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

MEMOIRE

Présenté en vue de l'obtention du diplôme

Master en informatique

Spécialité : Ingénierie de l'informatique décisionnelle

THEME

La transformation automatique des diagrammes d'activité vers les réseaux de Petri

Présenté par :

BENZIOUCHE BELKACEM

BENMERIOUL HICHAM

Soutenu publiquement le : 20 /06/2023

Devant le jury composé de :

Président : Benmessahel ilyes

Examinateur : mossaoui boubakeur

Encadreur :bendiaf messaoud

2022/2023

Dédicace

Nous dédions humblement ce travail à nos chers parents, dont le soutien inconditionnel et la confiance indéfectible nous ont accompagnés au quotidien. Nous sommes incapables d'exprimer nos gratitudes par de simples mots. Que Dieu vous protège et veille sur vous pour nous. À nos précieux frères et sœurs, les mots ne suffisent pas à résumer notre reconnaissance et notre amour envers vous. À tous les membres de notre famille, nous vous remercions. À nos adorables amies, pour votre fidélité et votre soutien avec lesquels nous avons partagé des moments de joie et de bonheur. À tous nos enseignants, pour votre soutien, votre enseignement et vos conseils tout au long de notre parcours éducatif et professionnel.

Remerciement

Nous remercions d'abord et avant tout Allah qui nous a donné le courage et la patience pour réaliser ce travail. Un remerciement particulier à notre encadreur **BENDIAF MESSAOUD** pour sa présence, son aide et surtout pour ses précieux conseils qui nous ont assistés pour l'accomplissement de notre projet, Nous adressons également nos vifs remerciements aux membres du jury qui ont bien voulu et accepté d'examiner ce modeste travail, Á mes très chers frères, sœurs, ami (es) et à toutes les personnes qui ont participé de près ou de loin à la réalisation de ce projet. Enfin, je tiens à remercier toute la promotion 2022-2023 Master informatique. Ainsi que tous mes enseignants et les membres du département informatique d'Université Bordj Bou-Arreridj

Résumé

L'ingénierie basée sur les modèles joue un rôle très important dans le développement logiciel, où la transformation de modèle consiste à transformer le modèle source en un modèle cible basé sur les méta-modèles source et cible en résolvant divers problèmes (réutilisation, interopérabilité, migration de modèle). L'idée principale de notre travail est de créer un pont technologique entre l'atelier IDM et l'atelier Graph Grammar et d'effectuer une transformation bidirectionnelle à l'aide de l'outil d'interprétation TGG.

Notre travail consiste à transformation du diagramme d'activité vers les réseaux de Petri (RDP) basée sur la transformation des graphes. Notre approche consiste à proposer une grammaire de graphes contenant un ensemble de règles de transformation entre deux formalismes différents. Nous avons réalisé ce travail avec l'outil Eclipse en définissant deux métamodèles (un pour le diagramme d'activité et un pour le réseau de Petri) et une grammaire des graphes.

Enfin, nous concluons notre contribution par une étude de cas bien illustrée et présentons les résultats qui complètent notre approche.

Mots-clés : Les triples grammaires de graphes, Modèle, transformation de modèles, Transformation de graphe, grammaires, méta-modèles, diagramme d'activité, [(RDP) les réseaux de petri], [(TGG) triple graph grammar]

Abstract

Model-driven engineering plays a crucial role in software development, where model transformation involves converting a source model into a target model based on source and target metamodels, addressing various issues such as reuse, interoperability, and model migration. The main idea of our work is to create a technological bridge between the model-driven engineering workshop and the Graph Grammar workshop, enabling bidirectional transformation using the TGG interpretation tool.

Our thesis presents the transformation from activity diagrams to Petri nets based on graph transformation. Our approach involves proposing a graph grammar that encompasses a set of transformation rules between two different formalisms. We implemented this work using the Eclipse tool, defining two metamodels (one for activity diagrams and one for Petri nets) and a graph grammar.

Finally, we conclude our contribution with a well-illustrated case study and present the results that validate our approach.

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

ملخص

تلعب الهندسة التي تعتمد على النموذج دو ًرا مه ًما في تطوير البرامج، حيث يتضمن تحويل النموذج تحويل نموذج المصدر إلى نموذج مستهدف يعتمد على النماذج الوصفية المصدر والهدف، ومعالجة العديد من القضايا مثل إعادة االستخدام والتشغيل البيني وترحيل النموذج. تتمثل الفكرة الرئيسية لعملنا في إنشاء جسر تقني بين ورشة العمل الهندسية القائمة على النموذج وورشة عمل GRAPH GRAMMAR مما يتيح التحول ثنائي االتجاه باستخدام أداة تفسير TGG

تقدم أطروحتنا التحول من مخططات النشاط إلى شبكات بتري بنا ًء على تحويل الرسم البياني. يتضمن نهجنا اقتراح قواعد البياني التي تشمل مجموعة من قواعد التحويل بين شكلين مختلفين. قمنا بتنفيذ هذا العمل باستخدام أداة.Eclipseالرسم

حيث حددنا نموذجين أحدهما لمخططات النشاط واآلخر لشبكات بتري أخي ًرا، نختتم مساهمتنا بدراسة حالة موضحة جيدًا ونقدم النتائج التي تثبت صحة نهجنا

الكلمات الرئيسية: القواعد النحوية للرسم البياني الثالثي، النموذج، تحويل النموذج، تحويل الرسم البياني، القواعد النحوية، [قواعد الرسم البياني الثالثية (TGG)]، [شبكات بتري (RDP)]النماذج الوصفية، مخطط النشاط،

Table des matières

Liste des abréviations

UML : Unified Modeling Language.
RDP : Réseau de petri.
OPN : Oracle Partner Network.
IDM : Ingénierie Dirigée par les Modèles.
OMG: Object Management Group MOF: Meta-Object Facility.
MDA: Model Driven Architecture.
OCL: Object Constraint Language.
IBM: International Business Machines
XML: Extensible Markup Language.
CIM: Computation Independent Model.
PIM: Platform Independent Model.
PDM: Platform Description Model
PSM: Platform Specific Model
M2T: Model to Text.
M2M: Model to Model.
LHS: left Hand Side
RHS: Right Hand Side
TGG : Triple Graph Grammar
VPM : Volume Plaquettaire Moyen
EMF: Eclipse Modeling Framework.
SysML: Système Modeling Language
GMF: Graphical Modeling Framework
ATL: Atlas Transforming Language
JDK : Java Développement Kit
XMI : XML Metadata Interchange
JDT : Java Development Tooling
J2SE : Java 2 Standard Edition
J2EE : Java 2 Enterprise Edition
JME : Java Micro Edition QVT : (Query/View/Transformation)

Liste des figures

Liste des tableaux

Introduction Générale

Contexte :

La transformation de modèles joue un rôle fondamental dans l'ingénierie dirigée par les modèles (IDM). Elle peut être réalisée de manière manuelle ou automatisée, mais dans ce dernier cas, elle exige du développeur qui la conçoit une maîtrise des méta-modèles impliqués dans la transformation. Les modèles sont considérés comme des éléments essentiels dans ce processus. L'adage populaire de l'IDM, "Modéliser une fois, générer partout", prend tout son sens ici. La transformation de modèles consiste à convertir un ensemble de modèles d'une application donnée en d'autres modèles de la même application. Elle repose sur un ensemble de règles permettant de passer d'un modèle source conforme à un méta-modèle source à un modèle cible conforme à un méta-modèle cible.

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'activité 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.

Méthodologie et résultat

Dans ce travail, nous proposons une approche de transformation des diagrammes d'activité vers les réseaux de Pétri. L'approche que nous définissons vise à proposer:

1. Une modélisation des system de production et des réseaux de Pétri.
2. Une grammaire de graphe pour appliquer la transformation des diagrammes d'activité vers des réseaux de Petri.

La construction d'un outil de modélisation à partir de zéro est une tâche prohibitive. Les approches de méta-modélisations sont utiles pour faire face à ce problème, car elles permettent la modélisation des formalismes eux-mêmes. Un modèle de formalisme qui contient suffisamment d'informations permet la génération automatique d'un diagramme d'activité pour

construire des modèles conforme à la syntaxe du formalisme décrit. Pour cela, nous utilisons éclipse.

Structure du rapport

Á la suite de ce préambule, notre mémoire se divise principalement en quatre chapitres.

Le premier chapitre : nous présentons (Ingénieries dirigée par les modèles (IDM)) parles sur les méta-modélisations et les Notions générales de l'IDM, les Architecture et les phrase de modélisation ….

Le deuxième chapitre : nous présentons Les diagrammes d'activité (Présentation de diagramme d'activité, Les différents type, Les concepts avancés, Les activité…)

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 : (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 01 : L'ingénierie dirigée par les modèles

1.1 Introduction :

L'ingénierie dirigée par les modèles (IDM) est une approche de développement logiciel qui met l'accent sur l'utilisation de modèles comme élément central du processus de conception, de développement et de maintenance des systèmes logiciels et matériels. L'IDM repose sur le principe fondamental selon lequel les modèles peuvent être utilisés comme une représentation abstraite des systèmes, capturant leurs aspects structurels, comportementaux et fonctionnels.
Dans ce chapitre nous expliquons en détail les concepts, les techniques et les outils associés à l'Ingénierie dirigée par les modèles, Nous présentons par la suite la transformation des modèles et enfin nous expliquons la transformation de graphe.

1.2 L'ingénierie dirigée par les modèles :

L'ingénierie dirigée par les modèles (Model-Driven Engineering en anglais) est une approche de développement logiciel qui met l'accent sur l'utilisation de modèles abstraits pour concevoir, analyser et générer des systèmes logiciels. Elle se concentre sur la création de modèles représentant différentes perspectives d'un système et utilise ces modèles pour automatiser les processus de développement[1].

1.3 Concepts de base de l'ingénierie dirigée par les modèles

Voici quelques notions générales liées à l'IDM[1] :

1.3.1 Modèle

Un modèle est une représentation abstraite d'un système, d'un processus ou d'un concept. Il capture les aspects essentiels du système et fournit une vue simplifiée mais précise de la réalité. Les modèles peuvent être utilisés pour décrire l'architecture, le comportement, les fonctionnalités et les contraintes d'un système.

1.3.2 Méta-modèle et métamodélisation :

La méta-modélisation est une activité implique de définir le méta-modèle d'un langage de modélisation, ce qui comprend la création des méta-modèles ainsi que la définition de la sémantique du langage. Elle nécessite également la mise en place d'analyseurs, de compilateurs, de générateurs de code et d'autres outils exploitant les méta-modèles. Ainsi, la méta-modélisation englobe l'analyse, la construction et le développement d'un ensemble de cadres,

de règles, de contraintes, de modèles et de théories qui sont applicables et utiles pour modéliser une classe spécifique de problèmes. Ce concept s'applique dans les domaines du génie logiciel et du génie système en utilisant les notions de méta-modèle et de modélisation. Il existe différents types de méta-modèles avec diverses applications.

1.3.3 Transformation du modèle :

La transformation du modèle consiste à convertir une représentation en une autre. Elle permet de passer d'un niveau d'abstraction à un autre ou d'effectuer des manipulations sur les modèles pour obtenir des résultats spécifiques. Les transformations de modèle peuvent être effectuées manuellement ou automatiquement à l'aide d'outils.

1.3.4 Méta-métamodèle :

L'Ingénierie Dirigée par les Modèles (IDM) adopte le principe selon lequel "tout est modèle". Ainsi, un méta-métamodèle est utilisé pour décrire un modèle de méta-modèles. Il s'agit essentiellement d'un langage de description des méta-modèles, qui permet d'exprimer les règles de conformité qui relient les entités du niveau modèle à celles du niveau méta-modèle. Le méta-métamodèle est conçu de manière à posséder la propriété de méta-circularité, c'est-à-dire la capacité de se décrire lui-même.

1.4 L’architecture dirigée par les modèles

L'ingénierie dirigée par les modèles est un domaine qui repose sur l'utilisation de modèles et de technologies associées pour manipuler ces modèles. Dans le processus de développement d'un système, il existe différentes méthodes d'utilisation des modèles. L'approche la plus avancée et couramment utilisée est celle de l'architecture dirigée par les modèles.

1.4.1 Définition

L'Object Management Group (OMG) a défini l’architecture dirigée par les modèles en quatre niveaux : [2]

Niveau M0 : C'est la couche la plus basse qui représente les données concrètes du monde réel. Les éléments de cette couche sont des instances de la couche supérieure, qui est le niveau M1.

Niveau M1 : C'est le niveau des modèles où chaque élément du monde réel est représenté par un modèle. Les modèles de cette couche sont décrits à l'aide d'un métamodèle de la couche M2.

Niveau M2 : Dans ce niveau, les métamodèles sont utilisés pour décrire les langages de

modélisation.

Niveau M3 : C'est le niveau le plus élevé, le niveau du méta-métamodèle, qui permet de définir la structure des métamodèles du niveau M2. Étant donné que le méta-métamodèle est auto-descriptif, il permet de décrire sa propre sémantique. (**La figure 1.1**) illustre les quatre niveaux de l'architecture de métamodélisation.

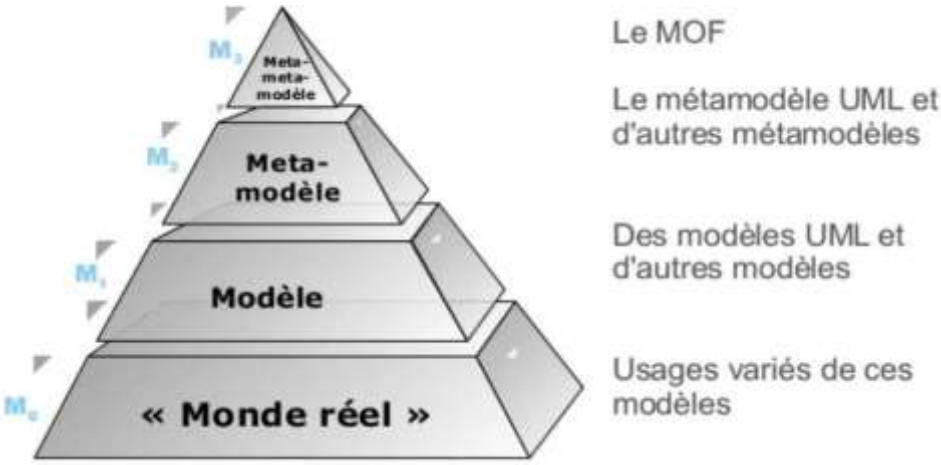


Figure 0-1 : Architecture à quatre niveaux.

1.4.2 Les standards fondamentaux de L'OMG :

OMG a défini plusieurs normes pour le MDA, nous pouvons citons :

1.4.2.1 MOF (Meta-Object Facility) :

MOF est un standard qui fournit un cadre pour la création de méta-modèles et de langages de modélisation. Il définit une infrastructure pour la création, le stockage et l'échange de modèles conformes à un méta-modèle spécifié**[3]**.

1.4.2.2 UML (Unified Modeling Language) :

UML (Unified Modeling Language) : UML est un langage de modélisation visuel qui offre une notation graphique pour spécifier, visualiser, concevoir et documenter les systèmes logiciels. L'OMG a standardisé UML et l'a adopté comme langage de modélisation de référence pour le MDA**[3]**.

1.4.2.3 OCL (Object Constraint Language) :

L'Object Constraint Language (OCL) est un langage de contraintes déclaratives qui fait partie des standards de l'OMG (Object Management Group). Il est spécialement conçu pour être utilisé avec des langages de modélisation tels que l'UML (Unified Modeling Language) et d'autres langages basés sur le méta-modèle MOF (Meta-Object Facility). L'OCL permet de spécifier des contraintes, des invariants et des expressions sur les modèles, les classes, les attributs et les opérations. **[3]**

1.4.2.4 Xml Meta data Interchange) :

XMI (XML Metadata Interchange) : XMI est un standard qui définit un format d'échange basé sur XML pour les modèles MOF. Il permet l'échange de modèles entre différents outils et environnements de modélisation**[3]**.

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

Le principe de base de l'IDM est le développement de différents modèles. Partant d'un modèle d'affaires où il n'y a pas de considérations informatiques (Computation Independent Model, CIM), en passant par la transformation en un modèle d'analyse et de conception (Platform Independent Model, PIM) et enfin la transformation de ce dernier en un modèle de code (Platform Specific Model)., MSP).) pour une implémentation de système spécifique**[2]**.

1.5.1 Le CIM (Computation Independent Model) :

C'est le modèle de base pour analyser un domaine ou une application métier. Ce modèle est indépendant de tout système informatique, il décrit les concepts commerciaux, le savoir-faire, les processus, la terminologie et les principes de gestion (supérieurs) et décrit également la situation dans laquelle le système est utilisé. C'est un modèle de très longue durée qui ne change qu'au fur et à mesure que les connaissances ou les besoins de l'entreprise évoluent**[2]**.

1.5.2 Le PIM (Platform Independent Model) :

C'est un modèle de conception qui décrit un système informatique quelles que soient la plate-forme technique et la technologie utilisées pour mettre en œuvre l'application. Représente la logique métier native (action des entités et des services) **[2]**.

1.5.3 Le PDM (Platform Description Model) :

En tant que modèle de conception, il décrit le système informatique de manière indépendante de toute plate-forme technique ou technologie utilisée pour déployer l'application.

Il représente la logique métier spécifique au système**[2]**.

1.5.4 Le PSM (Platform Specific Model):

Il est utilisé pour générer du code exécutable pour des plates-formes techniques spécifiques. De manière générale, on peut distinguer quatre catégories d'outils.

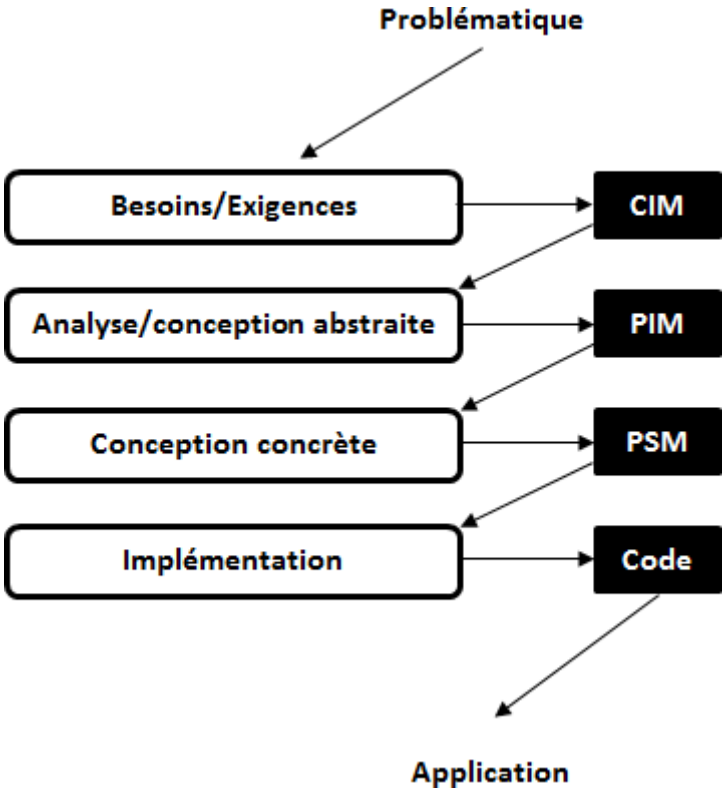


Figure 0-2 : phases de modélisation.

1.6 La transformation de modèle :

Est un processus clé dans l'approche MDA (Model-Driven Architecture) qui consiste à convertir un modèle dans un langage ou une représentation abstraite en un autre modèle dans un langage ou une représentation plus concrète. Cette transformation permet de passer d'un niveau d'abstraction à un niveau plus détaillé ou spécifique, tout en préservant la sémantique et les informations essentielles du modèle d'origine.

1.6.1 Principe général d'une transformation :

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.

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 :

- **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**[20]**.

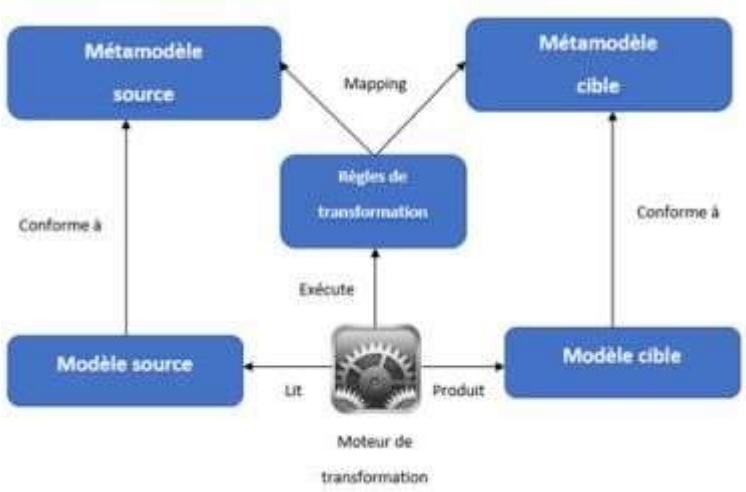


Figure 0-3 : Principes générale d'une transformation.

1.6.2 Types de transformation :

Une transformation de modèles et en correspondances des éléments des modèles cibles et sources. On distingue les types de transformation suivant **[6]**:

1.6.2.1 Les transformations simples (1 vers 1) :

Les transformations simples, également connues sous le nom de transformations 1 vers 1, sont des transformations de modèles dans lesquelles un élément du modèle source est directement converti en un élément équivalent dans le modèle cible, sans modification ou manipulation supplémentaire. Ces transformations sont généralement utilisées pour des conversions directes et basiques entre deux modèles.

1.6.2.2 Les transformations multiples (M vers 1) :

Les transformations multiples, également connues sous le nom de transformations M vers 1, sont des transformations de modèles dans lesquelles plusieurs éléments du modèle source sont combinés pour générer un seul élément dans le modèle cible. Dans ce type de transformation, plusieurs éléments du modèle source contribuent à la création d'un élément unique ou agrégé dans le modèle cible.

1.6.2.3 Les transformations de mis à jour :

Les transformations de mise à jour, également appelées transformations de modification ou transformations 1 vers M, sont des transformations de modèle dans lesquelles un élément du modèle source est utilisé pour mettre à jour ou modifier plusieurs éléments dans le modèle cible. Dans ce type de transformation, un élément du modèle source est utilisé comme source de données pour mettre à jour plusieurs éléments cibles dans le modèle cible.

1.6.2.4 Transformations endogènes :

Les transformations endogènes sont des transformations de modèle qui agissent sur un modèle lui-même, c'est-à-dire qu'elles modifient le modèle source pendant le processus de transformation. Ces transformations sont également appelées "transformations auto-modifiantes" ou "transformations in-place".

1.6.2.5 Transformations exogènes :

Les transformations exogènes sont des transformations de modèle qui produisent un nouveau modèle cible distinct du modèle source, sans modifier le modèle source lui-même. Ces transformations sont également connues sous le nom de "transformations sans effets latéraux" ou "transformations non-destructives".

1.6.2.6 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

1.7 Approches de transformation de modèles :

Prise en compte de divers critères de classification, tels que spécifications, règles transformation, relation source-destination, incrément, traçabilité, etc. identifie plusieurs approches de transformation dont les principales catégories sont présentées dans la figure 3 ci-dessous. Au niveau de classification le plus élevé, il existe deux approches de transformation : les transformations "modèle" et les transformations "modèle à modèle" modèle à modèle. Code source » (génération de code). Dans chacune de ces deux catégories, il existe sous-catégories différentes, comme le montre la figure 5-1.

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

Les approches de modèle au code, également connues sous le nom de Model-to-Text (M2T), sont des approches de transformation de modèles qui se concentrent sur la génération automatique de code source à partir de modèles. L'objectif principal de ces approches est de produire du code exécutable à partir d'un modèle abstrait ou d'un modèle de plus haut niveau d'abstraction.

Voici quelques caractéristiques clés des approches de modèle au code (M2T) :

- **Modèle abstrait :** Les approches M2T utilisent un modèle abstrait qui représente le système ou l'application à développer. Ce modèle peut être indépendant de la plate-forme (CIM) ou un modèle de plus haut niveau d'abstraction (PIM) qui capture les concepts et les règles métier.
- **Règles de génération de code :** Les approches M2T utilisent des règles de transformation qui définissent comment les éléments du modèle sont traduits en code source. Ces règles décrivent les correspondances entre les éléments du modèle et les constructions du langage de programmation cible, ainsi que les opérations et les transformations à appliquer.
- **Template de code :** Les approches M2T utilisent souvent des Template.

1.7.2 Les approches de modèle au modèle (Model To Model M2M) :

Ces transformations ont beaucoup évolué depuis l'introduction du MDA. Ce type de transformation permet de générer plusieurs modèles intermédiaires avant d'atteindre le modèle de code afin d'explorer différentes vues du système, de l'optimiser, de vérifier ses propriétés et de le valider.

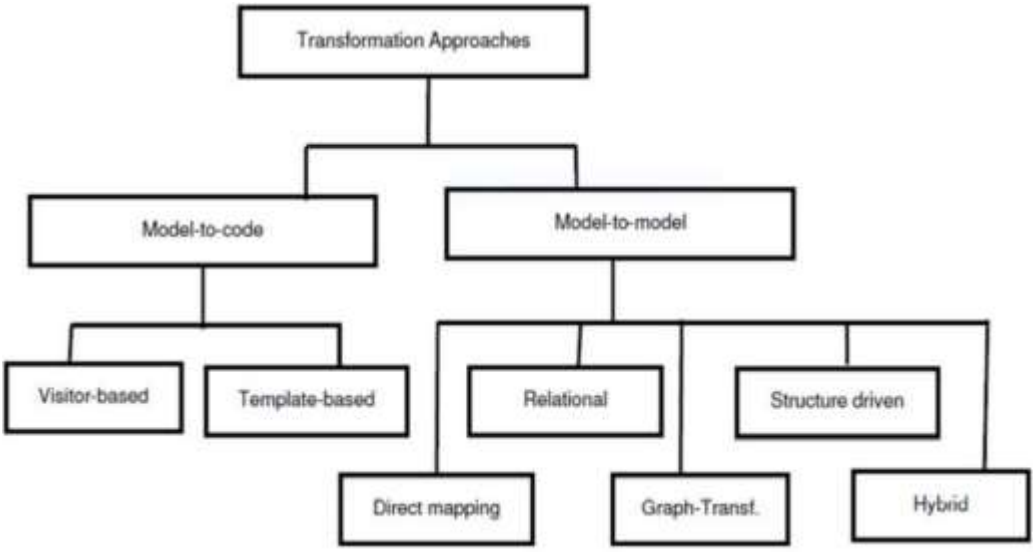


Figure 0-4 : Les approches de transformation de modèles.

1.8 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[4].

1.8.1 Principe de transformation de graphe :

Les méthodes traditionnelles de paraphrase, telles que la grammaire de Chomsky ou la paraphrase des termes, peuvent manquer de clarté. C'est pourquoi des transformations graphiques ont été développées pour résoudre ce problème. Le processus de transformation graphique consiste en l'application itérative de règles à un graphe. Chaque fois qu'une règle est appliquée, une partie du graphe est remplacée par une autre partie selon la définition de la règle. Voici comment fonctionne le mécanisme de transformation :

1. Sélectionnez la règle appropriée parmi un ensemble de règles.
2. Appliquez cette règle au graphe d'entrée.
3. Recherchez d'autres règles applicables (itération) jusqu'à ce qu'il n'y ait plus de règles pouvant être appliquées.

Ce processus repose sur un ensemble de règles qui suivent une grammaire spécifique, connue sous le nom de modèle de grammaire des graphes.

Les outils de transformations de graphes :

- **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) **[5]**.

- **GReTL (Graph Transformation Language) :** GReTL est un langage de transformation de graphes développé spécifiquement pour la manipulation et la transformation de graphes. Il fournit des primitives pour décrire les règles de transformation et les opérations sur les graphes, ainsi que des mécanismes pour spécifier des motifs de recherche et des contraintes.
- **AGG (Attributed Graph Grammar) :** AGG est un environnement de développement qui prend en charge la spécification et la manipulation de graphes ainsi que la transformation de graphes basée sur des grammaires de graphes attribués. Il permet de définir des règles de transformation et de les appliquer à des graphes pour effectuer des modifications et des raffinements.
- **Henshin :** Henshin est un outil de transformation de graphes développé dans le cadre du projet Eclipse Modeling Framework (EMF). Il permet de spécifier des règles de transformation graphique à l'aide d'un langage de transformation déclaratif et d'exécuter ces règles pour transformer des modèles basés sur EMF.
- **ATL (Atlas Transformation Language) :** ATL est un langage de transformation de modèles qui prend également en charge la transformation de graphes. Il permet de définir des règles de transformation pour effectuer des manipulations et des raffinements sur des graphes de modèles basés sur le langage MOF (Meta-Object Facility).
- **Neo4j :** Neo4j est une base de données orientée graphe qui offre des fonctionnalités de manipulation et de transformation de graphes. Il permet de stocker, de requêter et de transformer des données structurées sous forme de graphes en utilisant le langage de requête de graphes Cypher.

1.9 Conclusion :

Dans ce chapitre, nous avons essayé d'introduire le concept de transformation de modèle, qui est l'un des éléments centraux de l'approche MDA, car il permet d'automatiser et/ou

de soutenir les activités qui composent les modèles de développement déjà dans les phases de pré-développement, à travers des tests et la production de code. Dans ce contexte, nous avons essayé de présenter différentes approches existantes dans la littérature, en commençant par une brève introduction aux approches IDM et MDA, puis nous présentons une énumération des différents types de transformations, puis une classification des différentes approches, et enfin nous présentons un cadre spécifique de transformation de modèle basé sur les transformations de graphe.

Chapitre 02 : diagramme d'activité

2.1 Introduction

Dans ce chapitre, nous explorerons en détail les diagrammes d'activité, un outil essentiel dans la modélisation et la représentation graphique des processus métier.

Les diagrammes d'activité sont largement utilisés dans le domaine de l'ingénierie logicielle et de la gestion de projet pour décrire visuellement le flux des activités, les décisions prises et les étapes suivies dans un processus donné. Ils offrent une vue d'ensemble claire et structurée des différentes étapes et interactions, ce qui facilite la communication et la compréhension entre les membres d'une équipe.

2.2 Définition du diagramme d'activité

Les diagrammes d'activité sont des représentations graphiques utilisées dans le cadre du langage de modélisation unifié (UML) pour décrire visuellement le comportement d'un système, d'un processus ou d'un workflow. Ils permettent de représenter les activités, les actions, les décisions et les flux de contrôle entre les différentes étapes d'un processus**[7]**.

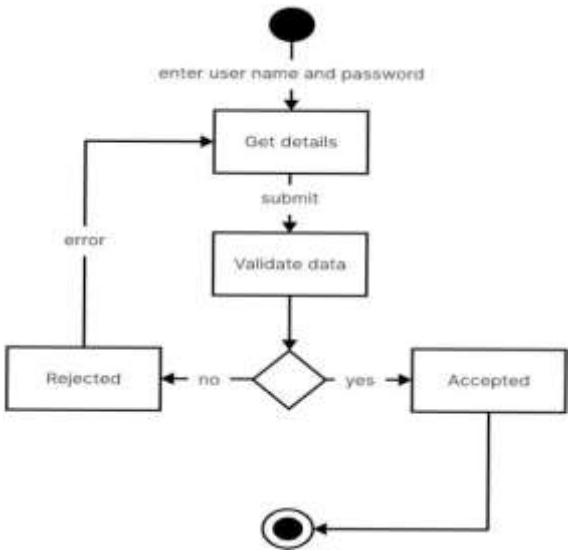


Figure 0-1 Exemple de diagramme d'activités

2.3 Intérêts des diagrammes d'activités

Les diagrammes d'activités présentent plusieurs intérêts dans le domaine de l'ingénierie logicielle et de la modélisation des processus. Voici quelques-uns des principaux avantages**[8]** :

- **Visualisation du flux des activités :** Les diagrammes d'activités offrent une représentation visuelle claire et intuitive du flux des activités d'un processus. Cela permet de comprendre facilement l'ordre séquentiel des actions et les dépendances entre celles-ci.
- **Communication efficace :** Les diagrammes d'activités servent de langage commun entre les différentes parties prenantes d'un projet. Ils permettent aux développeurs, aux analystes métier et aux utilisateurs finaux de visualiser et de discuter du comportement attendu d'un système ou d'un processus. Cela facilite la communication et réduit les risques d'interprétation erronée.
- **Analyse et amélioration des processus :** Les diagrammes d'activités aident à analyser les processus métier existants et à identifier les points forts et les points faibles. En visualisant le flux des activités, il devient plus facile de repérer les redondances, les goulots d'étranglement, les inefficacités ou les problèmes potentiels. Cela permet ensuite de proposer des améliorations et des optimisations pour rendre les processus plus efficaces.
- **Modélisation des comportements complexes :** Les diagrammes d'activités permettent de modéliser des comportements complexes en utilisant des actions parallèles, des branchements conditionnels et des boucles. Cela facilite la représentation de scénarios complexes et la gestion des différentes trajectoires d'exécution possibles
- **Documentation et maintenance :** Les diagrammes d'activités servent de documentation pour décrire le comportement d'un système ou d'un processus. Ils sont utiles pour la formation des utilisateurs, la documentation des fonctionnalités et la maintenance future. Ils facilitent également la compréhension du système par les nouveaux membres de l'équipe ou les intervenants externes ...
- **Intégration avec d'autres diagrammes :** Les diagrammes d'activités peuvent être intégrés avec d'autres types de diagrammes, tels que les diagrammes de classes, les diagrammes de séquence ou les diagrammes de déploiement. Cela permet une vue d'ensemble cohérente du système et favorise la compréhension globale de son fonctionnement.

2.4 Domaines d'application des diagrammes d'activité

Les diagrammes d'activité sont des outils de modélisation polyvalents largement utilisés dans divers domaines. Voici quelques-uns des domaines d'application courants des diagrammes d'activité**[9]** :

1. **Développement logiciel :** Les diagrammes d'activité sont utilisés pour modéliser le flux de contrôle et les comportements des systèmes logiciels. Ils aident à spécifier les étapes et les actions nécessaires pour réaliser une fonctionnalité ou un processus logiciel.
2. **Analyse et conception des processus métier :** Les diagrammes d'activité sont utilisés pour modéliser les flux de travail et les étapes d'un processus métier. Ils permettent de visualiser et d'analyser les activités impliquées, d'identifier les goulots d'étranglement et d'optimiser les processus.
3. **Ingénierie des systèmes :** Les diagrammes d'activité sont utilisés pour modéliser le comportement des systèmes complexes, y compris les interactions entre les composants, les états du système et les flux d'informations. Ils aident à comprendre et à spécifier le comportement global du système.
4. **Analyse des exigences** : Les diagrammes d'activité sont utilisés pour capturer les exigences fonctionnelles d'un système en modélisant les scénarios d'utilisation et les comportements attendus. Ils aident à clarifier les besoins des utilisateurs et à communiquer efficacement avec les parties prenantes.
5. **Modélisation des processus décisionnels :** Les diagrammes d'activité sont utilisés pour représenter les processus de prise de décision complexes. Ils permettent de modéliser les différentes options, les conditions et les décisions intermédiaires pour atteindre un résultat souhaité.
6. **Simulation et analyse des performances :** Les diagrammes d'activité peuvent être utilisés pour simuler et analyser les performances des systèmes. En modélisant les actions et les flux d'activité, il est possible d'évaluer les temps d'exécution, les charges de travail et les goulots d'étranglement potentiels.
7. **Formation et documentation :** Les diagrammes d'activité sont utilisés comme outil pédagogique pour enseigner les concepts de modélisation et de flux de contrôle. Ils servent également de documentation visuelle pour décrire les processus et les comportements des systèmes.

2.5 Composants de base d’un diagramme d'activités

Dans un Diagramme d'activité, il existe différents types de nœuds qui sont utilisés pour

représenter des éléments spécifiques. Voici quelques types de nœuds couramment utilisés dans les diagrammes d'activité[9] :

2.5.1 Nœud initial (Initial Node)

Il représente le point de départ du flux d'activités dans le diagramme. Il est généralement représenté par un cercle ou une ellipse avec un point d'entrée.

2.5.2 Nœud d'activité

Représentent à la fois des actions qui reçoivent des entrées et les transforment en sorties pour d'autres nœuds, ou appellent d'autres activités, et des groupes d'actions.

2.5.3 Flèche de flux

Dans un diagramme d'activité, une flèche de flux entrante marque le début d'une étape ou d'une action dans l'activité. Une fois que cette étape est terminée, le flux se poursuit avec la flèche de flux sortante, indiquant ainsi la continuité du processus.

La flèche de flux est utilisée pour représenter la séquence d'exécution des activités, montrant comment le contrôle se déplace d'une étape à une autre dans le processus modélisé.

2.5.4 Décision (Decision) :

Une décision est un point dans le diagramme où le flux d'exécution se divise en plusieurs chemins possibles. Il est représenté par un losange et est généralement suivi de gardes (conditions) qui déterminent les chemins à suivre.

2.5.5 Synchronisation (join node) :

La synchronisation est utilisée pour coordonner les actions simultanées dans un diagramme d'activité. Elle spécifie des points où les différentes branches parallèles doivent se synchroniser avant de poursuivre l'exécution. Cela garantit que certaines conditions ou dépendances sont satisfaites avant de passer à l'étape suivante. Les points de synchronisation peuvent être des nœuds spécifiques dans le diagramme, où toutes les branches parallèles doivent arriver avant de continuer, ou des gardes (conditions) qui contrôlent la synchronisation en fonction de critères spécifiques.

2.5.6 Nœud de bifurcation ou de débranchement (fork Node)

Le nœud de bifurcation permet de créer une séparation dans le flux d'activités, où deux activités peuvent être exécutées en parallèle. Les lignes fléchées représentent les chemins d'activités distincts qui peuvent être suivis simultanément.

2.5.7 Le nœud de remarque

Le nœud de remarque est utilisé pour permettre aux créateurs d'un diagramme, ainsi qu'à leurs collaborateurs, de communiquer des messages supplémentaires qui ne sont pas inclus directement dans le diagramme lui-même. Il sert à laisser des notes pour plus de clarté et de précision.

2.5.8 Le nœud d'envoi de signal

Le nœud d'envoi de signal est utilisé pour indiquer qu'un signal est envoyé à une activité réceptrice spécifique dans un diagramme d'activité, permettant ainsi la communication et l'interaction entre les différentes activités.

2.5.9 Le nœud du signal de réception

Le nœud du signal de réception est utilisé pour représenter la réception d'un signal dans un diagramme d'activité. Il marque le point où une activité est en attente de la réception d'un signal spécifique avant de pouvoir poursuivre son flux d'exécution.

2.5.10 Boucle optionnelle

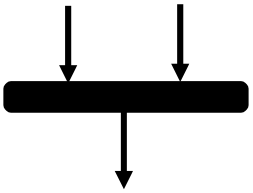
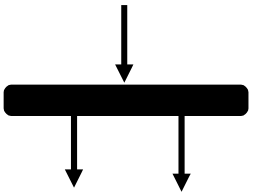
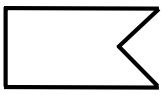
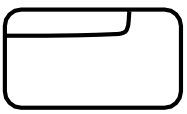
Utilisé pour représenter une boucle facultative dans un diagramme d'activité. Il indique qu'une activité peut être répétée plusieurs fois en fonction d'une condition spécifiée, mais que cette répétition n'est pas obligatoire.

2.5.11 Nœud final (Final Node) :

Un nœud final est un élément de contrôle dans un diagramme d'activité où un ou plusieurs flux d'activités se terminent au sein d'une activité donnée. Il existe deux types de nœuds finaux :

- **Nœud de flux final :** Ce type de nœud final représente la fin d'un flux d'activités spécifique à l'intérieur de l'activité. Lorsque le flux d'activité atteint ce nœud, cela signifie que cette branche spécifique du flux d'activités est terminée. Le nœud de flux final est généralement représenté par un petit cercle plein.
- **Nœud d'activité final :** Ce type de nœud final représente la fin complète de l'activité elle-même. Lorsque le flux d'activité atteint ce nœud, cela signifie que toute l'activité

est terminée. C'est souvent utilisé pour marquer la fin d'un processus ou d'un scénario complet. Le nœud d'activité final est généralement représenté par un cercle avec un contour épais

Notation	Symbole
Nœud initial	
Activité	
Flèche de flux	
Décision	
Synchronisation	
Bifurcation	
Remarque	
Signal d'émission	
Signal de réception	
Boucle optionnelle	
Fin de flux	

Nœud final	

Tableau 0-1: les nœud du diagramme d'activité

2.6 Modélisation des comportements avec les diagrammes d'activité

2.6.1 Séquencement des actions et des activités

- Les diagrammes d'activité permettent de modéliser le séquencement des actions et des activités, c'est-à-dire l'ordre dans lequel elles doivent être exécutées.
- Les actions et les activités sont représentées par des nœuds dans le diagramme d'activité, et les arcs (transitions) indiquent le flux de contrôle entre eux, déterminant ainsi l'ordre d'exécution.

2.6.2 Actions de base (actions simples, actions structurées)

- Les actions simples sont des tâches élémentaires qui ne peuvent pas être décomposées davantage. Elles représentent des actions atomiques dans le système modélisé.
- Les actions structurées regroupent un ensemble d'actions simples ou d'autres actions structurées pour représenter des comportements plus complexes. Elles permettent de décomposer une tâche en sous-tâches pour une meilleure compréhension et modélisation.

2.6.3 Activation des actions (parallèle, conditionnelle, boucles)

- L'activation parallèle permet d'indiquer que plusieurs actions peuvent être exécutées simultanément, sans ordre précis. Cela est représenté par des branches parallèles dans le diagramme d'activité.
- L'activation conditionnelle permet de spécifier que l'exécution d'une action dépend d'une condition spécifique. Cela est réalisé en utilisant des gardes (conditions) sur les transitions.
- Les boucles permettent de représenter des répétitions d'actions ou d'activités jusqu'à ce qu'une condition spécifique soit satisfaite. Elles peuvent être modélisées à l'aide de nœuds de décision et de transitions spécifiques.

2.6.4 Gestion des exceptions et des erreurs :

Les diagrammes d'activité permettent de modéliser la gestion des exceptions et des erreurs dans un système.

- Des nœuds spécifiques tels que les nœuds "try", "catch" et "finally" peuvent être utilisés pour représenter la gestion des exceptions et définir les actions à prendre en cas d'erreur.
- Les transitions peuvent être utilisées pour indiquer les chemins d'exécution alternatifs en cas d'exception ou d'erreur. [11]

2.7 Bonnes pratiques et conseils pour un diagramme d'activité efficace

2.7.1 Simplification et abstraction des activités :

Identifiez les activités clés et les étapes essentielles du processus. Évitez de vous encombrer de détails excessifs qui pourraient rendre le diagramme trop complexe.
Regroupez les activités similaires ou répétitives pour simplifier le diagramme. Utilisez des sous-processus pour représenter des actions récurrentes ou des parties du processus qui se répètent.

2.7.2 Utilisation de libellés et de commentaires pour clarifier les actions

Utilisez des libellés clairs et concis pour décrire chaque activité. Assurez-vous que les libellés sont compréhensibles pour les personnes impliquées dans le processus.

Ajoutez des commentaires ou des notes explicatives pour fournir des informations supplémentaires ou des clarifications sur des activités spécifiques. Cela peut aider les lecteurs à mieux comprendre le fonctionnement du processus.

2.7.3 Gestion de la complexité avec des niveaux d'abstraction :

Si votre diagramme d'activité devient trop complexe, envisagez d'utiliser des niveaux d'abstraction pour représenter différents niveaux de détail. Vous pouvez utiliser des sous-processus ou des diagrammes d'activités séparés pour traiter les parties complexes du processus. Utilisez des symboles de regroupement pour regrouper des activités connexes ou pour indiquer des parties du processus qui peuvent être traitées séparément.

2.7.4 Validation et vérification des diagrammes d'activité :

Vérifiez la logique et la cohérence du diagramme d'activité pour vous assurer qu'il représente correctement le processus. Assurez-vous que les actions sont ordonnées de manière logique et que les décisions sont correctement représentées.

Faites valider votre diagramme d'activité par les parties prenantes concernées pour vous assurer de son exactitude. Obtenez des commentaires et des suggestions pour améliorer le diagramme si nécessaire.

Révisez régulièrement vos diagrammes d'activité pour les maintenir à jour et refléter les évolutions du processus réel.

En suivant ces bonnes pratiques, vous pouvez créer des diagrammes d'activité efficaces qui communiquent clairement le déroulement du processus et facilitent la compréhension pour les parties prenantes**[9]**.

2.8 Conclusion

En conclusion, les diagrammes d'activité sont des outils précieux pour modéliser et représenter les comportements et les processus dans divers domaines, tels que l'ingénierie logicielle, la gestion des processus métier et l'analyse des systèmes. En suivant les bonnes pratiques et les conseils appropriés, il est possible de créer des diagrammes d'activité efficaces qui facilitent la compréhension, la communication et l'optimisation des processus.

Chapitre 03 : Les réseaux de Petri

3.1 Introduction :

Les réseaux de Petri sont des outils extrêmement puissants pour modéliser, analyser qualitativement et quantitativement différentes classes de systèmes. L'un de leurs principaux avantages réside dans leur représentation graphique intuitive, qui facilite l'analyse, ainsi que dans leurs propriétés analytiques qui permettent une évaluation simple du comportement du système étudié. En résumé, les réseaux de Petri constituent un outil complet et efficace.

Le modèle des réseaux de Petri est particulièrement approprié pour la modélisation des systèmes collaboratifs en raison de sa rigueur, de sa lisibilité et de ses deux éléments fondamentaux : les places et les transitions. Ces réseaux permettent de représenter l'état d'un système ainsi que les différentes évolutions possibles de cet état. De plus, ce modèle peut être étendu pour prendre en compte des aspects temporels et aléatoires grâce à son extension stochastique.

3.2 Historique :

Les réseaux de Petri sont t en anglais. Suivez la newsletter Petri-Net aujourd'hui, il y a 600 à 800 travaux Petri-Net publiés chaque année.

Ce formalisme est proposé comme un outil mathématique permettant la modélisation de systèmes dynamiques à événements discrets. Les réseaux de Petri fournissent un outil formel avec une bonne représentation graphique pour la modélisation et l'analyse de systèmes discrets, y compris les systèmes concurrents et parallèles. L'intérêt principal de ces réseaux réside dans leur possibilité de modéliser des systèmes analytiques. En fait, ce formalisme bénéficie d'un grand nombre de techniques et d'outils analytiques**[12]**.

3.3 Définitions fondamentales :

3.3.1 Définition informelle :

Un réseau de Petri est un type de graphe en deux parties composées de deux composants principaux : les places (Pi) et les transitions (Ti). Les lieux sont des cercles qui peuvent contenir des jetons, représentant des ressources, des informations ou des conditions requises pour que la transition se produise. Les transitions sont des rectangles qui représentent des événements ou des actions qui peuvent se produire dans le système. Les lieux et les transitions sont reliés par des crochets avec une signification spécifique. La valeur (1) a été omise et les parenthèses indiquant la valeur (0) sont absentes.

Un réseau de Petri a une sémantique opérationnelle, ce qui signifie qu'il a des comportements assignés qui décrivent la dynamique du système représenté. Pour cela, des jetons sont ajoutés aux carrés sous forme de points ou de chiffres à l'intérieur des cercles. La distribution de ces jetons à des endroits à un instant donné s'appelle l'apprentissage par réseau de Petri**[13]**

.

3.3.2 Définition formelle :

Formellement, un Réseau de Petri marqué est un 5-uplet, Rdp = (P, T, F, W, M0)
Où [03] :

- ❑ P = {P1, P2, · · ·, Pm} est un ensemble fini non vide de places P.
- ❑ T = {t1, t2, · · ·, tn} est un ensemble fini non vide de transitions T.
- ❑ F ⊆ (P × T) ∪ (T × P) est un ensemble d'arcs où :
  - ➥ (P × T) est l'arc allant de P à T.
  - ➥ (T × P) est l'arc allant de T à P.
- ❑ W : F → {1, 2, 3, · · ·} est une fonction du poids où :
  - ➥ W (P, T) : "Pré (p, t)" est le poids de l'arc allant de P à T.
  - ➥ W (T, P) : "Post (p, t)" est le poids de l'arc allant de T à P.
- ❑ M0 : P → {0, 1, 2, 3, · · ·} est le marquage initial.
- ❑ P ∩ T = φ et P ∪ T 6= φ.

3.4 Marquage d'un Réseau de Petri :

Un marquage est représenté par un vecteur du nombre de jetons dans chaque place : Le

i^{éme} élément du vecteur correspond au nombre de jetons contenus dans la i^{ième} place Pi **[04].**
M = (1, 0, 1, 0, 0, 2, 0)

Le réseau de Petri marqué correspondant peut-être représenter graphiquement comme suit :

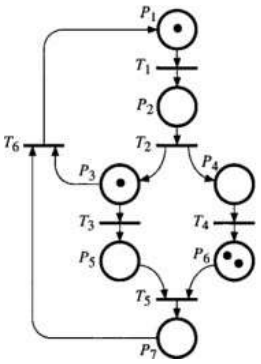


Figure 0-1: Exemple de Réseau de Petri marqué**[15].**

3.5 Évolution d'un Réseau de Petri :

Le réseau de Petri évolue en fonction des changements de marquage, qui sont représentés par les jetons indiquant l'état du réseau à un moment donné. Les jetons peuvent être déplacés d'une place à une autre en franchissant une transition ou en tirant une transition. Dans les réseaux de Petri à arcs simples ou de poids égal à 1, le franchissement d'une transition implique la suppression d'un jeton dans chaque place d'entrée de la transition et l'ajout d'un jeton dans chaque place de sortie. **[16]**

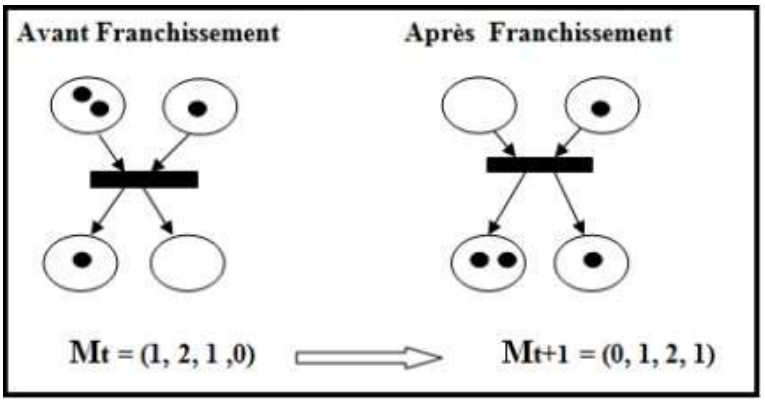


Figure 0-2 : Evolution d'états d'un réseau de pétri.

En général, l'évolution des états d'un réseau de Petri marqué simple suit les règles suivantes **[16]**:

- Une transition est dite franchissable, sensibilisée ou tirable lorsque chacune des places d'entrée possède au moins le nombre de jetons correspondant au poids de l'arc qui relie cette place à la transition.
- Le réseau ne peut évoluer qu'en franchissant une seule transition à la fois, sélectionnée parmi toutes celles qui sont validées au moment du choix.
- Le franchissement d'une transition est indivisible et de durée nulle.

Ces règles introduisent un certain degré d'indétermination dans l'évolution des réseaux de Petri, car différents états peuvent être atteints en fonction du choix des transitions tirées. Cependant, ce mode de fonctionnement est souvent adapté aux situations réelles où il n'y a pas de priorité claire dans la succession des événements.

3.6 Propriétés des RdP :

Les Réseaux de Petri (RdP) possèdent plusieurs propriétés qui permettent de les caractériser et de les analyser. Voici les principales propriétés des RdP**[17]** :

- **Vivacité :** un réseau de Petri est dit vivant (ou safeness en anglais) s'il est possible de franchir toutes les transitions à partir de tous les marquages accessibles. Autrement dit, il n'y a pas de blocage dans le réseau de Petri. La vivacité est une propriété importante pour les systèmes critiques où tout blocage pourrait entraîner des conséquences graves. La vivacité peut être vérifiée en construisant un graphe des marquages et en vérifiant qu'il n'y a pas de marquage de blocage.
- **Borgnitude :** un réseau de Petri est borné si pour tout marquage accessible, il existe un nombre fini de jetons dans chaque place. Cette propriété assure que le réseau de Petri ne peut pas générer de comportement infini ou non-terminant. La borgnitude peut être vérifiée en construisant une matrice d'incidence et en calculant ses valeurs propres. Si toutes les valeurs propres sont finies, alors le réseau de Petri est borné.
- **Accessibilité :** une place est dite accessible si elle contient un jeton dans au moins un marquage accessible. Un réseau de Petri est dit accessible si toutes les places sont accessibles. Cette propriété permet de s'assurer que toutes les parties du réseau peuvent être atteintes et que le réseau est cohérent. L'accessibilité peut être vérifiée en construisant un graphe de couverture et en vérifiant qu'il n'y a pas de place inaccessible.
- **Réversibilité :** un réseau de Petri est dit réversible si pour tout marquage accessible, il existe au moins un autre marquage accessible à partir duquel on peut revenir au

marquage initial en exécutant les mêmes transitions dans l'ordre inverse. Cette propriété permet de vérifier que le réseau de Petri peut fonctionner dans les deux sens.

- **Blocage :** Un réseau de Petri est dit bloqué si aucune transition marquée n'est franchissable. Autrement dit, le système est dans un état où il ne peut plus progresser.
- **Conservativité :** un réseau de Petri est dit conservatif si le nombre total de jetons dans le réseau reste constant lors de l'exécution de n'importe quelle séquence de transitions. Cette propriété permet de vérifier que le réseau de Petri ne crée pas ou ne détruit pas de ressources.
- **Réinitialisabilité :** un réseau de Petri est dit réinitialisable si à partir de n'importe quel marquage accessible, il est possible de revenir à un marquage où toutes les places sont vides. Cette propriété permet de vérifier que le réseau de Petri peut être remis à zéro.

3.7 Méthodes d'analyse pour les réseaux de pétri :

La modélisation d'un système doit permettre l'analyse de ses propriétés. Les réseaux de pétri offrent des techniques d'analyse puissantes pour valider des modèles de comportement de systèmes à événements discrets. Parmi ces techniques, nous citons le graphe de marquages, l'équation de matrice et la réduction des réseaux de pétri**[16]**.

- **Le graphe de marquage** : est un outil permettant de construire un graphe de tous les marquages possibles du réseau. Les propriétés du réseau peuvent ensuite être déduites à partir de ce graphe en utilisant des techniques de la théorie des graphes.
- **L'équation de matrice** : est une méthode qui consiste à trouver une représentation matricielle du réseau de pétri. En utilisant des techniques d'algèbre linéaire, il est alors possible d'obtenir les propriétés du réseau, telles que le nombre de marquages accessibles ou les temps de franchissement des transitions.
- **La réduction des RDPs :** pour analyser les propriétés d'un grand réseau de Pétri marqué, le graphe de marquage ou l'équation de matrice peuvent ne pas suffire. C'est pourquoi la technique de réduction des réseaux de Pétri est utilisée. Son objectif est de simplifier le réseau en réduisant le nombre de places et de transitions, tout en conservant les propriétés importantes. Cette technique permet d'obtenir un réseau de Pétri marqué plus simple, qui peut être analysé plus efficacement à l'aide d'autres méthodes telles que le graphe de marquage ou l'équation de matrice.

- **Analyse structurelle :** cette méthode consiste à étudier les propriétés structurelles du RdP telles que le nombre de places et de transitions, les relations entre les places et les transitions, etc. pour déterminer si une propriété donnée est satisfaite ou non.
- **Simulation :** cette méthode consiste à simuler le comportement du RdP pour une certaine période de temps et à observer si une propriété donnée est satisfaite ou non.

3.8 Réseaux de pétri particuliers

3.8.1 Graphe d'état

Le graphe d'état est un type particulier de réseau de Pétri non marqué, dans lequel chaque transition possède exactement une place d'entrée et une place de sortie. Ainsi, un réseau de Pétri non marqué peut être considéré comme un graphe d'état si cette condition est satisfaite pour toutes les transitions. Par exemple, un réseau de Pétri avec les transitions T1, T2, T3 et T4 qui ont chacune une place d'entrée et une seule place de sortie. Comme il est schématisé dans la (**Figure 0-3)**

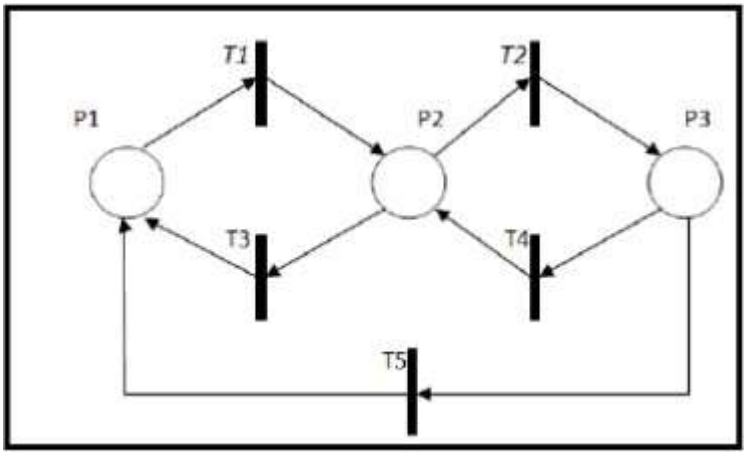


Figure 0-3 : Graphe d'état.

3.8.1.1 Graphe d'événement

Pour être considéré comme un graphe d'événement, un réseau de pétri doit avoir la propriété suivante : chaque place du réseau doit être connectée à exactement une transition d'entrée et une transition de sortie. Cette caractéristique peut être visualisée sur un schéma graphique, comme li est détaillé dans **(Figure 0-4)**.

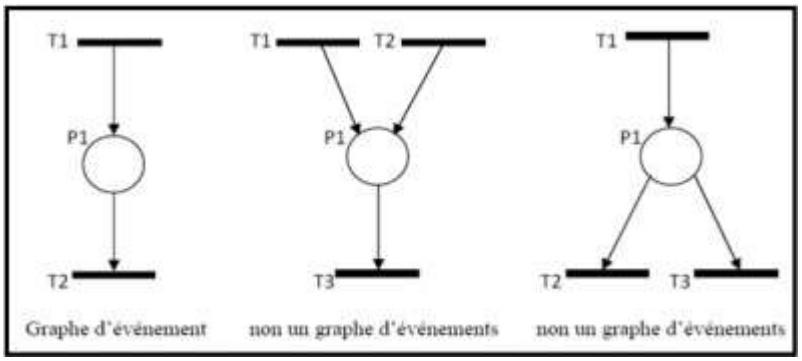


Figure 0-4 : Graph d'événement.

3.8.2 Rdp sans conflit

Un réseau de Petri est considéré comme sans conflit lorsque chaque place possède au maximum une transition de sortie. En revanche, un réseau de Petri avec conflit est caractérisé par une place ayant au moins deux transitions de sortie, noté sous la forme [Pi, T1, T2, ..., Tn], où T1, T2, ..., Tn sont les transitions de sortie de la place Pi. Comme il est présenté dans **(figure 3-5)**

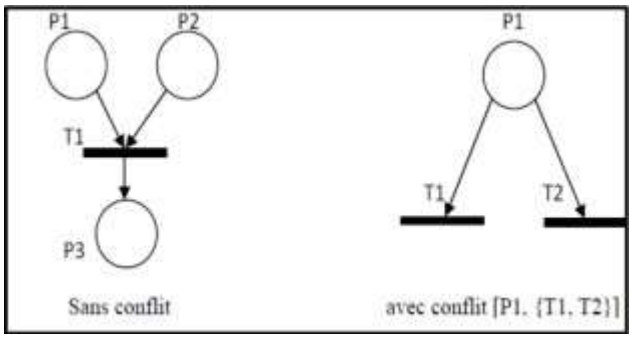


Figure 0-5 : Graphe RdP sans conflit.

3.8.3 RDP a choix libre

Le Réseau de Petri à choix libre est un type de réseau dans lequel, pour tout conflit impliquant une place Pi et les transitions T1, T2, ..., Tn, aucune de ces transitions n'a d'autre place d'entrée que Pi. Autrement dit, lorsqu'il y a plusieurs transitions qui peuvent potentiellement être déclenchées à partir d'une même place, dans un Rdp à choix libre, aucune de ces transitions n'a besoin d'être en concurrence avec une autre place. Cela permet de simplifier la structure du réseau et de rendre son comportement plus prévisible. Comme il est exprimé dans **(figure 3-6).**

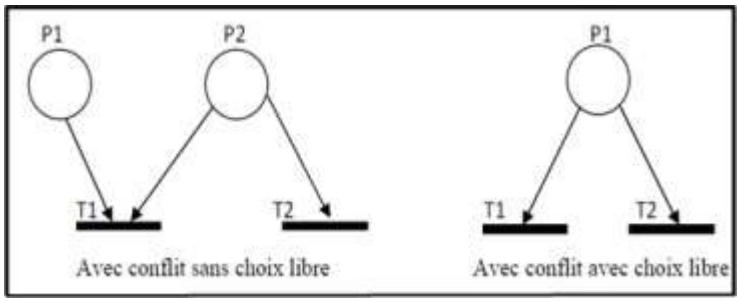


Figure 0-6 : RdP avec choix libre.

3.8.4 Réseau de Petri à priorité

Dans un Réseau de Petri à priorité si plusieurs transitions sont franchissables à partir d'un marquage, il est obligatoire de franchir la transition ayant la plus grande priorité. Cela signifie que les transitions ont été classées en fonction de leur importance et qu'il existe une hiérarchisation claire des transitions. La priorité peut être établie selon différents critères, tels que l'importance fonctionnelle, la criticité du système, les besoins en ressources, etc. En suivant cet ordre de priorité, on peut garantir un comportement fiable et cohérent du système. Comme il est exprimé dans la figure suivante

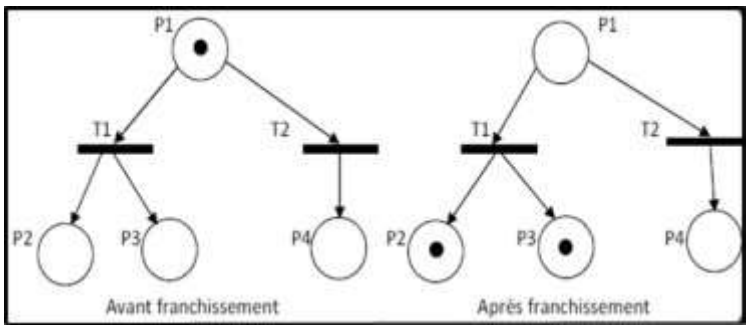


Figure 0-7 : RdP à priorité.

3.9 Extensions des réseaux de pétri

Les systèmes réels sont souvent soumis à de nombreuses contraintes, ce qui peut entraîner la création de Réseaux de Petri de taille importante. La manipulation et l'analyse de tels réseaux peuvent s'avérer très difficiles. De plus, les Réseaux de Petri traditionnels ne permettent pas d'exprimer certaines propriétés, telles que la mobilité, ce qui rend leur analyse encore plus complexe. Dans la section suivante, nous donnerons un aperçu de certaines extensions des Réseaux de Petri qui ont été développées pour pallier ces limitations.

3.9.1 Les réseaux de pétri temporisés

Les Réseaux de Pétri Temporisés (RDPT) étendent les Réseaux de Pétri classiques en ajoutant des intervalles temporels associés aux transitions pour spécifier les bornes de délai de tir. Cela permet de modéliser des systèmes avec des contraintes temporelles telles que les protocoles de communication ou certains systèmes temps réel. Les bornes de délai peuvent être définies sur les places, les arcs ou les transitions. Selon Merlin, un RDPT est un RDP avec deux valeurs temporelles a et b associées aux transitions, où $0 \leq a \leq b$ et b peut être illimité. Ces valeurs spécifient les bornes de délai de franchissement des transitions. Si la transition t a été sensibilisée pour la dernière fois à l'instant θ, elle ne peut être franchie avant l'instant $\theta + a$ et ne doit pas être franchie plus tard que l'instant $\theta + b$ (ou exactement à cet instant). L'intervalle [a, b] représente le temps pendant lequel les jetons ne sont plus présents dans les places d'entrée (car ils sont réservés), mais les jetons produits ne sont pas encore visibles dans les places de sortie. **[19]**

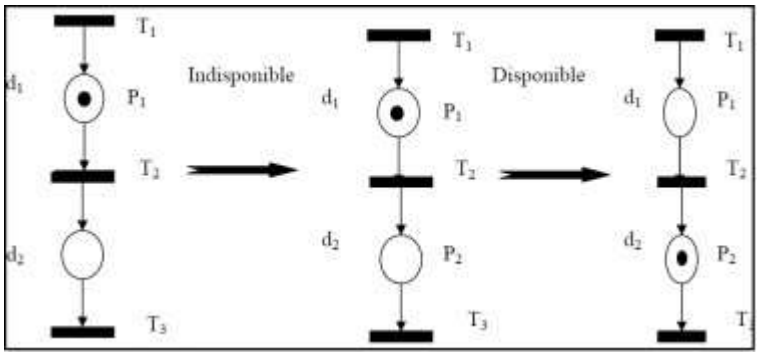


Figure 0-8 : RdP temporisés.

3.9.2 Les réseaux de pétri colorés

Les Réseaux de Pétri colorés (RdPC) sont une extension des RdP permettant la modélisation de systèmes avec des états non marqués, c'est-à-dire des systèmes dans lesquels les transitions peuvent être activées même si les places ne sont pas pleines. Dans les RdPC, les jetons et les arcs peuvent être associés à des couleurs, qui peuvent représenter différentes caractéristiques ou attributs des objets modélisés. Les RdPC permettent ainsi une modélisation plus précise et expressive des systèmes complexes.

Les méthodes d'analyse des RdPC sont basées sur des algorithmes de coloration de graphe et de couverture de graphe, qui permettent de vérifier des propriétés telles que la vivacité, la borgnitude et l'accessibilité. Les RdPC ont de nombreuses applications, notamment dans la modélisation de systèmes de communication, de systèmes de production, de systèmes informatiques, etc. **[20]**

3.9.3 Les réseaux de Petri synchronisés

Les réseaux de Petri synchronisés sont une extension des réseaux de Petri classiques dans lesquels les transitions sont activées en synchronisation avec un événement externe ou une condition. Cela signifie que les transitions ne peuvent pas être activées tant que les conditions nécessaires ne sont pas remplies ou que l'événement synchronisateur n'a pas eu lieu. Cette caractéristique permet de modéliser des systèmes qui impliquent des interactions entre plusieurs parties et qui nécessitent une coordination précise pour fonctionner correctement[14].

3.9.4 Les Réseaux de Pétri stochastiques

Les Réseaux de Pétri stochastiques (RdPS) sont une extension des Réseaux de Pétri classiques qui permettent de modéliser des systèmes dont le comportement est aléatoire. Dans un RdPS, chaque transition est associée à une probabilité de franchissement qui dépend de l'état du réseau à un instant donné. Contrairement aux RdP classiques, où les transitions sont franchissables dès que les jetons nécessaires sont disponibles, les RdPS introduisent une notion de probabilité pour chaque transition, qui permet de modéliser des situations où plusieurs transitions concurrentes peuvent être activées, ou bien des transitions qui ne sont pas certaines d'être franchies.

Les RdPS sont souvent utilisés pour modéliser des systèmes avec des temps de traitement aléatoires, ou bien des situations où le comportement du système dépend de facteurs externes aléatoires, tels que des pannes ou des défaillances. Les méthodes d'analyse des RdPS sont basées sur des techniques de simulation stochastique ou bien de calcul de probabilités, et permettent d'estimer des mesures telles que les temps de réponse moyens, les taux d'occupation, ou les probabilités de panne**[21]**.

3.10 Modélisation de systèmes à l'aide de RdP :

Les réseaux de Petri (RdP) sont des outils de modélisation puissants qui peuvent être utilisés pour représenter divers types de systèmes, notamment les systèmes de production, les systèmes de transport ou les systèmes informatiques**[18]**.

3.10.1 Systèmes informatiques

Les réseaux de Petri ont été largement utilisés pour la modélisation de systèmes informatiques, tels que les protocoles de communication, les systèmes d'exploitation et les

systèmes distribués. Les RdP permettent de représenter de manière formelle et graphique les différents états du système, les interactions entre les différents composants du système, ainsi que les événements qui peuvent se produire.

Par exemple, un RdP peut être utilisé pour modéliser le fonctionnement d'un système d'exploitation en représentant les différents processus du système en tant que places, et les ressources système en tant que transitions. Chaque processus peut être représenté par une place contenant une certaine quantité de jetons, qui représentent l'état du processus. Les transitions représentent les différentes actions que le système d'exploitation peut effectuer, telles que l'allocation de ressources à un processus, la libération de ressources, ou la création et la suppression de processus.

3.10.2 Systèmes de transports

Les Réseaux de Petri peuvent également être utilisés pour modéliser des systèmes de transport, tels que des réseaux de circulation routière, des systèmes de transport en commun, ou encore des chaînes de production et de distribution de marchandises.

Par exemple, dans le cas d'un réseau de circulation routière, les intersections et les routes peuvent être modélisées sous forme de places et de transitions, où les voitures représentent des jetons se déplaçant entre les différentes places. Les transitions peuvent alors représenter les feux de signalisation, les carrefours ou les ronds-points, qui permettent aux voitures de changer de direction ou de continuer tout droit. La modélisation de ce type de système permet d'analyser la fluidité du trafic, les temps d'attente, les congestions, etc.

3.10.3 Systèmes de production

La modélisation de systèmes de production à l'aide de RdP est une application courante des RdP en génie industriel. Les RdP peuvent être utilisés pour modéliser et analyser divers aspects des systèmes de production, tels que la planification de la production, l'ordonnancement, la gestion des stocks et la logistique.

Par exemple, dans une chaîne de production, chaque étape peut être modélisée par une place dans le RdP, et chaque transition peut représenter une action qui transforme les ressources en produits finis. Les entrées et les sorties des places représentent les flux de matières premières, de produits semi-finis et de produits finis. Les transitions représentent les machines, les opérateurs ou les robots qui effectuent les différentes étapes de production.

Les RdP peuvent également être utilisés pour modéliser la gestion des stocks. Dans ce cas, les places représentent les niveaux de stock des différents produits, tandis que les transitions

représentent les mouvements de stock, tels que les entrées et sorties de stock.

3.11 Conclusion :

En conclusion, les réseaux de Petri sont un outil puissant pour modéliser et analyser les systèmes dynamiques concurrents. Ils permettent de représenter les interactions complexes entre les différents éléments du système et ont des propriétés mathématiques bien définies qui permettent de les analyser et de les simuler. Les réseaux de Petri ont été largement utilisés dans de nombreux domaines et continuent d'être une méthode de modélisation et d'analyse de choix pour de nombreux chercheurs et ingénieurs.

Chapitre 04 : Implémentation et mise en œuvre

4.1 Introduction

Dans ce chapitre, nous introduisons l'environnement d'implémentation que nous avons utilisé pour notre recherche, ainsi que le processus de transformation des modèles à l'aide de la TGG (Transformation Graph Grammar). L'objectif principal de notre travail est d'utiliser l'outil TGG-interpréter afin de maîtriser la transition des modèles de diagrammes d'activité UML vers les Réseaux de Petri. Cette migration implique la traduction du schéma source en un schéma cible.

4.2 Environnement d'implémentation

Nous avons utilisé TGG (Transformation Graph Grammar) et JDK (Java Development Kit) pour effectuer la transformation et la manipulation des modèles dans un environnement technique adapté à la modélisation et à la transformation des modèles. Pour transformer les diagrammes d'état transition en réseaux de Petri, nous proposons une méthode basée sur les trois étapes suivantes :

1. Construction du méta-modèle des diagrammes d'état transition : Cette étape permet la transformation des métamodèles de l'espace technique EMF (Eclipse Modeling Framework) en grammaires de graphes dans l'espace technique TGG. Cela implique de définir les éléments et les relations nécessaires pour représenter les diagrammes d'état transition.

2. Construction du méta-modèle des réseaux de Petri : Cette étape consiste à spécifier les règles de transformation et la grammaire de correspondance des graphes. Elle permet de définir les éléments et les relations nécessaires pour représenter les réseaux de Petri.

3. Définition des règles de transformation : Cette étape est utilisée pour générer le moteur de transformation et exécuter ce dernier sur un modèle source afin d'obtenir un modèle cible. Les règles de transformation définissent les correspondances et les modifications à effectuer lors de la transformation des diagrammes d'état transition en réseaux de Petri.

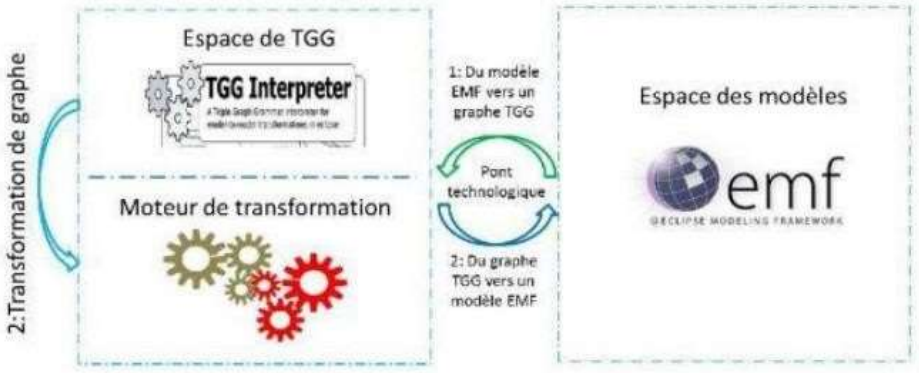


Figure 0-1 : Principe de mise en œuvre de transformation de modèles.

4.2.1 Eclipse :

Eclipse est un projet de développement logiciel de la fondation Eclipse. Il est composé de plusieurs sous-projets qui visent à créer un environnement de production de logiciels libre, extensible, universel et polyvalent. L'environnement Eclipse est principalement basé sur Java. L'objectif principal d'Eclipse est de fournir des outils pour faciliter la création de logiciels, couvrant notamment les activités de programmation, telles que les environnements de développement intégrés (IDE) et les frameworks.

4.2.2 Eclipse Modeling Framework :

Le projet EMF (Eclipse Modeling Framework) est un framework de modélisation et un générateur de code utilisé pour la construction d'outils et d'applications basées sur un modèle de données structuré. EMF permet de spécifier un modèle de données à l'aide de la norme XMI (XML Metadata Interchange), puis fournit des outils et un support d'exécution pour générer un ensemble de classes Java correspondant à ce modèle.

Une fois que les classes Java sont générées, EMF fournit également un ensemble de classes adaptateurs qui permettent d'afficher et d'éditer le modèle de manière basée sur des commandes. Cela signifie que les modifications apportées au modèle sont effectuées en utilisant des opérations définies dans le modèle lui-même, ce qui facilite la manipulation du modèle de données.

EMF propose également un éditeur de base qui offre une interface utilisateur pour visualiser et

modifier les instances du modèle. Cet éditeur de base peut être étendu pour ajouter des fonctionnalités spécifiques à l'application ou au domaine d'utilisation. **[22].**

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**[22] .**

EMF se compose de trois éléments fondamentaux :

1. **EMF - Le Framework** L'élément central d'EMF (Eclipse Modeling Framework) est son méta-modèle, appelé Ecore, qui permet de décrire les modèles de manière structurée. EMF prend en charge l'exécution des modèles, ce qui inclut la notification des modifications, la gestion de la persistance avec la sérialisation XMI par défaut, ainsi qu'une API réfléchissante hautement performante pour manipuler les objets EMF de manière générique. Grâce à Ecore, on peut définir la structure des modèles et leurs relations, tandis que l'exécution des modèles gère les instances et assure la mise à jour des modifications. La persistance des modèles est facilitée par la sérialisation en format XMI, permettant leur enregistrement et chargement. L'API réfléchissante d'EMF offre une grande flexibilité en permettant une manipulation dynamique des objets EMF sans nécessiter une connaissance préalable de leur structure. En somme, EMF propose un cadre complet pour la création, l'exécution, la persistance et la manipulation générique des modèles grâce à son méta-modèle Ecore et ses fonctionnalités intégrées.
2. **EMF.Edit - Le Framework** : EMF.Edit fournit un ensemble de classes génériques réutilisables pour faciliter la création d'éditeurs pour les modèles EMF. Il comprend des classes de fournisseur de contenu et d'étiquettes, ainsi que la prise en charge des sources de propriétés et d'autres classes pratiques. Ces fonctionnalités permettent d'afficher les modèles EMF à l'aide de visualiseurs et de feuilles de propriétés standard (JFace) dans un environnement de bureau.

3. **EMF.Codegen** : La fonction de génération de code d'EMF offre la possibilité de générer tous les éléments nécessaires à la création d'un éditeur complet pour un modèle EMF. Elle comprend une interface graphique qui permet de spécifier les options de génération et d'invoquer les générateurs nécessaires. L'outil de génération s'appuie sur le composant JDT (Java Development Tooling) d'Eclipse, qui fournit un ensemble de fonctionnalités dédiées au développement Java. Grâce à cette intégration, l'outil de génération de code

EMF offre une solution complète pour la création d'éditeurs basés sur les modèles EMF, en simplifiant et en automatisant le processus de génération du code source et des artefacts nécessaires. Cela permet aux développeurs de se concentrer davantage sur la logique métier de leur application, en réduisant le temps et les efforts nécessaires pour créer un éditeur fonctionnel. **[22] .**

4.2.3 Le Java Développement Kit (JDK) :

Le Java Development Kit (JDK) fait référence à un ensemble de bibliothèques logicielles essentielles du langage de programmation Java, ainsi qu'aux outils permettant la compilation du code Java et sa transformation en bytecode destiné à la machine virtuelle Java.
Le JDK se décline en plusieurs éditions, en fonction de la plateforme Java considérée et de la version de Java ciblée :
JSE (Java 2 Standard Edition), également connue sous le nom de J2SE, pour le développement d'applications Java standard.
JEE (Java Enterprise Edition), également appelée J2EE, pour le développement d'applications d'entreprise Java.
JME (Java Micro Edition), destinée au développement d'applications mobiles.
Chacune de ces plateformes a une base de développement commune avec les kits, ainsi que des bibliothèques supplémentaires spécifiques à la plateforme ciblée. Cependant, le terme JDK est utilisé indifféremment pour désigner n'importe laquelle de ces plateformes Java.
En résumé, le JDK est un ensemble de bibliothèques et d'outils essentiels pour le développement Java, avec des éditions spécifiques pour différentes plateformes Java, telles que JSE, JEE et JME. **[23]**.

4.2.4 TGG interpreter :

Le TGG Interpréter est un outil spécialement conçu pour la transformation de modèles basée sur les TGG (Transformation Graph Grammar). Il offre une fonctionnalité de mise à jour incrémentale résultant de la comparaison entre les TGG et la norme bidirectionnelle QVT (Query/View/Transformation) de l'OMG (Object Management Group) pour les transformations de modèles.
L'objectif principal du TGG Interpréter est de permettre des transformations de modèles efficaces et réversibles, en assurant la cohérence entre les modèles sources et cibles. Grâce à sa capacité de mise à jour incrémentale, il est possible d'effectuer des transformations partielles

sur les modèles sans avoir à les reconstruire entièrement à chaque itération.

En utilisant les TGG et en se basant sur la norme QVT, le TGG Interpréter offre une approche robuste et formelle pour la transformation de modèles. Cela permet aux développeurs de mettre en œuvre des transformations complexes tout en assurant la cohérence et la traçabilité des modèles tout au long du processus de transformation. **[24].**

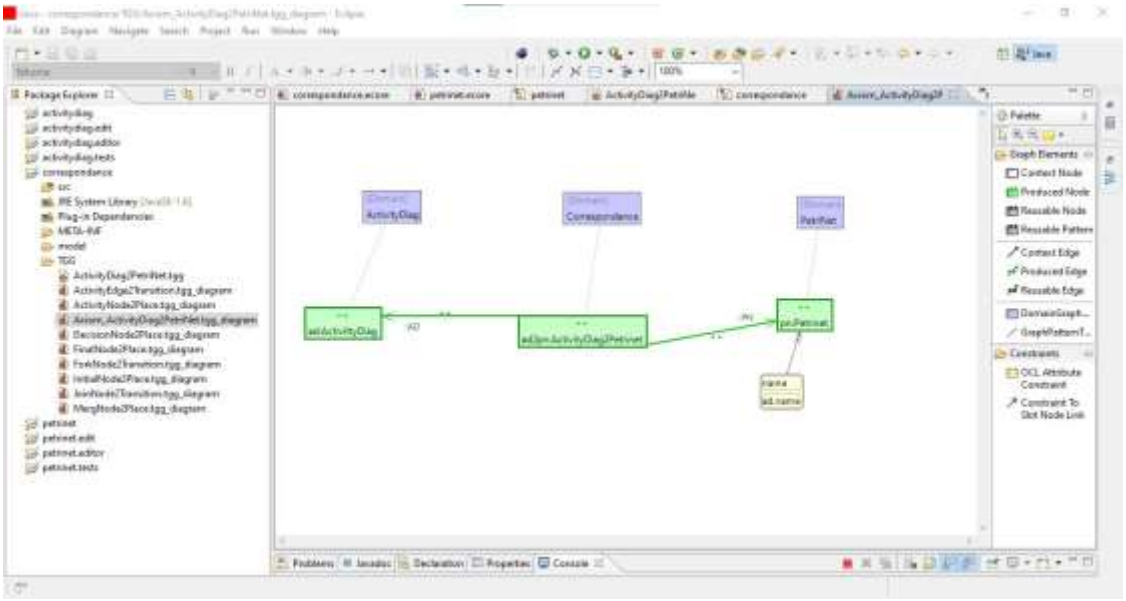


Figure 0-2 : Éditeur de Règle TGG de TGG Interpreter.

Une transformation de modèles efficace repose sur une définition claire et précise des règles de transformation. Dans le contexte de TGG (Transformation Graph Grammar), chaque transformation est composée de trois graphes : le graphe source, le graphe cible et le graphe de correspondance. Chaque graphe appartient à un domaine spécifique et est associé à un métamodèle.

Le domaine de gauche représente le graphe source, tandis que le domaine de droite représente le graphe cible. Le domaine de correspondance sert de lien entre les deux et permet d'établir les correspondances entre les éléments des graphes source et cible. Cette relation de correspondance est illustrée dans la figure 4.2.

Dans chaque domaine, les nœuds correspondent aux classes du métamodèle associé. Les nœuds de règle de transformation comprennent les nœuds de contexte, les nœuds de production, les contraintes et les nœuds réutilisables. Ces nœuds jouent un rôle clé dans la définition des règles de transformation et déterminent les actions à effectuer lors de la transformation. **[25]** .

- **Nœud de contexte :** Les nœuds de contexte jouent un rôle important dans la transformation des modèles et sont représentés graphiquement par des cases avec un contour noir dans les règles de TGG. Ils sont utilisés pour établir une correspondance avec les objets du modèle qui ont été traités précédemment. Ainsi, dans l'état actuel de

TGG, il est nécessaire de spécifier une grammaire complète pour toutes les parties du modèle.

Cependant, TGG offre également la possibilité de formuler une grammaire partielle pour un modèle. Dans ce cas, les nœuds de contexte ne nécessitent pas obligatoirement un prétraitement par une autre règle. Cela signifie que ces nœuds de contexte peuvent être utilisés pour établir de nouvelles correspondances sans dépendre d'une transformation préalable.

- **Nœud de production :** Les nœuds de production sont représentés par des boîtes vertes avec une bordure verte et une étiquette "++". Les arêtes de production sont affichées sous forme de flèches de couleur vert foncé avec une étiquette "++". L'objectif des nœuds de production est de créer de nouveaux éléments dans le domaine cible et le domaine de correspondance, à condition qu'ils n'existent pas déjà. Si ces éléments sont trouvés dans le domaine source, les éléments du modèle cible et du modèle de correspondance peuvent être créés selon la règle. Tous les objets créés sont liés aux nœuds de contexte de la règle, ce qui signifie qu'un objet de modèle ne peut être présenté en tant que nœud de production qu'une seule fois. Cette propriété est appelée la sémantique "bind-only-once" pour les nœuds de production.
- **Nœud réutilisable :** Il est important de noter que les types de composants ne doivent pas être générés à partir de zéro, mais peuvent être réutilisés à partir de nœuds existants qui possèdent les propriétés requises. Ces nœuds sont appelés "nœuds réutilisables". Graphiquement, ils sont représentés en gris. La sémantique des nœuds réutilisables permet de les générer à nouveau ou de les réutiliser de manière arbitraire.

  La couleur grise reflète le fait que chaque nœud réutilisable peut être soit en noir (un nœud du côté gauche de la règle de TGG, indiquant qu'il est réutilisé), soit en vert (un nœud du côté droit de la règle de TGG, indiquant qu'il est nouvellement généré). Ce choix peut être effectué à chaque fois que la règle est appliquée. En utilisant des nœuds réutilisables, nous pouvons réduire le nombre et la complexité des règles de TGG, ce qui évite d'avoir un nombre exponentiel de règles. De plus, dans des exemples complexes, l'absence de nœuds réutilisables peut entraîner un désordre dans les règles. Les nœuds réutilisables nous permettent d'avoir des règles plus simples et de nous concentrer sur les aspects essentiels des modèles pertinents. Ils offrent une approche plus modulaire et facilitent la compréhension et la maintenance des transformations de modèles.

- **Les contraintes :** Une contrainte réelle dans un nœud de contrainte peut consister en une expression OCL (Object Constraint Language) qui fait référence aux objets auxquels elle est attachée. Les contraintes sont utilisées pour mettre en œuvre des transformations ou des synchronisations plus efficaces. Une contrainte OCL est représentée par la couleur jaune et doit être attachée à un autre nœud.

  En utilisant des expressions OCL, les contraintes peuvent spécifier des conditions ou des règles supplémentaires à respecter lors de la transformation des modèles. Elles permettent de définir des critères plus précis pour filtrer ou manipuler les éléments des modèles. Les contraintes OCL offrent une flexibilité supplémentaire en permettant l'utilisation d'un langage formel pour exprimer des règles de transformation plus complexes.

4.3 Étude de cas

Pour mieux appréhender notre contribution dans le domaine de la transformation de modèles basée sur les grammaires de graphes, nous proposons une étude de cas qui illustre le processus de spécification et d'implémentation de la transformation. Cette étude de cas porte sur la transformation des diagrammes d'activité en réseaux de Petri.

En utilisant cet exemple concret, nous démontrons concrètement les étapes nécessaires pour spécifier la transformation et la mettre en œuvre. Nous analysons les diagrammes d'activité existants et définissons les règles de transformation correspondantes pour les convertir en réseaux de Petri équivalents. En suivant ce processus, nous mettons en évidence les avantages et les capacités de notre approche de transformation de modèles basée sur les grammaires de graphes.

Cette étude de cas nous permet de mieux comprendre les défis et les solutions liés à la transformation des modèles et de valider l'efficacité de notre méthode dans un contexte réel. Elle constitue une illustration concrète de l'application de notre approche et de ses résultats tangibles dans le domaine de la transformation des modèles

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'activité :

Un métamodèle de diagramme d'activité est une représentation abstraite qui décrit les concepts, les règles et les relations qui sous-tendent les diagrammes d'activité. Il définit les éléments de base d'un diagramme d'activité, tels que les nœuds (étapes de traitement), les arcs (transitions), les objets de données, les partitions, les groupes d'activités et les actions. Le métamodèle définit également les règles de syntaxe et de sémantique pour la construction de diagrammes d'activité.

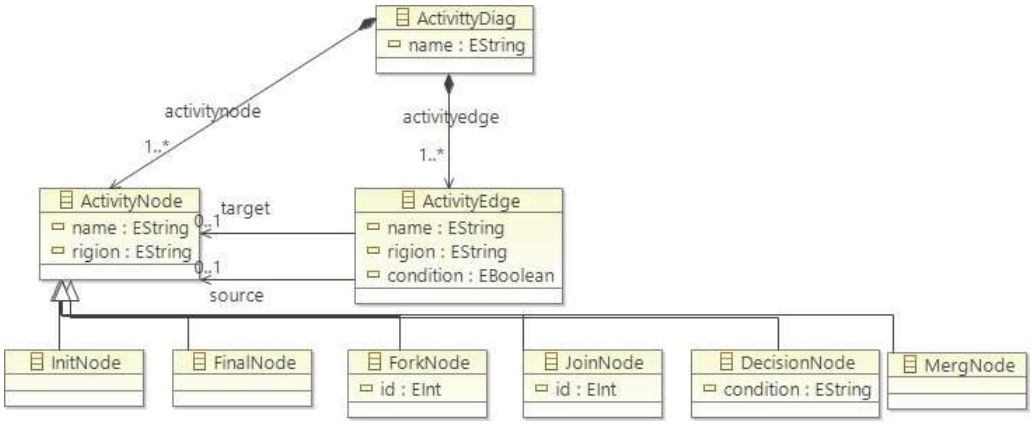


Figure 0-3 : Métamodèle de diagramme d'activité.

4.3.1.2 Métamodèle de réseau de Petri :

Le métamodèle de réseau de Petri définit les éléments de base d'un réseau de Petri, tels que les places, les transitions, les jetons et les arcs. Il décrit également les règles de syntaxe et de sémantique pour la construction de réseaux de Petri.

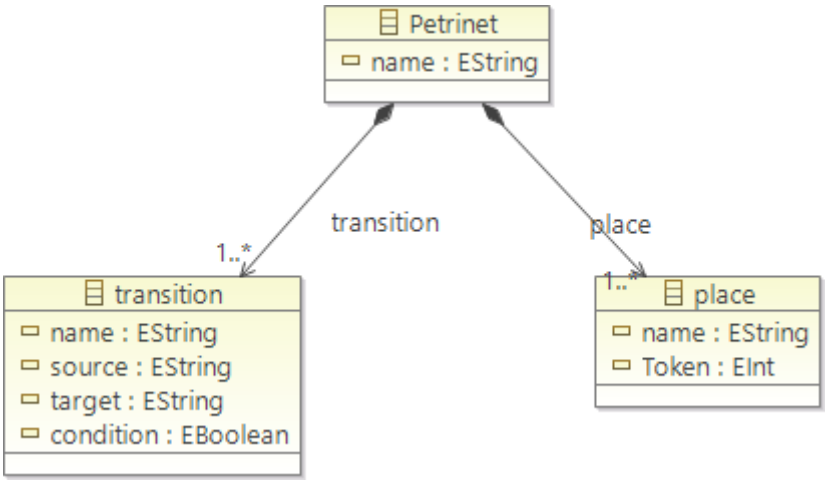


Figure 0-4 : Métamodèle réseau de petri.

4.3.1.3 Métamodèle de correspondance :

Un métamodèle de correspondance est une représentation abstraite qui décrit les relations entre les modèles de différentes sources. Il permet de modéliser les correspondances et les transformations entre les modèles, souvent utilisé dans les processus de transformation de modèle.

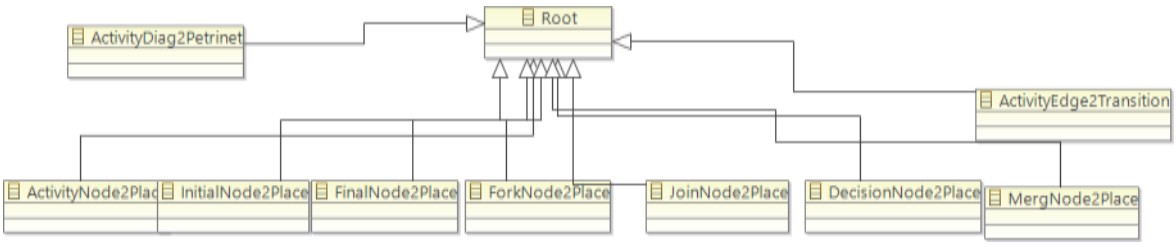


Figure 0-5 : Métamodèle de correspondance.

4.3.2 Génération des outils pour la transformation

En se basant sur le méta-modèle on peut générer un outil qui nous permettra de créer des exemples de diagramme d'activité (des instances) avec les étapes suivantes :

- Nous allons créer un projet EMF (RDP), puis un diagramme Ecore avec leur nom (ActivityDiag) dans le dossier (Model).

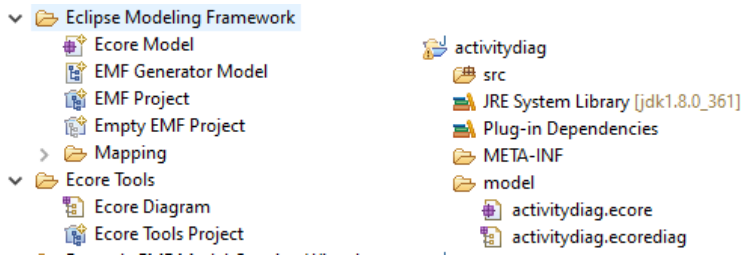


Figure 0-6 : Création d'un projet EMF et un diagramme Ecore.

- Les éléments de notre méta-modèle de réaux de pétri sont affichés dans le fichier (PetriNet.ecore) (**figure 4.7**).

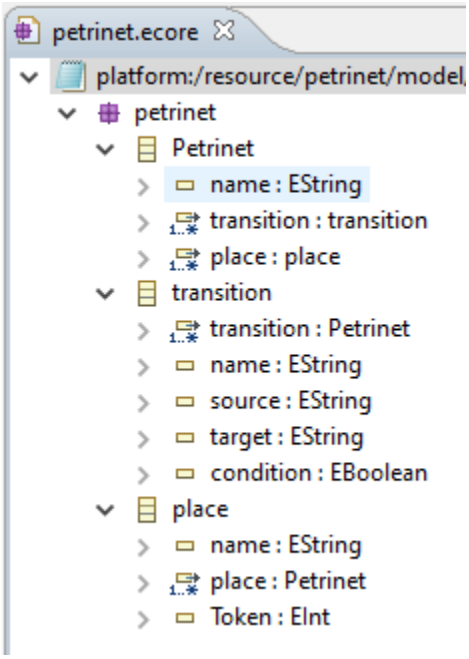


Figure 0-7: éléments de RDP.

- Dans le projet EMF nommé (Correspondance), dans le dossier model nous allons présenter le schéma de méta-model de correspondante (Correspondance.ecorediag), le fichier (Correspondance.ecore) s'affichera automatiquement, mais nous devrons saisir les attributs et leur type pour chaque règle de transformation

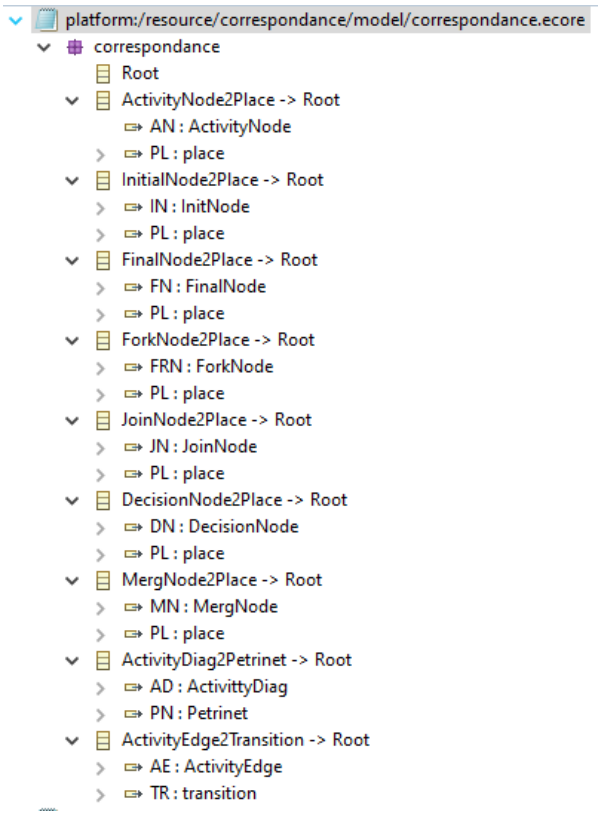


Figure 0-8 : Correspondence.ecore.

4.3.3 Définition des règles de TGG

Les règles de TGG (Transformationnel Graph Grammaire) sont des règles formelles qui décrivent comment transformer un modèle source en un modèle cible à l'aide d'une approche de transformation de modèle bidirectionnelle. Elles sont utilisées pour décrire les correspondances et les transformations entre deux modèles différents.

Les règles de TGG sont généralement composées de deux parties : une partie pour la transformation directe (Forward Transformation Rule - FTR) qui décrit comment les éléments du modèle source peuvent être transformés en éléments du modèle cible, et une partie pour la transformation inverse (Backward Transformation Rule - BTR) qui décrit comment les éléments du modèle cible peuvent être transformés en éléments du modèle source.

4.3.3.1 Règle diagActivity2RDPetri_daigram (Axiome) :

C'est le point de départ de toute transformation, nommé l'axiome dans TGG. Cet axiome est illustré dans (la figure 4-9). Sur le côté gauche, on trouve l'objet racine du

diagramme d'activité, sur le côté droit, on trouve l'objet racine d'un réseau de pétri, et dans la partie médiane, on trouve un nœud de correspondance relative des deux.

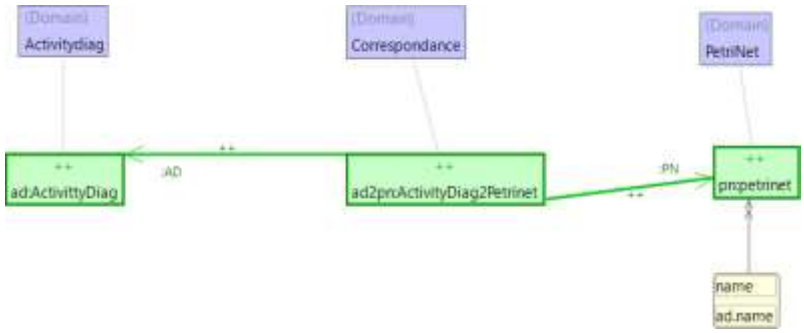


Figure 0-9 : Présentation graphique de l'axiome (diagActivity2RDPetri).

4.3.3.2 RègleInitialNode2place

Chaque nœud initial se transformer à une place dans le réseau de pétri.

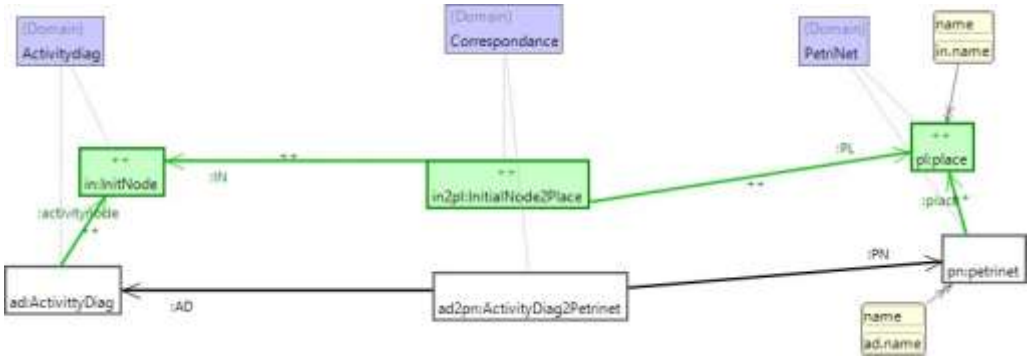


Figure 0-10 : Règle InitialNode2place.

4.3.3.3 Règle FinalNode2place

Chaque nœud initial se transformer à une place dans le réseau de petri

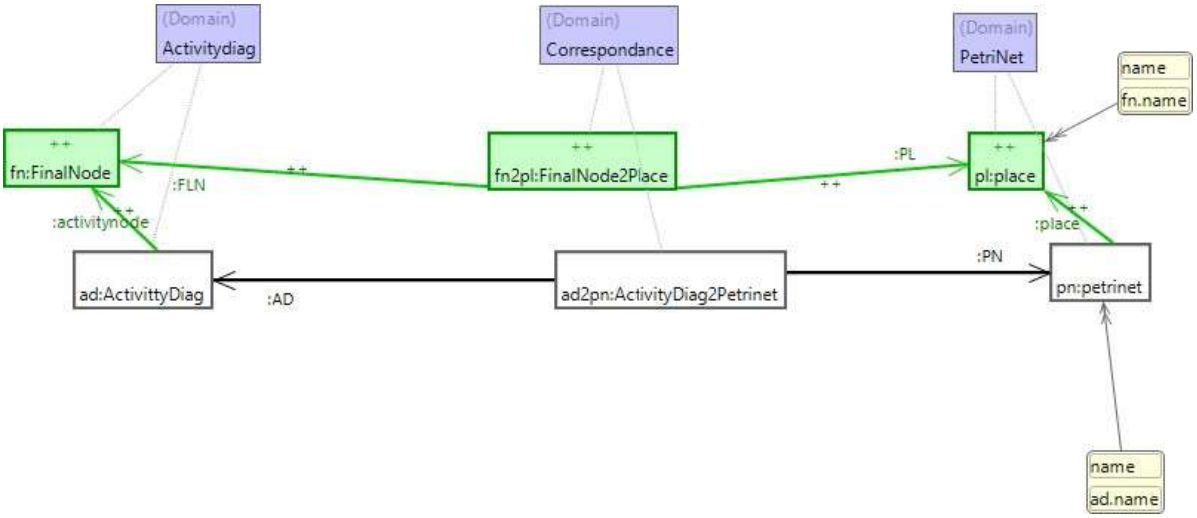


Figure 0-11 : Règle FinalNode2place.

4.3.3.4 Règle ForkNode2Place

Chaque nœud de bifurcation se transformer à une place dans le réseau de pétri.

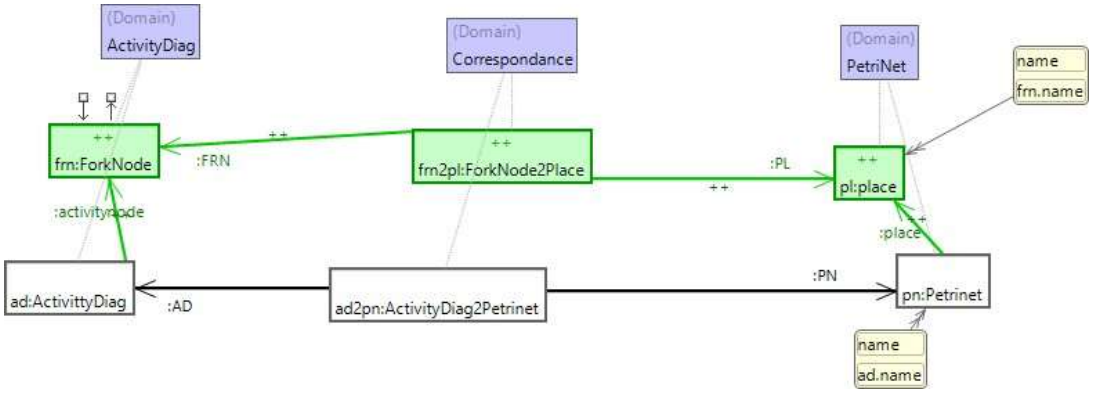


Figure 0-12 : Règle ForkNode2place.

4.3.3.5 Règle JoinNode2Place

Chaque nœud de synchronisation se transformer à une place dans le réseau de pétri.

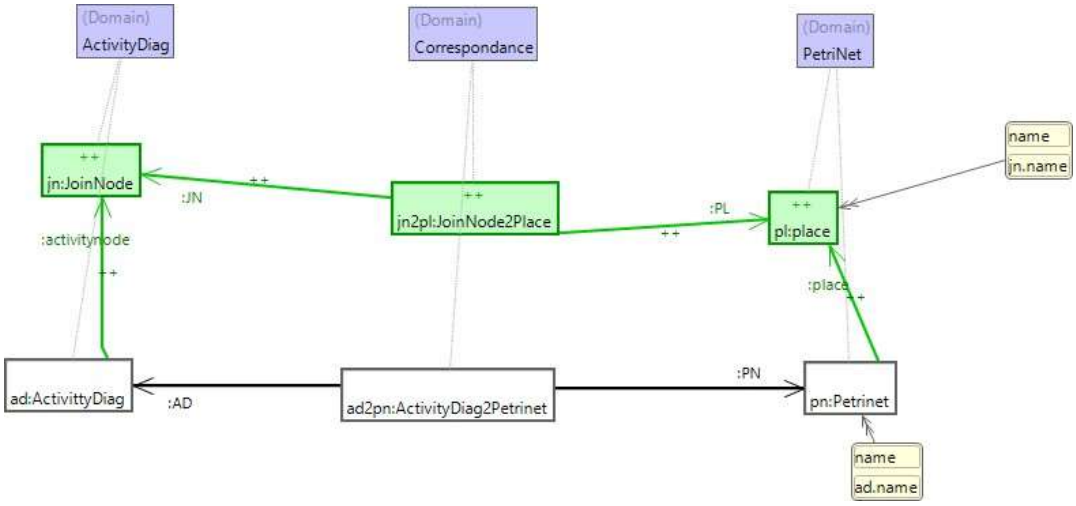


Figure 0-13 : Règle JoinNode2place

4.3.3.6 Règle DecisionNode2place

Chaque nœud de décision se transformer à une place de réseau de pétri.

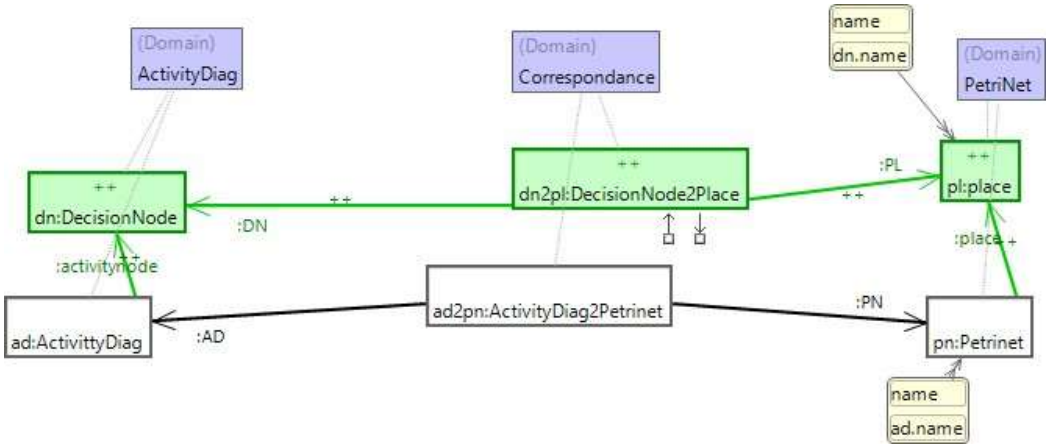


Figure 0-14 : Règle DecitionNode2place.

4.3.3.7 Règle MergeNode2place

Chaque nœud de fusion se transformer à une place dans le réseau de pétri

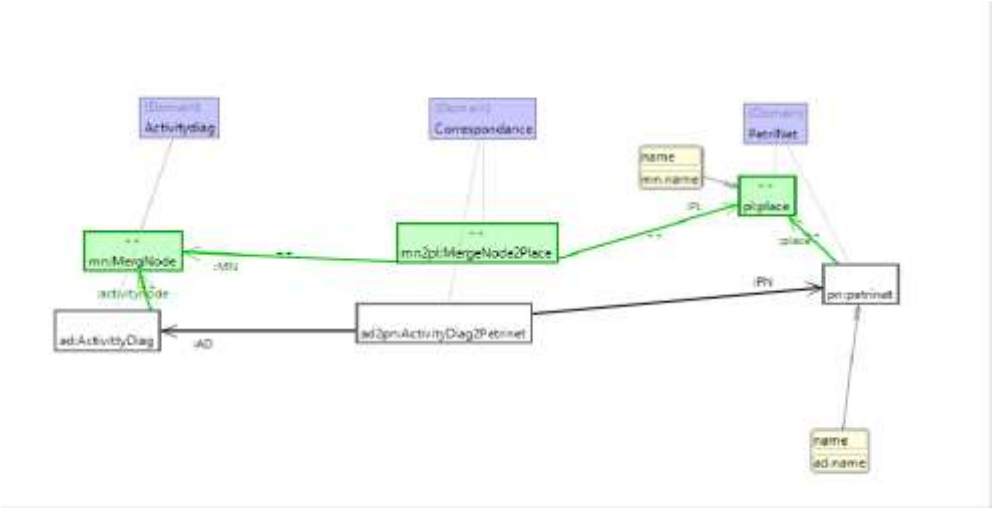


Figure 0-15 : Règle MergeNode2place.

4.3.3.8 Règle ActivityEdge2Transition

Chaque flèche de flux se transformer à une transition dans le réseau de pétri

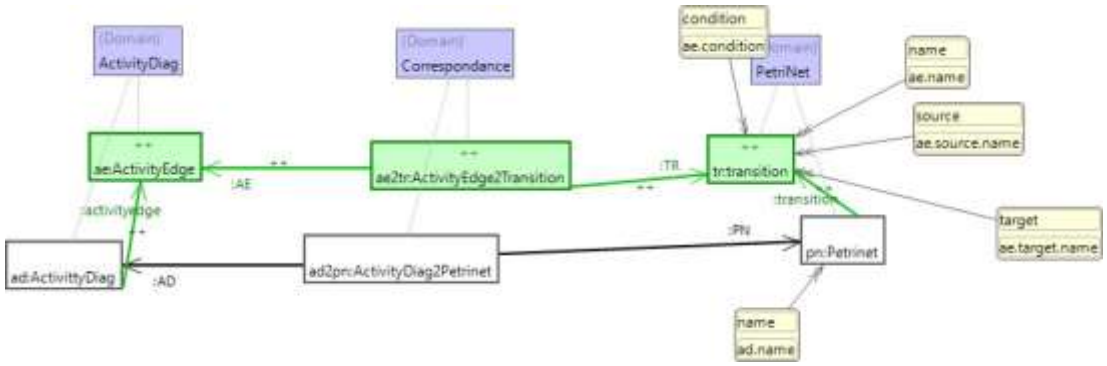


Figure 0-16 : Règle ActivityEdge2Transition.

4.3.3.9 Règle Activity2Place

Chaque activité se transformer à une place dans le réseau de pétri

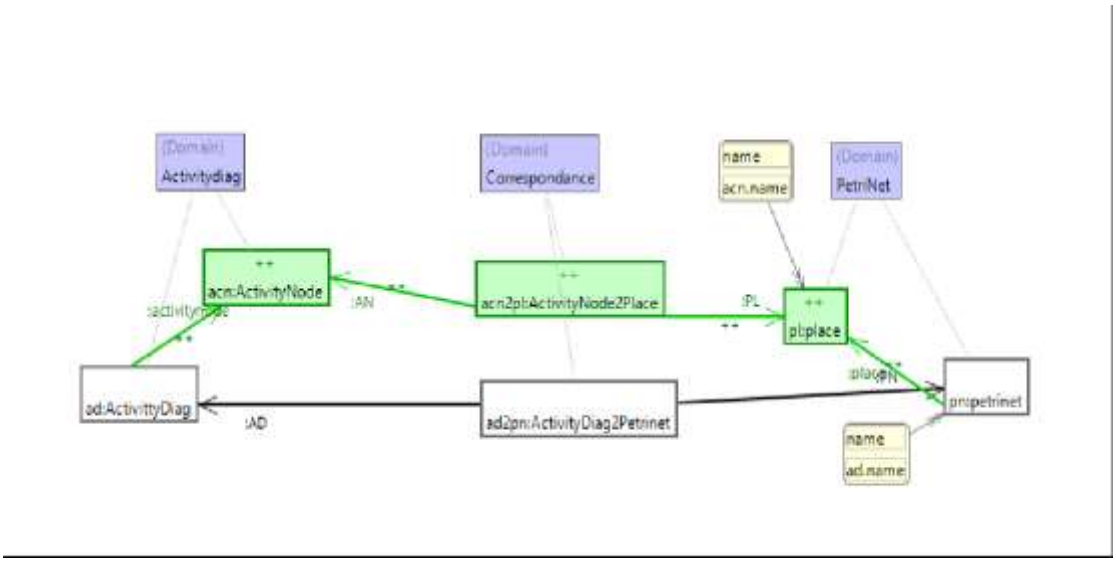


Figure 0-17 : Règle ActivityNode2place.

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 Generator Model le nom (source.genmodel) dans le dossier model, ensuite nous allons générer les projets (activitydiag.edit/activitydiag.editor/ activity diag.tests). Nous ferons de même pour le réseau de petri : génération des projets.edit/.editor/.tests

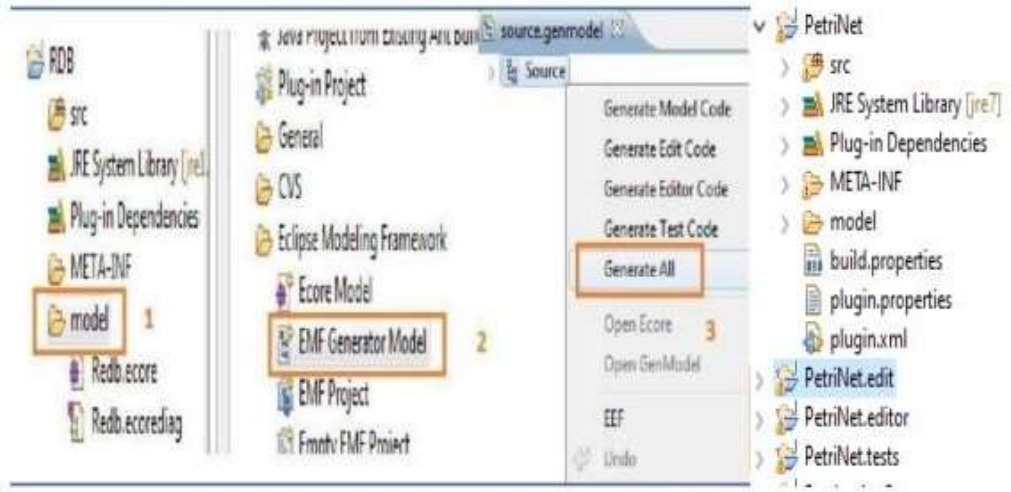


Figure 0-18 : génération des projets.edit/.editor/.tests

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 modèle Les éléments de source.genmodel/destination.genmodel.

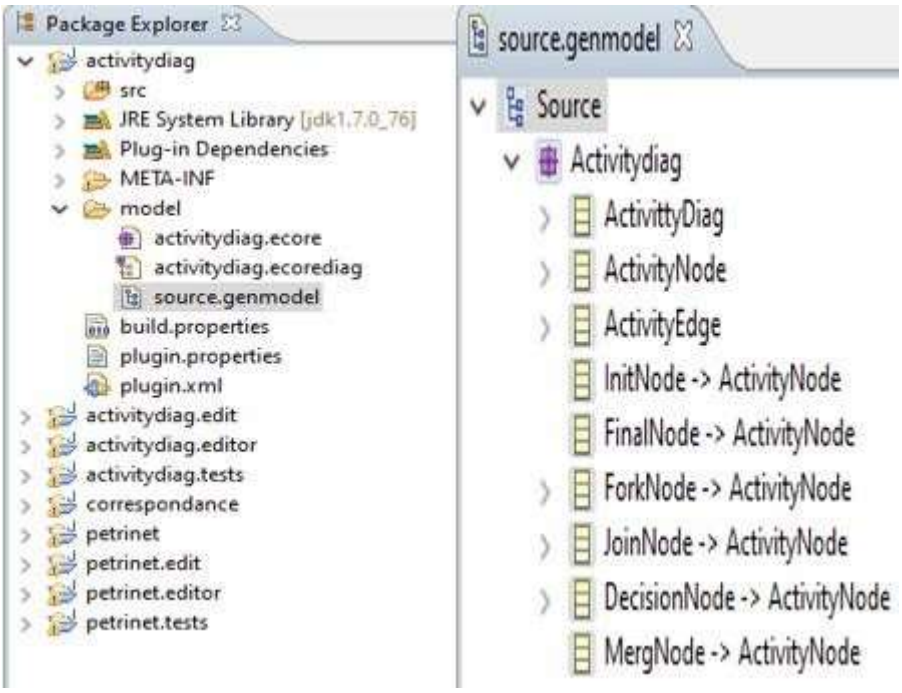


Figure 0-19 : Les éléments de source.genmodel.

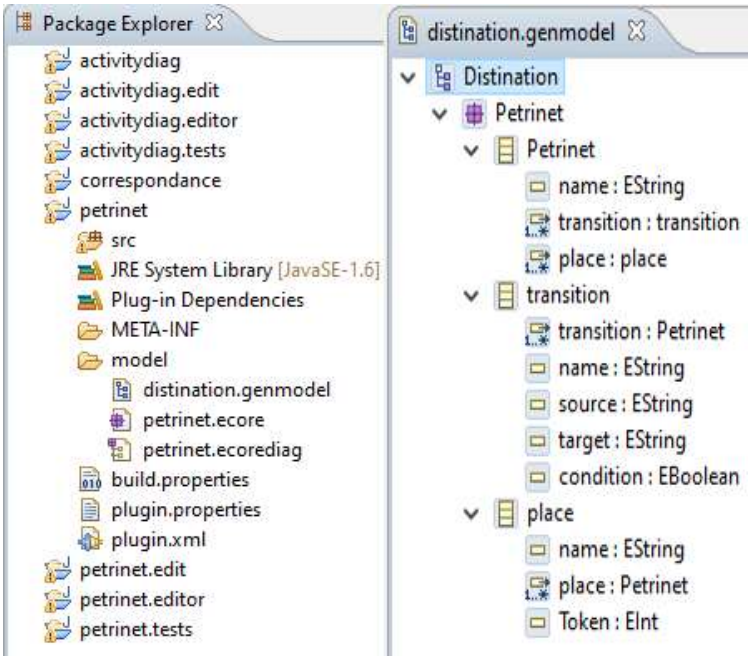


Figure 0-20 : Les éléments de destination genmodel.

4.4 L'exécution

La phase d'exécution dans TGG Interpréter 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 générée.

4.4.1 Le modèle d'entrée :

On a pris un exemple simple d'un diagramme d'activité (login process)

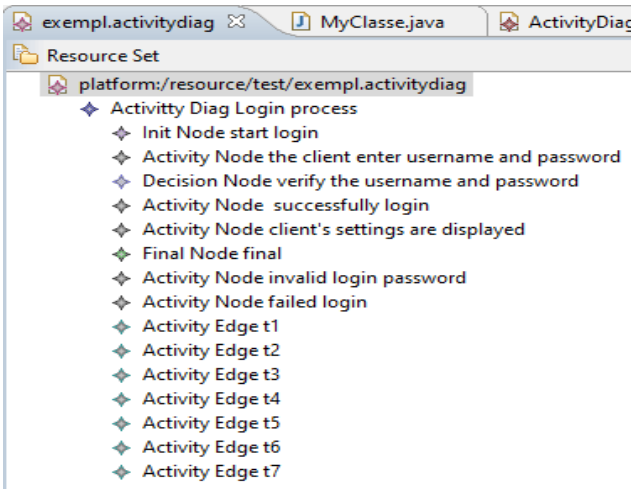


Figure 0-21: Exemple de modèle de diagramme d'activité.

4.4.2 Appliquer les règles

On va lancer une seconde instance d'Eclipse appliquée au projet source (activitydiag), puis on va créer un projet appelé (test) contenant le fichier (exempl) pour réaliser un exemple du modèle source qui à son tour se transforme en modèle cible, dans le dossier (test) on va créer un fichier (exempl.interpreterconfiguration) avec lequel nous faisons le processus de transformation et une fenêtre de fin de transformation sera afficher (voir Figure 0-22).

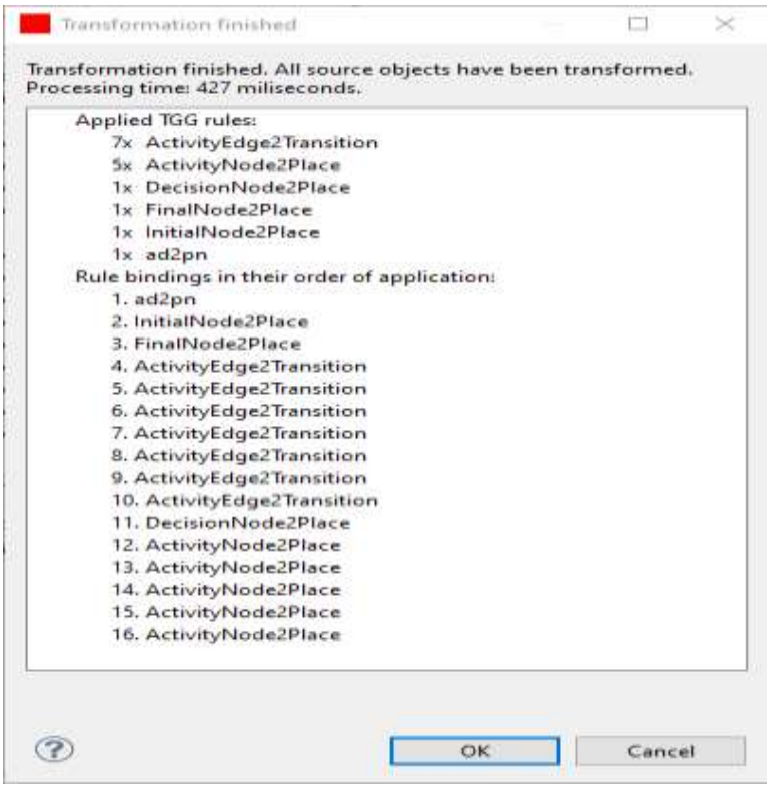


Figure 0-22: Résultat de fin de transformation.

4.4.3 Le modèle de sortie

Après avoir exécuter notre moteur de transformation on a eu ce réseau de pétri. **La figure 4-3** présente le modèle cible de la transformation.

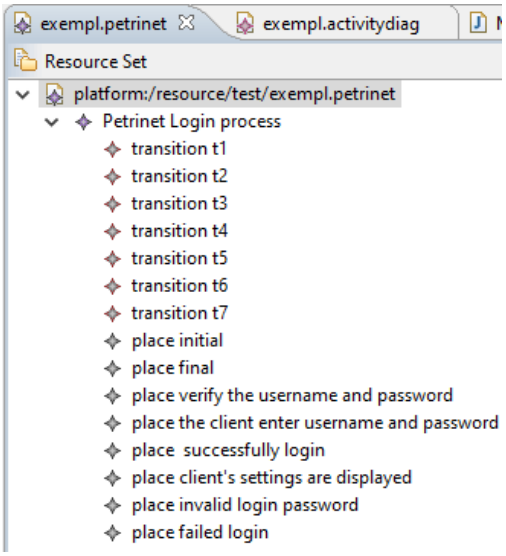


Figure 0-23: réseau de petri cible.

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 textuel, 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 chk pour définit Quelques contraintes qui ont 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'une 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 »

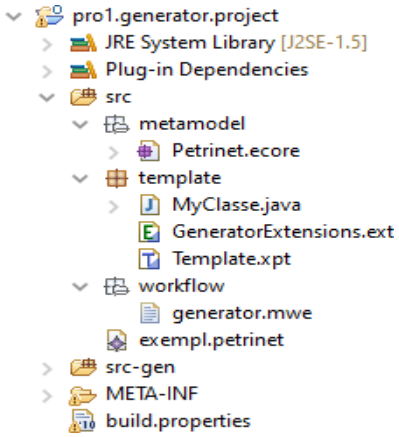


Figure 0-24 : les trois packages (métamodèle, Template, workflow).

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.

4.6 Transformations de modèle à texte (M2T)

Le projet Model-to-Texte (M2T) se concentre sur modèle [65]. Une grande classe de transformations convertit les modèles en texte. Le texte peut être du code généré, d'autres modèles dans la grammaire du texte ou d'autres objets texte, tels que des rapports ou des documents. Le texte est généralement généré à l'aide de modèles, qui est une technologie très mature (par exemple dans le développement Web). Les modèles peuvent être considérés comme du texte cible avec des trous pour les parties variables. Ces trous contiennent du métacode (et donc du code de création de code), qui est exécuté à l'instanciation du modèle pour calculer les parties variables.

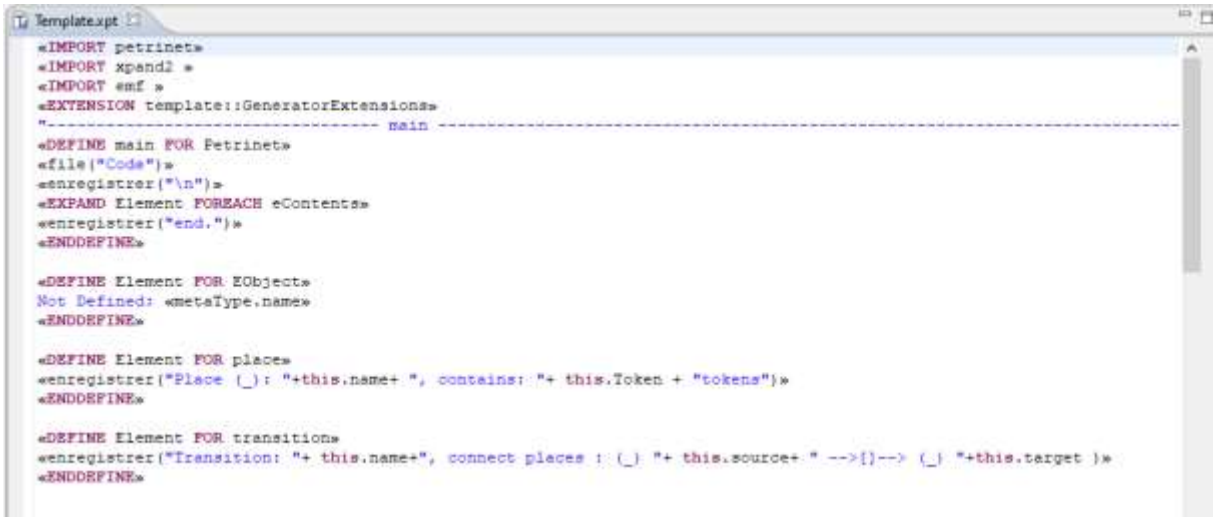


Figure 0-25 : Le Template de génération de code.

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 0-26 : Template Xpand pour la génération de code RDP (Myclass.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.

Figure 0-27 : résultat de la transformation M2T.

4.7 Conclusion :

Dans ce chapitre, nous avons proposé une approche basée sur les modèles pour la transformation de modèles basée sur l'utilisation de grammaires de graphes. Nous avons réalisé une étude de cas sur les transformations de graphes à l'aide de Triple Graph Grammars (TGG) pour maîtriser la transition à partir d'un diagramme d'état UML. Des modèles aux réseaux de Petri, la mise en œuvre de cette approche nécessite l'utilisation d'environnements dédiés, comme le Eclipse Modeling Framework (EMF), qui nous a permis de mettre en œuvre des métamodèles et des transformations. En particulier, nos travaux ont porté sur la spécification des transformations utilisant le formalisme TGG et leur implémentation à l'aide de l'outil interpréteur TGG intégré à l'environnement EMF pour définir les règles de la transformation.

Conclusion générale

A travers ce mémoire, notre objectif était de proposer une approche de transformation de diagramme d'activité vers réseaux de Petri. On utilise la combinaison entre la méta-modélisation et la transformation de graphe. Pour répondre à cette tendance, l'ingénierie des modèles apparaît comme une évolution prometteuse des techniques de génie logiciel. Cependant, le succès d'une approche de développement est conditionné par l'existence de techniques permettant d'assurer la qualité du logiciel produit. Comme nous l'avons montré dans ce mémoire, le fait d'utiliser les modèles de manière productive nécessite de disposer d'environnements de modélisation bien formalisés et de techniques pour valider les programmes de transformation de modèles.

Le premier chapitre été consacré pour discuter l'approche MDA ainsi que le processus de transformation des différents modèles, finalement on a nommé quelques outils pour la mise en œuvre de cette dernière.

Dans le deuxième et le troisième chapitre nous avons présenté le diagramme d'activité et le réseau de petri

Le dernier chapitre, détail la partie pratique de notre travail, de la modélisation des méta-modèles de diagramme d'activité et de réseau de petri jusqu'à l'exécution et la génération du code.

Le résultat de notre travail est une approche automatique pour transformer diagramme d'activité vers Rdp « Réseaux de pétri ». L'approche proposée est basée sur la transformation de graphes, et elle est réalisée à l'aide de l'outil éclipse Le travail est réalisé dans deux étapes :

- La première étape consiste à proposer deux méta-modèles : diagramme d'activité et réseau de petri.
- La deuxième étape : propose une grammaire de graphe permettant de transformer le diagramme d'activité vers un réseau de petri.

Finalement, afin d'arriver à développer une approche de transformation automatique de diagramme, nous proposerons «TGG» et l'outil éclipse.

[1] ["Model-Driven Engineering" par Richard F. Paige, Jean-Michel Bruel, Marsha Chechik, et Betty H.C. Cheng.]

[2] DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE Ingénierie dirigée par les modèles d'un pilotage robuste de la prise en charge médicamenteuse Présentée et 52 soutenue par : Mme RAFIKA THABET le vendredi 23 octobre 2020

[3] site web IDM (accédé le 8/2/2023) https://www.modeliosoft.com/fr/technologies/uml.html

[4] site web Transformation de graphes https://123dok.net/document/nzwpn0qeapproche-specification-changements-besoins-basee-transformations-graphes.html

[5] https://www.journaldunet.fr/web-tech/dictionnaire-duwebmastering/1445294-idm-logiciel-presentation-et-fonctionnement-technique/ (accédé le :18/02/2023)

[6] : M.Amroune ,(2014).vers une approche orientée aspect d'ingénierie des besoins dans les organisations multi-entreprise ,thèse de doctorat,UniversitéToulouse 2 Jean Jaurès .

[7] Object Management Group (OMG), UML Specification Version 2.5.1, Section 15.3.3.1 Activity Diagrams.

[8] Smith, J., & Johnson, A. (2022). "Advantages of Activity Diagrams in Process Modeling and Communication." International Conference on Software Engineering, vol. 2, pp. 45-52. DOI[

[9] Martin Fowler, "UML Distilled: A Brief Guide to the Standard Object Modeling Language" (Addison-Wesley Professional, 2003).

[10] L. AUDIBERT. UML 2. 2007-2008.

[11] Ivar Jacobson, Grady Booch, James Rumbaugh, "The Unified Modeling Language User Guide" (Addison-Wesley Professional, 2005)

[12] Benmoussa, Fatima Zohra et Houidi, Farida. Une approche basée sur la transformation du graphe sous ATom3 pour l'intégration des deux outils de réseau de petri. Mémoire de master. Université Echahide Hamma Lakhdar, EL-OUED. Pp :4-6.

[13] Amira Dardour. Estimation et diagnostic de réseaux de Petri partiellement observables. PhD thesis, Université d'Angers ; École nationale d'ingénieurs de Sfax (Tunisie), 2018.

[14] T. Murata, "Petri nets : Properties, analysis and applications," Proceedings of the IEEE, vol. 77, pp. 541–580, April 1989.

[15] R. David and H. Alla, Discrete, Continuous, and Hybrid Petri Nets. Springer, 1 ed., 23 November 2004.

[16] Bahri, Mohamed Rédha.(2011).Une approche intégrée Mobile-UML/Réseaux de Petri pour l'Analyse des systèmes distribués à base d'agents mobiles. Thèse de doctorat, Université

de Mentouri, Constantine.pp : 55-71.

[17] "Petri Nets: Properties, Analysis and Applications" par Giorgio De Michelis et Mieke Massink.

[18] "Modeling and Analysis of Manufacturing Systems Using Petri Nets" de Ping Ji, Mohammed Y. Jaber, et Benoit Montreuil.

[19] Merlin, M. (1991). Timed Petri Nets. Springer.

[20] J. P. Bodeveix, F. Vernadat, and M. Khendek, “Colored Petri Nets,” in Encyclopedia of Software Engineering, J. Marciniak, Ed. John Wiley & Sons, Inc., 2011, pp. 235–242.

[21] "Stochastic Petri Nets: An Introduction to the Theory" de James R. Harries

[22] site web Eclipse Modeling Framework https://www.eclipse.org/modeling/emf/ . (accédé le 7/05/2023)

[23] site web (accédé le 4/5/2023) https://www.encyclopedie.fr/definition/JDK.

[24] MEMOIRE DE MAGISTER (UNE APPROCHE HYBRIDE POUR TRANSFORMER LES MODELES) Présenté par MECHERI NACERA Soutenue le 27 Octobre 2015.

[25] MEMOIRE DE MAGISTER (Génération d’une BDD-NoSQL à partir d’une BDD relationnelle : Approche basée sur la transformation de graphe) Présenté par Brahimi Khel en 2020.