avec le formalisme de diagramme d'état transition vers un graphe de réseau de Petri.

(IDM) il permet également L'ingénierie dirigée par des modèles développement logiciel, elle a apporté plusieurs avantages dans la production et la maintenance des systèmes. L'IDM n'est intéressante que si la transformation de modèles qui occupe une place très importante, est partiellement ou complètement automatisée dans le processus de développement ,Le travail présenté dans ce mémoire s'inscrit dans le domaine de l'ingénierie Dirigée par les modèles. Il se base essentiellement sur : La transformation automatique. Plus précisément, la méta-modélisation et transformation des diagrammes d'état transition, à l'aide d'TGG qui est intégré dans la plateforme Eclipse.

Comme perspectives, nous envisageons à l'avenir la réalisation des travaux suivants :
Nous avons pu à travers ce travail atteindre nos objectifs fixés, cependant un certain nombre d'améliorations peuvent être envisagés. Tout d'abord, il pourrait être envisageable de réaliser une implémentation de notre approche en utilisant d'autres outils de transformation de graphes tels que AGG et VIATRA2 dans un but de comparaison des performances. Aussi, Il serait possible d'étendre cette étude pour prendre en compte la transformation des autres diagrammes UML (diagramme de séquence, diagramme d'activité.).
Une perspective intéressante de ce travail consisterait en l'obtention d'étendre cette approche de spécification et de vérification sur d'autres types de systèmes. Finalement, la dernière perspective réside au niveau de l'approche graphique ou textuelle choisie pour la transformation de modèles. Pour répondre à cette problématique, il serait supportable d'étudier l'utilisation des autres approches de transformation de graphes présentées dans ce document et autres, pour en faire l'objet d'expérimentation dans le contexte de transformation de modèles.

Bibliographie

[1] Master recherche Systèmes Dynamiques et Signaux : la modélisation des systèmes de production par réseaux de petri Publié le 2011

[2] Salim ben Saoud chapitre 4 les réseaux de pitre (accédé le 17/1/2022) https://www.uvt.rnu.tn/resources-uvt/cours/Automatismes/chapitre4_rdp

[3] place transition et arcs. site web. (accédé le 17/1/2022) https://dept-info.labri.fr/~griffaul/Enseignement/MasterInfo-04-05/FDS/vademecum-petri

[4] Anouar Mahla coure de Réseaux de Petri site web (accédé le 17/1/2022) https://www.ensta-bretagne.fr/jaulin/gozone_minicours_anouar.pdf

[5] Guerrouf fayçal : Une Approche de Transformation des Diagrammes d'Activités d'UML Mobile 2.0 vers les Réseaux de Petri, mémoire de master Université el hadj Lakhdar – batna

[6] Les Réseaux de Petri Théorie, propriétés et applications Notes de Cours par N.Bennis site web

[7] Bendiaf messaoud, spécification et vérification des systèmes embarqués temps réel en utilisant la logique de réécriture thèses doctorat : Informatique 15/05/2018

[8] Stéphane Mariel cours de l'étudiant en réseaux de petri

[9] Chapitre 5 Diagramme d'états-transitions. site web : (accédé le 20/3/2022) https://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-etats-transitions#L5

[10] Diagramme d'état transition site web (accédé le 21/3/2022) https://www.lucidchart.com/pages/fr/diagramme-etats-transitions-uml

[11] Cours de Méthodes et Analyse objet - IUP MIAGE – 2003/2004 Chantal Reynaud Université Paris X - Nanterre UFR SEGMI – Maîtrise MIAGE

[12] uml5 site web. (accédé le 29/3/2022) http://projet.eu.org/pedago/sin/1ere/5diagramme_etats

[13] Khalifa Mansouri coure ingénierie système, université de Hassan II de Casablanca ENSET de Mohammedia

[14] Le langage UML : Les diagrammes d'états-transitions site web (accédé le 29/3/2022) https://www.coursehero.com/file/89878998/5Statechartpdf/

[15] Mouna Aouag diplôme de Docteur 3ème cycle LMD Des diagrammes UML 2.0 vers les diagrammes orientés aspect à l'aide de transformation de graphes Université Constantine 2 Soutenue le 09/10/2014

[16] Transformation de graphes site web https://123dok.net/document/nzwpn0qe-approche-specification-changements-besoins-basee-transformations-graphes.html

[17] Mecheri Nacera, Une Approche Hybride Pour Transformer Les Modèles. Thèse de Doctorat. Informatique. Université Ahmed Ben Bell soutenue le 27 octobre 2015

[18] Rafika Thabet, Ingénierie dirigée par les modèles d'un pilotage robuste de la prise en charge médicamenteuse Thèse de Doctorat l'université de Toulouse 23 octobre 2020
[19] Dimitris kolovos, Qu'est-ce que l'ingénierie pilotée par les modéles ? le 5 mars 2021 Site web https://codebots.com/app-development
[20] Transformation de modèles et Ingénierie Dirigées par les Modèles, site web (accédé le 8/4/2022) https://www.urbanisation-si.com/
[21] IDM site web (accédé le 8/4/2022)
https://www.modeliosoft.com/fr/technologies/uml.html
[22] IDM (logiciel) : présentation et fonctionnement technique, site web (accédé le 15/4/2022) https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445294-idm-logiciel-presentation-et-fonctionnement-technique/
[23] Eclipse Modeling Framework site web (accédé le 4/5/2022)
https://www.eclipse.org/modeling/emf/
[24] site web (accédé le 4/5/2022) https://www.encyclopedie.fr/definition/JDK
[25] Brahimi Khelaf, Génération d'une BDD-NoSQL à partir d'une BDD relationnelle Approche basée sur la transformation de graphe, mémoire de magister Université de BBA en 2020.