REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE IBN KHALDOUN - TIARET

MEMOIRE

Présenté à :

FACULTÉ DES MATHEMATIQUES ET DE L'INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

MASTER

Spécialité : Génie informatique

Par :

ABDOU Islam Bilal
BENAMARA Ramzi

Sur le thème

Transformation de Modèle Comportemental À L'aide D'exemples

Application sur les jeux reéactifs

Soutenu publiquement le …/…2022 à Tiaret devant le jury composé de

Mr. BEKKAR KHALED	Grade University MAA	Président
Mr. BENGHENI ABDELMALEK	Grade University MCB	Examinateur
Mr.SI ABDELHEDI AHMED	Grade University MAA	Encadreur

Résumé

La transformation de modèles (MT) est souvent décrite comme le « cœur et l'âme » de l'ingénierie pilotée par les modèles (MDE). C'est un catalyseur clé pour apporter des capacités de calcul dans MDE. En particulier, le comportement des langages spécifiques au domaine (DSL) peut être facilement capturé et automatisé avec MT. De manière simplifiée, un modèle peut être défini comme un ensemble de données représentant un système et écrites dans un langage bien défini. La transformation des modèles (TM) est une étape primordiale pour l'IDM, elle consiste à transformer d'un modèle source (MS) vers un modèle cible (MC) en se basant sur des règles de transformation , connues appropriées. Nous pouvons citer par exemple le passage de formalise UML vers le modèle relationnel . Ces transformations peuvent être proposées par des experts et des professionnels dans le domaine de transformation des modèles. Dans quelques situations, nous constatons l'absence des règles de transformation pour effectuer un certain nombre d'actions. Notre objectif consiste à déduire des règles de transformation en se basant sur l'historique des transformations effectuées sur les modèles. Cette situation de l'absence d'un dataset, nous a motivé de voir la possibilité de proposer une approche de transformation inspirée de l'apprentissage par renforcement (*Reinforcement Learning*) qui raffine au fur et à mesure les règles de transformation. Ce type d'apprentissage est une technique qui permet à un agent (par exemple Pacman) d'apprendre dans un environnement interactif par essais et erreurs en utilisant les feedbacks de ses propres actions et expériences. Nous avons déroulé notre approche sur une étude de cas de jeux de Pacman. Les résultats obtenus motivent notre initiative et adoptent une solution d'apprentissage par renforcement dans le domaine des transformations des modèles.

Mots-clés : Transformation de modèles, Ingénierie dirigée par les modèles, DSL, Métamodèle, Modèle dynamique,Transformation par l'exemple, Apprentissage par Renforcement, DSL Pacman

ملخص

غالبًا ما يوصف تحويل النموذج (MT) على أنه "قلب وروح" الهندسة التي يحركها النموذج.(MDE) إنه حافز رئيسي لجلب قدرات الحوسبة إلىMDE . على وجه الخصوص ، يمكن بسهولة التقاط سلوك اللغات الخاصة بالمجال (DSL) وتشغيله آليًا باستخدامMT . بطريقة مبسطة ، يمكن تعريف النموذج على أنه مجموعة من البيانات التي تمثل نظامًا ومكتوبة بلغة محددة.

يعد تحويل النموذج (TM) خطوة أساسية في MDE ، فهو يتكون من التحول من نموذج المصدر (MS)إلى نموذج مستهدف (MC) بناءً على قواعد التحويل المعروفة المناسبة. يمكننا أن نذكر على سبيل المثال الانتقال من إضفاء الطابع الرسمي على UML إلى النموذج العلائقي. يمكن اقتراح هذه التحولات من قبل الخبراء والمهنيين في مجال تحويل النموذج. في حالات قليلة ، نرى عدم وجود قواعد التحويل لأداء عدد من الإجراءات. هدفنا هو استنتاج قواعد التحويل بناءً على تاريخ التحولات التي أجريت على النماذج. حفزنا هذا الوضع المتمثل في عدم وجود مجموعة بيانات على رؤية إمكانية اقتراح نهج تحول مستوحى من التعلم المعزز (التعلم المعزز) الذي يعمل على تحسين قواعد التحول تدريجيًا. هذا النوع من التعلم هو أسلوب يسمح للوكيل مثل Pacman بالتعلم في بيئة تفاعلية عن طريق التجربة والخطأ باستخدام التغذية الراجعة من أفعاله وتجاربه.لقد طرحنا نهجنا في دراسة حالة لألعاب PACMAN .النتائج التي تم الحصول عليها تحفز مبادرتنا وتبني حل التعلم المعزز في مجال التحولات النموذجية.

الكلمات المفتاحية: تحويل النموذج ، نموذج الهندسة المدفوع ، DSL، Metamodel ، النموذج الديناميكي ، التحول بالمثال ، التعلم المعزز ، DSL Pacman.

Abstract

Model Transformation (MT) is often described as the “heart and soul” of Model-Driven Engineering (MDE). It is a key enabler for bringing computational capabilities in MDE. In particular, the behaviour Domain-Specific Languages (DSLs) can easily be captured, and automated with MT. In a simplifed way, a model can be defined as a set of data representing a system and written in a well-defined language. Model transformation (TM) is an essential step for MDE, it consists of transforming from a source model (MS) to a target model (MC) based on appropriate known transformation rules. We can cite for example the transition from formalizing UML to the relational model. These transformations can be proposed by experts and professionals in the field of model transformation. In a few situations, we see the absence of transformation rules to perform a number of actions. Our objective is to deduce transformation rules based on the history of transformations performed on the models. To design a transformation model based on machine learning algorithms. We rolled out our approach on a Pacman Games case study. The results obtained motivate our initiative to adopt the einforcement Learning solution in the field of model transformations.

Keywords : Model Transformation, Model Driven Engineering, DSL, Metamodel, Dynamic Model, Transformation by Example, Reinforcement Learning, DSL Pacman.

Remerciements

C’est avec grand plaisir que je réserve cette page, en signe de gratitude et de reconnaissance à tous ceux qui m’ont aidé à la réalisation de ce travail.

Nous remercions, tout d’abord, **Siabdelhedi Ahmed** pour sa rigueur et la pertinence de ses jugements qui ont été très constructifs. Nous remercions monsieur **BENGHENI ABDELMALEK** et monsieur **BEKKAR KHALED** de mávoir fait l´honneur d’accepter d’être jury de ce memoire.

nous voudrions exprimer à nos proches toute nos gratitude : nos très chers parents, nos frères et nos soeurs.

Sans leur soutien, leur confiance et leurs encouragements, je n’y serais jamais arrivé.

À nos amis et collègues sans exception, pour leur présence, leur respect et leur gentillesse. Et enfin, merci à tous ceux qui ont participé de près ou de loin à l’aboutissement de ce travail.

À tous ceux qui me sont chers :
Ma mère, mon Père
Mes frères, Mes soeurs.
Islam

À tous ceux qui me sont chers :
Ma mère, mon Père
Mes frères, Mes soeurs.
Ramzi

Table des matières

Liste des figures

Liste des tableaux

Lexique

Liste des vocabulaires utilisés

- **DSL :** Domain Specific Language .
- **IDM :** Ingenierie Derigée Par les Modèles.
- **TMPE :** Transformation de Modèle Par L'Exemple.
- **ATL :** Atlas Transformation Language.
- **PSO :**Particale Swarm Optimozation.
- **AS :**Abstract Syntax.
- **CS :**Concrete Syntax.
- **TM :**Transformation de modèle.
- **LHS :**Left Hand Side.
- **RHS :**Right Hand Side.
- **MS :** Modèle Source.
- **MC :** Modèle Cible.
- **MOF :**Meta Object Facility.
- **EMOF :**Etential Meta Model Facility.
- **CMOF :**Complete Meta model Facility.
- **ATOM :**Domain Specific Visual Languages.
- **AGG :**Algebric Graph Grammar.
- **MDA :**Model driven Architecture.
- **IDE :**Integrated Development Environment.
- **M2M :**Model To Model.
- **M2T :**Model To Text.
- **API :**Application Programming Interface.
- **QVT :**Query View Transformation.
- **EMF :**Eclipse Modeling Framework.
- **FSM :**Finit State Machine.
- **CDS :**Credit Default Swap.
- **OCL :**Object Constraint Language.
- **NLP :**Natural Language Processing.
- **PSO :**Particle Swarm Optimization.
- **MTL :**Model Transformation Language.
- **GPL :**General Purpose language.

- **TP :**True Positive.
- **TN :**True Negative
- **FP :**False Positive.
- **FN :**False Negative.

Première partie

Introduction Générale

Chapitre

1

Introduction Générale

« There is nothing more difficult to take in hand,
more perilous to conduct, or more uncertain in its success,
than to take the lead in the introduction of a new order of things. »
— Niccolo Machiavelli. 1469 -1527

Sommaire

1.1 Contexte et Motivation

L'ingénierie dirigées par des modèles est un vis-à-vis des différentes activités du cycle de développement du logiciel, et d'une manière plus large, pour le génie logiciel. Un modèle à pour role d'éditer des vues, mais aussi pour générer du code, de la documentation et d'analyser et éditer ce dernier. lors du développement on peut avoir plusieurs points de vue c'est à dire beaucoup de modèles, ce qui nécéssite une formalisation des liens explicites entre les différents modèles et une automatisation des transformations. Une transformation est une opération qui prend en entrée des modèles (source) et fournit en sortie des modèles (cibles). Généralement un seul modèle source et un seul modèle cible. Afin de réaliser cette transformation, il existe énormément des formalismes (par ex. –UML, XMI, QVT, SysML, EMF, Ecore), et outils (par ex. –ATL (ATLAS Transformation Language), Kermeta, QVT) proposés dans le cadre de L'ingénierie des modèles.

1.2 Problématique

Nous avons exploré les travaux qui traitent l'apprentissage de transformations dans le cadre de la transformation des modèles par l'exemple (TMPE, en anglais, MTBE : Model Transformation By Example). Les approches MTBE existantes ne résolvent que partiellement le problème de dérivation des règles de transformation. La plupart d'entre eux nécessitent des mappings détaillés (traces de transformations) entre le modèle source et cible, difficiles à fournir dans certaines situations. D'autres peuvent difficilement dériver des règles de transformation qui testent de nombreuses constructions dans le modèle source et/ou produisent de nombreuses constructions dans le modèle cible. Cependant, dans certains domaines comme *la robotique* et *l'automobile*, la connaissance est immature, cette nature de connaissance rend la transformation du modèle plus complexe et on peut trouver plusieurs points de vue. De plus, le manque de données au départ peut être un autre défi qui doit être relevé. Dans ce mémoire, nous abordons ce problème pour générer les règles de transformation en commençant sans jeu de données, avec des règles qui peuvent être affinées lors de l'utilisation du système en cours de conception ou de simulation.

Dans les approches de transformations adoptées par la communauté de l'IDM. Nous avons distingons deux grandes famille (i) Approches basées sur les règles et (ii) Approches basées sur les exemples.

1.3 Objectives

Cette situation de l'absence d'un dataset, nous a motivé de voir la possibilité de proposer une approche de transformation inspirée de l'apprentissage par renforcement (*Reinforcement Learning*) [9] qui raffine au fur et à mesure les règles de transformation. Ce type d'apprentissage est une technique qui permet à un agent (par exemple Pacman) d'apprendre dans un environnement interactif par essais et erreurs en utilisant les feedbacks de ses propres actions et expériences.

Apprentissage par renforcement est une classe d'algorithmes d'apprentissage automatique (*machine learning*) qui apprennent aux modeles des solutions à partir des erreurs dans lesquels il y a peu de rétroaction (*feedback*). Le processus d'apprentissage est piloté par un agent qui exécute des actions dans

son environnement ; et traite ensuite la rétroaction de l'environnement : le nouvel état de l'agent, et un signal de récompense (*reward signal*).

1.4 Organisation du Mémoire

Notre mémoire est organisé en quatre chapitres :

- Chapitre 1 : La transformation des Modèles
- Chapitre 2 : La transformation des Modèles par l'exemple.
- Chapitre 3 : Présentation de notre approche
- Chapitre 4 : L'implémentation et la mise en oeuvre de notre application
- Enfin, la dernière partie présente une synthèse de notre proposition, des limites et des perspectives.

Deuxième partie

État de l’art

Chapitre

2

Transformation des modèles

« Nous ne sommes pas ici dans ce monde pour transformer les autres, mais pour nous transformer nous-mêmes. »

— Léon Tolstoï

Sommaire

2.1 Introduction

L'ingénierie dirigée par les modèles (IDM) [29] est une discipline relativement nouvelle, qui vise à faire une abstraction sur développement logiciel, de la technologie de mise en oeuvre et de réduire sa complexité en déplaçant l'attention de la phase de codage vers la phase de modélisation. Dans cette vue centrée sur le modèle, les modèles et transformations de modèles deviennent des citoyens de première classe : les modèles peuvent être vus comme différentes représentations du logiciel tandis que les transformations de modèles peuvent être considérées comme le mécanisme de collage entre les modèles. Ainsi, à partir d'un modèle, et au moyen de transformations de modèle, il est possible d'obtenir automatiquement une variété d'autres artefacts. En particulier, les transformations de modèles peuvent être exploitées pour une grande variété de tâches, non limitées à la génération de code source, telles que l'analyse, les tests, la vérification de la cohérence des modèles et la synchronisation des modèles.

Dans ce chapitre, nous allons présenter les généralités sur la transformation des modèles.

2.2 Concepts de base

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

2.2.1 Ingénierie Dirigée par les modèles

L' IDM est une approche de développement mettant à disposition des outils, des concepts et des langages afin de simplifier et de mieux maitriser le processus de développement de systèmes qui ne cessent de croitre en complexité. D'autre part, elle permet aussi d'augmenter la productivité, la qualité,la réutilisabilité et l'évolution de ces systèmes.

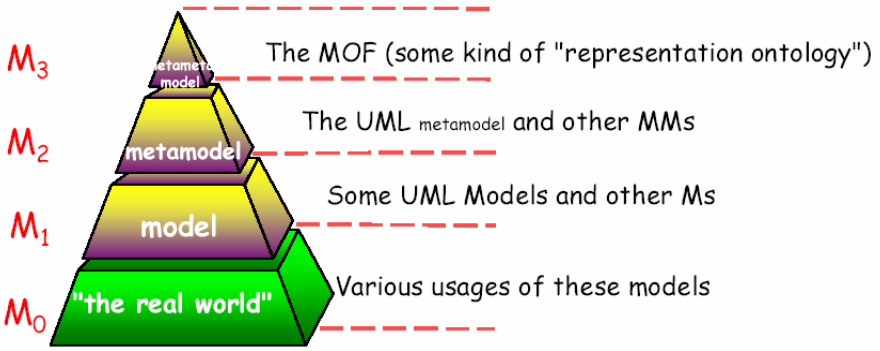

FIGURE 2.1 – L'architecture MOF.

2.2.2 Modèle

Nous avons trouvé dans la literature plusieurs definitions du terme "modèle".

Définition 1

Un modèle est une abstraction d'un système construit dans un but précis[35].

Définition 2

Un modèle est une abstraction qui contient un ensemble limité d'informations sur un système[28].

Définition 3

Un modèle est construit dans un but précis et les informations qu'il contient doivent etre pertinente et à usage spécifique auquel le modèle sera destiné (Muller, 2006).

La Figure 2.2 illustre un exmple d'un modèle UML. Ce modèle est réprésenté sous forme visuelle en utilisant les concepts de classes, d'attributs et d'associations.

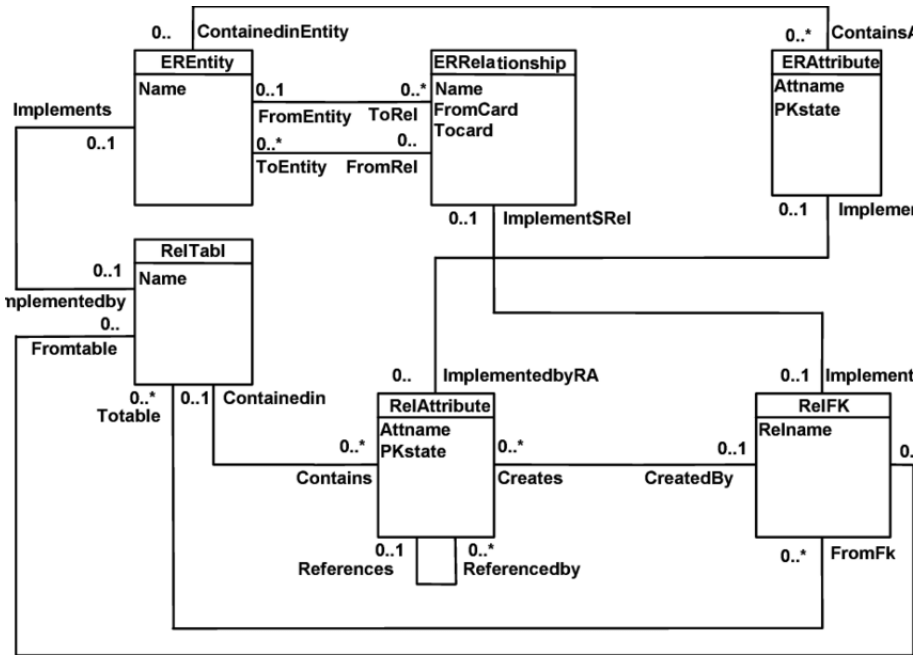

FIGURE 2.2 – Exemple d'un modèle UML.

2.2.3 Métamodèle

Le métamodèle est le langage utilisé pour représenter un modèle.

Définition 4

Selon Kühne[37], le métamodèle définit les types linguistiques et leurs relations. Un métamodèle est défini à l'aide d'un langage de métamodélisation, tel que UML.

Définition 5

En pratique, nous définissons des métamodèles à l'aide de diagrammes de classes UML, où les classes représentent les concepts de type, les attributs représentent leurs propriétés et les associations représentent leurs rapports[13] , la Figure 2.3 illustre les concepts de base de transformation de modèles.

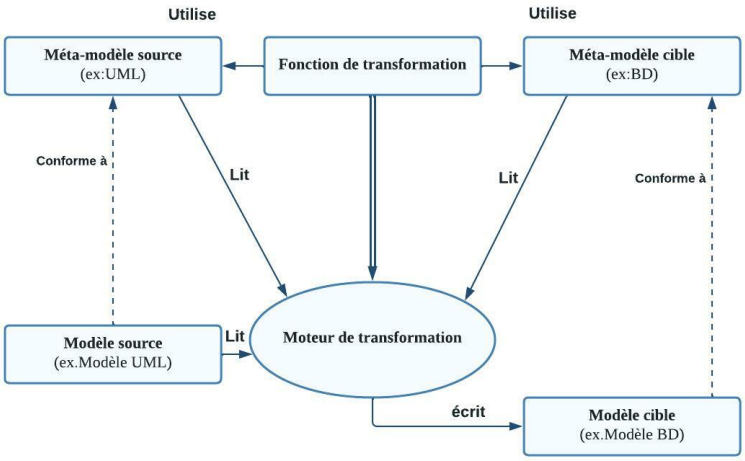

FIGURE 2.3 – Concepts de base de transformation de modèles.

Un métamodèle est un modèle de langage qui décrit des concepts et leurs propriétés, une syntaxe textuelle et/ou graphique et une sémantique. Comme illustrée dans la Figure 2.4. Un DSL peut s'exprimer en trois types de syntaxe :

- **Syntaxe abstraire** : La syntaxe abstraite (AS) d'un langage de modélisation exprime, de manière structurelle, l'ensemble de ses concepts et leurs relations.
- **Syntaxe concrète** : Les syntaxes concrètes (CS) d'un langage fournissent à l'utilisateur un ou plusieurs formalismes, graphiques et/ou textuels, pour manipuler les concepts de la syntaxe abstraite et ainsi en créer des « instances ». Le modèle ainsi obtenu sera conforme à la structure définie par la syntaxe abstraite.
- **Syntaxe semantique** : Définir la sémantique d'un langage ce qui revient à définir le domaine sémantique et le mapping entre la syntaxe abstraite et le domaine sémantique (AS <=> SD). Le domaine sémantique définit l'ensemble des états atteignables par le système, et le mapping permet d'associer ces états aux éléments de la syntaxe abstraite.

2.2.4 Transformations des modèles

Les auteurs fournissent les définitions suivantes de la transformation des modèles :

Définition 6

Les transformations de modèles peuvent être considérées comme le coeur du MDE (ou l'IDM en français) sans une maturité technologique de la transformation, la qualité et l'efficacité de l'IDM ne peut être accordées.

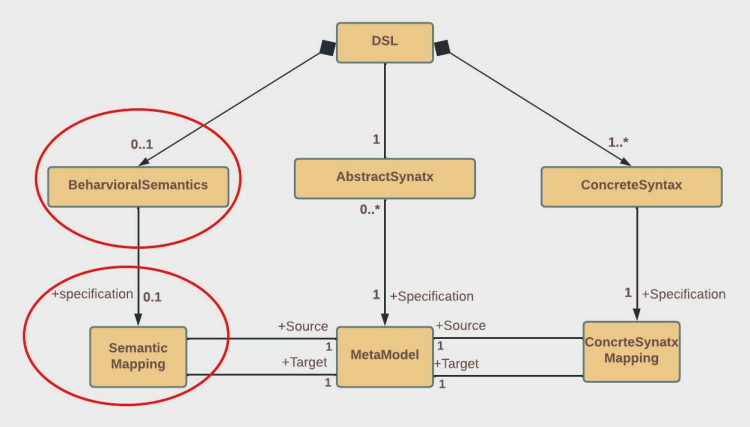

FIGURE 2.4 – Les trois types des syntaxes : abstraire, concrète et sémantique

Définition 7

La génération automatique d'un modèle cible à partir d'un modèle source, selon une définition de transformation [31].

Le processus de transformation des modèles est présentés dans la Figure 2.5.

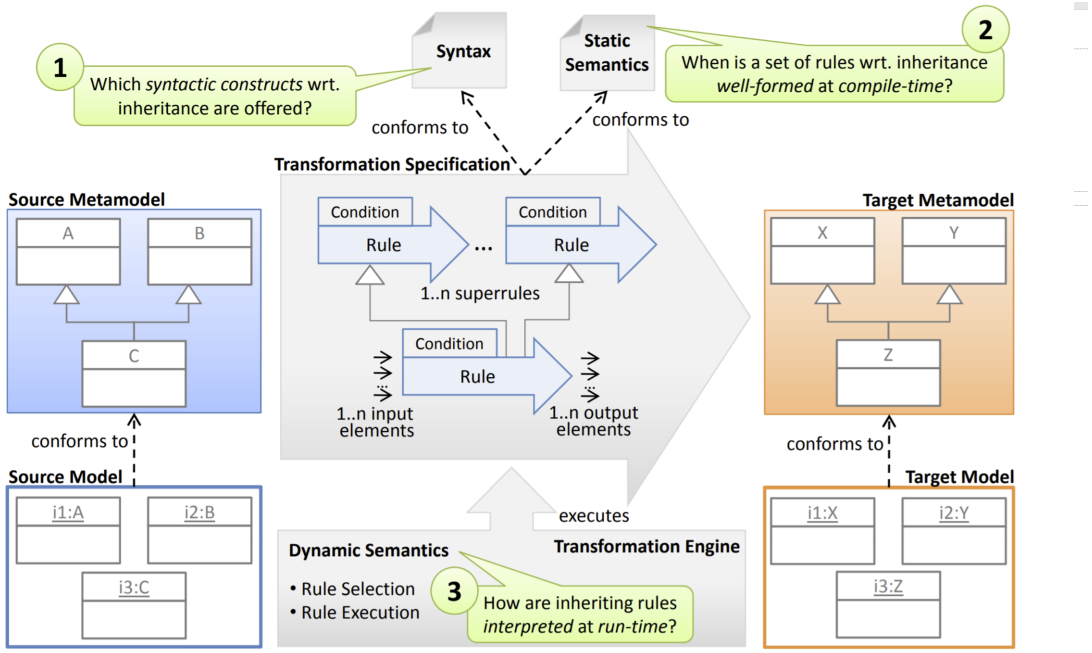

FIGURE 2.5 – Processus de transformation des modèles [42]

2.2.5 Règles de transformation

Définition 8

Les règles de transformation définissent essentiellement un ensemble de mappages entre les éléments du modèle source et les éléments du modèle cible , une fois spécifié la transformation est exécutée à partir de

la transformation moteur en instanciant les règles de transformation sur les modèles source et cible[7].

D'une manière générique, une règle de transformation est une instance de métamodèle présenté dans la Figure 2.7.

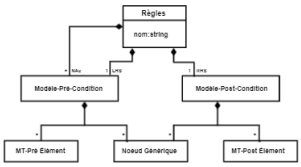

FIGURE 2.6 – Métamodèle d'une règle de transformation.

Une règle de transformation $\mathcal{R}$ est definie par un tuple $\mathcal{R}$ = (NAC, LHS, RHS), où LHS (Left Hand Side) et RHS (Right Hand Side) sont les graphes typés appelés graphes de gauche et graphe de droite de la règle, et (NAC : Negative application condition) représente un ensemble (potentiellement vide) de graphes typés appelés les conditions d'application négatives.

la Figure 2.7 représente un exemple de règle de transformation. La partie haute de la Figure 2.7 représente les trois composantes de la règle : NAC, LHS et RHS. La partie basse de la Figure 2.7 représente un exemple d'une règle de transformation qui concerne un jeu intéractif de deux dimensions (jeux pacman)

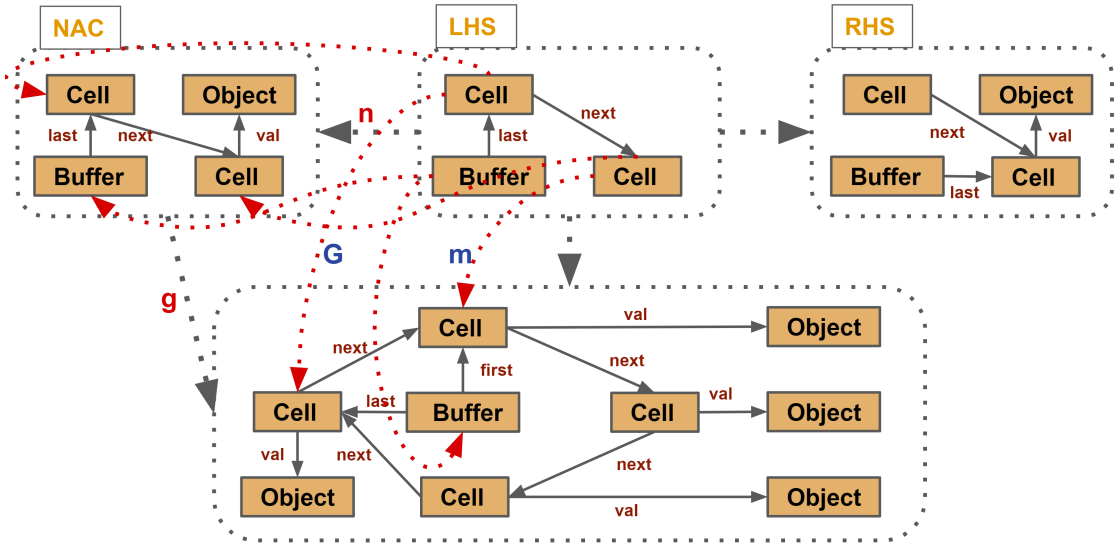

FIGURE 2.7 – Example d'une règle de transformation : cas de jeux pacman

Pour synthétiser :

- **LHS,NAC :** Sont les règles de la préconditon qui doivent être vérifier avant l'application de la règle (le modèle cible).
- **RHS (Right-Hand-Side) :** Sont les règles de la postcondition qui doivent être vérifiées dans le modèle cible .

2.2.6 DSL sémantique (xDSML)

Un DSL sémantique (xDSML) est défini par :
— Une syntaxe abstraite, c'est-à-dire un métamodèle.
— Sémantique opérationnelle, composée de :
  — Un métamodèle d'exécution permet de définir l'exécution des états des modèles en étendant la syntaxe abstraite avec de nouvelles propriétés et métaclasses à l'aide de la fusion de packages ou de tout mécanisme similaire.
  — Une transformation d'initialisation, c'est-à-dire une transformation de modèle exogène qui transforme un modèle conforme à la syntaxe abstraite en un modèle conforme au métamodèle d'exécution, ou il faut préserver au moins le contenu du modèle d'entrée.
  — Une transformation d'exécution, c'est-à-dire une transformation de modèle sur place qui modifie un modèle conforme au métamodèle d'exécution en modifiant les valeurs des champs dynamiques et en créant/détruisant des instances de métaclasses introduites dans le métamodèle d'exécution. Le sous-ensemble de règles de transformation qui sont considérées comme observables sont appelées règles par étapes .

La Figure 2.8 montre un exemple de réseau de Petri . En haut à gauche, sa syntaxe abstraite est représentée avec trois métaclasses Net, Place et Transition. À côté de la syntaxe abstraite, le métamodèle d'exécution est affiché. Il étend la métaclasse Place en utilisant la fusion de packages avec de nouveaux jetons de propriété dynamiques.

La fonction d'initialisation transforme chaque objet d'origine (c'est-à-dire un objet Place sans champ jetons) en un objet exécutable (c'est-à-dire un objet Place avec un champ jetons) tel que défini dans le métamodèle d'exécution. Il initialise également chaque champ de jetons avec la valeur d'initialTokens. En bas, sont représentées les descriptions des règles définies dans la sémantique opérationnelle . Lorsqu'elles sont appelées, ces règles peuvent modifier les champs de jetons des différents objets Place, run étant le point d'entrée de la transformation.

L'étiquette `@Step` est utilisée pour indiquer que les règles de transformation sont des règles d'étape. En étiquetant **run** et **fire** avec `@Step` , nous spécifions que le modèle n'est observable qu'avant et après l'application de ces règles, c'est-à-dire avant et après l'exécution (puisque run est le point d'entrée) et avant et après le déclenchement des transitions. Les jetons des places d'entrée sont ajouter aux jetons des places de sortie, mais seulement après le déclenchement complet d'une transition.

Définition 9

> *Une trace d'exécution est une séquence d'états d'exécution et les étapes d'exécution (petites et grandes étapes) sont responsables des changements d'état du modèle.*

Définition 10

> *Une étape d'exécution est l'application d'une étape régner. Une étape d'exécution qui n'est pas composée d'autres étapes est appelé une petite étape, tandis qu'une étape d'exécution composée de plusieurs étapes s'appelle une grande étape.*

Définition 11

> *Un état d'exécution est l'ensemble des valeurs de tous les champs dynamiques d'un modèle à un certain moment de l'exécution. L'état d'exécution d'un modèle est modifié par l'application des règles de la*

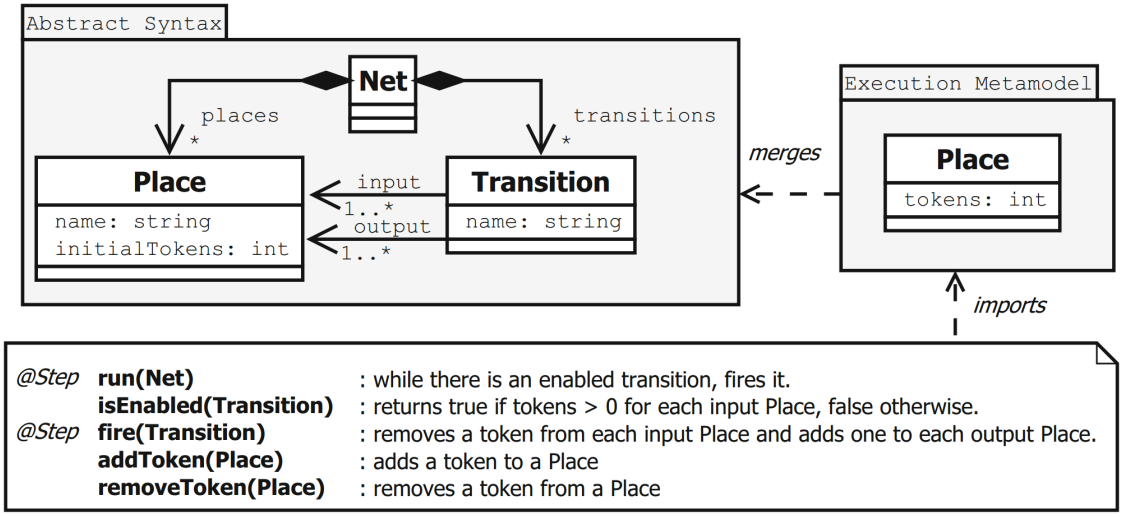

FIGURE 2.8 – Petri net xDSML

transformation d'exécution.

2.2.6.1 Les langages de transformation des modèles.

Dans cette section, nous allons présenter quelques langages de transformation des modèles.

2.2.6.1.1 Le langage de métamodélisation MOF

Le langage MOF (Meta Object Facility) se situe au sommet dans l'architecture de l'OMG à quatre niveaux. C'est un méta-formalisme, pour établir des langages de modélisation permettant eux-mêmes d'exprimer des modèles. Dans sa version 2.0, le méta-métamodèle MOF est constitué de deux parties : EMOF (Essential MOF), pour l'élaboration des métamodèles sans association, et CMOF (Complete MOF) pour l'élaboration des métamodèles avec associations. Il faut souligner que MOF s'auto-décrit pour pouvoir limiter l'architecture de l'OMG à quatre niveaux. Aussi, le langage MOF emprunte la syntaxe du langage de modélisation UML .

2.2.6.1.2 Kermeta :

Il est défini comme un langage de métamodélisation exécutable ou de méta- programmation objet, ce qui signifie qu'il permet de décrire des métamodèles dont les modèles sont exécutables[18]. Kermeta est basé sur le langage de métamodélisation EMOF et il est intégré à l'IDE Eclipse sous la forme d'un plugin. Il a été conçu comme un tissage entre un métamodèle comportemental et EMOF. Le métamodèle de Kermeta est donc composé de deux packages : Core et Behavior. Le premier correspond à EMOF et le second est une hiérarchie de métaclasses représentant des expressions impératives.

2.2.6.1.3 Ecore :

Le langage Ecore a été présenté par la fondation Eclipse[6] dans le cadre du projet EMF (Eclipse Modeling Framework)[10]. Le projet EMF propose un framework pour le développement des applications basées sur l'ingénierie dirigée par les modèles. Ce framework permet de générer automatiquement des interfaces Java à partir des métamodèles Ecore.

2.2.6.1.4 ATOM :

ATOM "Domain Specific Visual Languages" : est un outil pour la con- ception de Domain Specific Visual Languages (DSML). Il permet de définir la syntaxe abstraite et concrète d'un langage visuel au moyen de la métamodélisation et d'exprimer la manipulation du modèle en utilisant la transformation de graphe. Avec les informations de métamodèle, ATOM génère un environnement de modélisation personnalisé pour le langage décrit[22].

Récemment, ATOM a été étendu avec des fonctionnalités pour générer des environnements avec des vues multiples de langages visuels (tels que UML) et Triple Graph Grammar (TGG) . Ce dernier est utile pour exprimer l'évolution de deux modèles différents, liés par un modèle intermédiaire.

2.2.6.1.5 AGG :

AGG " Algebraic Graph Grammar" est un environnement à usage général pour le développement des systèmes de transformation de graphes attribués[34]. Il est basé sur l'approche algébrique pour la transformation de graphes. Il vise à la spécification et le prototypage rapide d'applications complexes tel que le graphe de données structurées.

Puisque la transformation de graphe peut être appliquée à des niveaux d'abstraction très différents, elle peut être attribuée ou non par des calculs simples ou par des processus complexes, selon le niveau d'abstraction. En raison de sa base formelle, AGG offre un support de validation, de vérification de la cohérence des graphes en fonction des contraintes graphiques

2.2.6.1.6 ATL :

Le langage ATL (ATLAS Transformation Language) est défini dans le contexte des langages de transformation par méta-modèles[12] . L'opération de transformation est dirigée par un programme ATL nommé mma2mmb.atl. Ce programme transforme le modèle source Ma en un modèle cible Mb. Il est lui-même un modèle. Les modèles M_a, M_b et le programme sont formes aux méta-modèles MM_a, MM_b et ATL respectivement. Ces métamodèles sont conformes au méta-méta-modèle MOF.

2.2.6.1.7 Acceleo :

Acceleo est un générateur de code source de la fondation Eclipse permettant de mettre en œuvre l'approche MDA (Model driven architecture) pour réaliser des applications à partir de modèles basés

sur EMF[2]. Il s'agit d'une implémentation de la norme MOF Models to Text (MOFM2T) de l'Object Management Group (OMG) pour les transformations de modèles à texte.

2.2.6.1.8 xtend :

Langage de programmation Xtend, un langage de type Java complet étroitement intégré à Java[25].

Xtend a une syntaxe plus concise que Java et fournit des fonctionnalitéssupplémentaires telles que l'inférence de type, les méthodes d'extension et les expressions lambda, sans parler des expressions de modèle multilignes (qui sont utiles lors de l'écriture de générateurs de code).

Tous les aspects d'un DSL implémenté dans Xtext peuvent être implémentés dans Xtend au lieu de Java, car il est plus facile à utiliser et permet d'écrire du code plus lisible. Étant donné que Xtend est complètement interopérable avec Java, vous pouvez réutiliser toutes les bibliothèques Java ; de plus, tous les JDT Eclipse (Java Development Tools) fonctionneront avec Xtend [1].

2.2.6.1.9 Epsilon :

Epsilon est une famille de langages de script basés sur Java pour automatiser les tâches courantes d'ingénierie logicielle basées sur des modèles, telles que la génération de code, la transformation de modèle à modèle et la validation de modèle, qui fonctionnent immédiatement avec EMF (y compris Xtext et Sirius), UML, Simulink, XML et autres types de modèles[11]. Epsilon comprend également des éditeurs et des débogueurs basés sur Eclipse, des outils de réflexion pratiques pour la modélisation textuelle et la visualisation de modèles, ainsi que des tâches Apache Ant. [2]

Après avoir présenté ces langages de transformation des modèles, voici une Figure 2.9 qui montre la variabilité dans la représentation de la règles de la transformation qui peut être textuelle ou visuelle.

2.2.6.2 Les framework de transformation des modèles.

Dans cette section, nous allons présenter Les framework de transformation des modèles.

2.2.6.2.1 Ecore Eclipse :

Pour pouvoir construire des modèles avec un langage de modélisation, des outils sont nécessaires pour éditer et interpréter les modèles. Les ateliers des langages (Fowler, 2005) tel que l'« Eclipse modelling Framework » (EMF), « Metacase » (MetaEdit+), « Generic Modelling Environnement» (GME) ou « outils DSL de Microsoft » réduisent de manière significative les efforts pour la construction des outils supports des langages de modélisation autour des métamodèles.

1. https://www.eclipse.org/Xtend/
2. https://www.eclipse.org/epsilon/

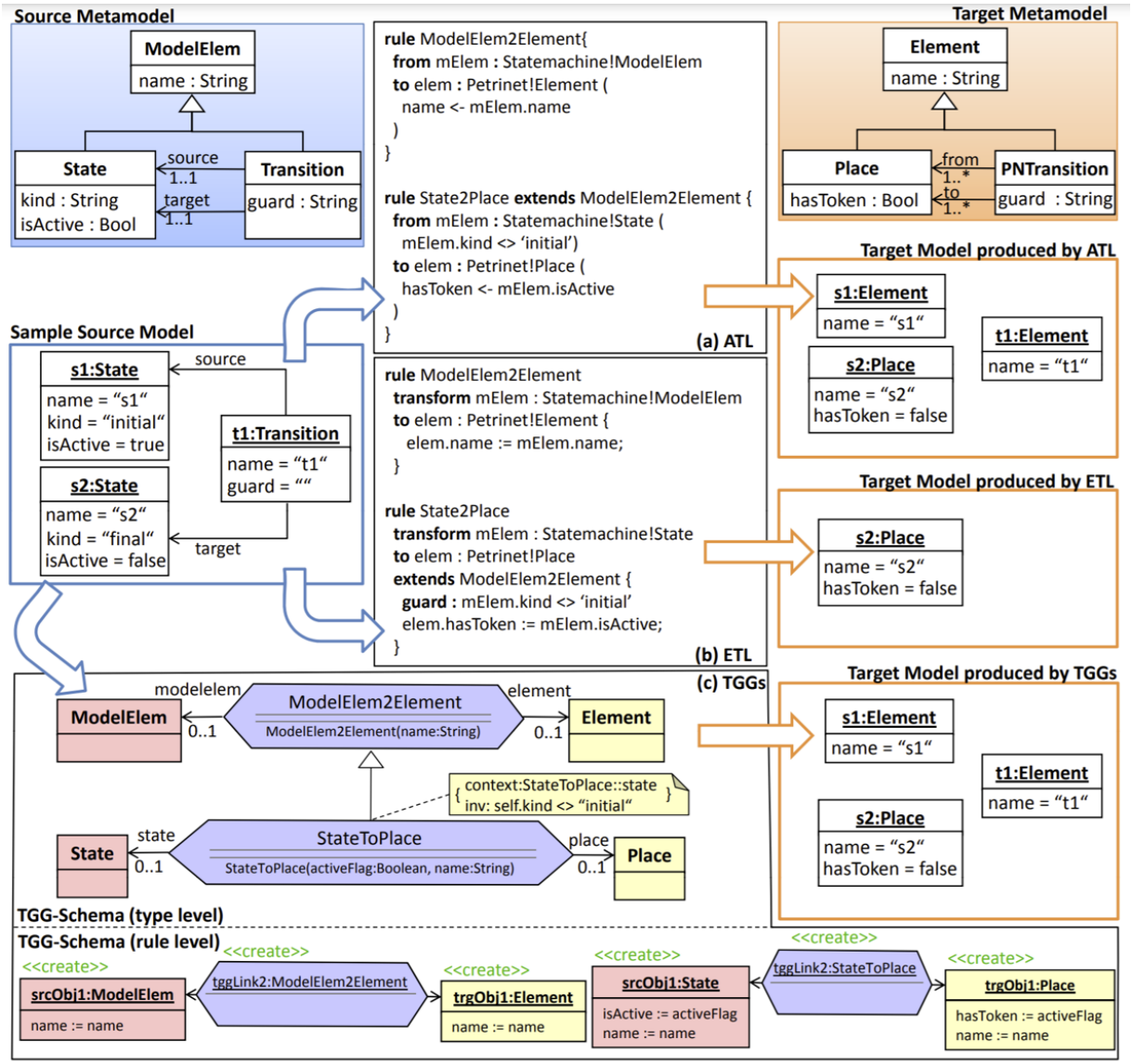

FIGURE 2.9 – Exemple de transformation des modèles avec différente spécification de la règles de la transformation.

Définition 12

EMF représente la plateforme de modélisation la plus utilisée pour l'implémentation de EMOF. Essentiellement le langage Ecore définit un environnement de modélisation EMF basé sur EMOF. Son but est de permettre une génération de code automatique des interfaces et des classes d'un métamodèle.

Définition 13

Le langage Ecore se distingue du langage MOF et EMOF en particulier par deux notions. Les associations ne sont pas représentées mais remplacées par des références directes entre les concepts. Pour générer du code, Ecore intègre également les notions de Java nécessaires par le concept d'EAnnotation. Les métaclasses Ecore sont toutes préfixées par la lettre 'E' (`EClass`, `EAttribute`, `EString`, etc.).

La Figure 5.7 montre un sous-ensemble simplifié du métamodèle Ecore. Les concepts d'un nouveau langage de modélisation sont des EClass qui peuvent définir des propriétés (`EAttribute`), des opérations (`EOperation`) ou des relations (`EReference`) vers d'autres `EClass`. Un des principaux avantages de Ecore est sa simplicité et le grand nombre d'outils disponible. Actuellement Ecore est devenu le standard dans les plateformes de l'ingénierie dirigée par les modèles.

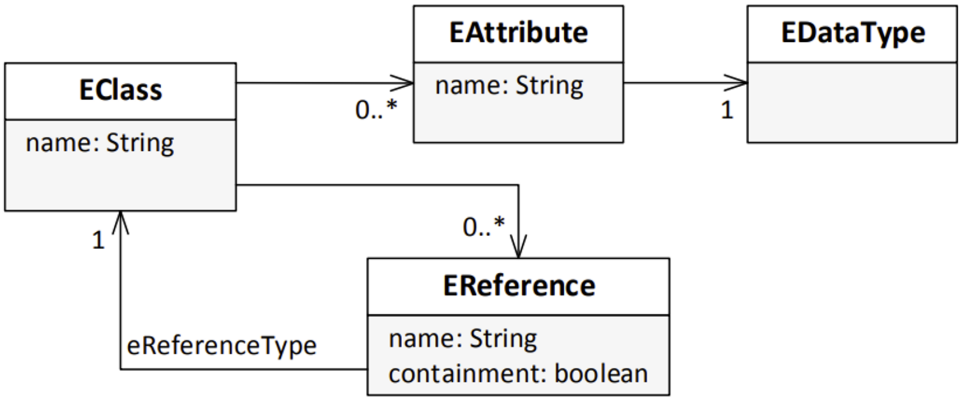

FIGURE 2.10 – le métamodèle ecore

2.2.6.2.2 Xtext :

Xtext [24] permet le développement de la grammaire des langages spécifiques aux domaines (DSL : Domain Specific Languages) et dautres langages textuels en utilisant une grammaire qui ressemble au langage EBNF (Extended Backus-Naur Form).Il est étroitement lié à l'Eclipse Modeling Framework (EMF). Il permet de générer un métamodèle Ecore, un analyseur syntaxique (parser, en anglais) basé sur le générateur ANTLR [3] ou JavaCC [4] et un éditeur de texte sous la plate-forme Eclipse afin de fournir un environnement de développement intégré IDE spécifique au langage. La forme textuelle du DSL constitue le point de départ. La grammaire du DSL est le point d'entrée de l'outil. Xtext produit en sortie un analyseur syntaxique, un éditeur et un méta-modèle pour le DSL.

Xtext permet de considèrer la forme textuelle du DSL comme un modèle conforme à la grammaire d'entrée, donc au méta-modèle généré. La Figure 2.11 ilustre une vue d'ensemble de l'outil Xtext

2.2.7 Quelques techniques de transformation de modèle

Les langages de transformation de modèles peuvent être classés en deux grandes catégories, selon la nature de la production de données : les transformations modèle à modèle (M2M) et modèle à texte (M2T). Dans ce contexte, divers langages de transformation de modèles sont inclus selon les deux catégories susmentionnées, tels que ATL (M2M) et Acceleo (M2T).

3. ANTLR (ANother Tool for Language Recognition) est un générateur puissant pour lire, traiter, exécuter ou traduire du texte structuré ou des fichiers binaires.

4. Compilateur java-compilateur. Il s'agit d'un outil de générateur d'analyseur lexical populaire open source développé par Oracle Corporation.

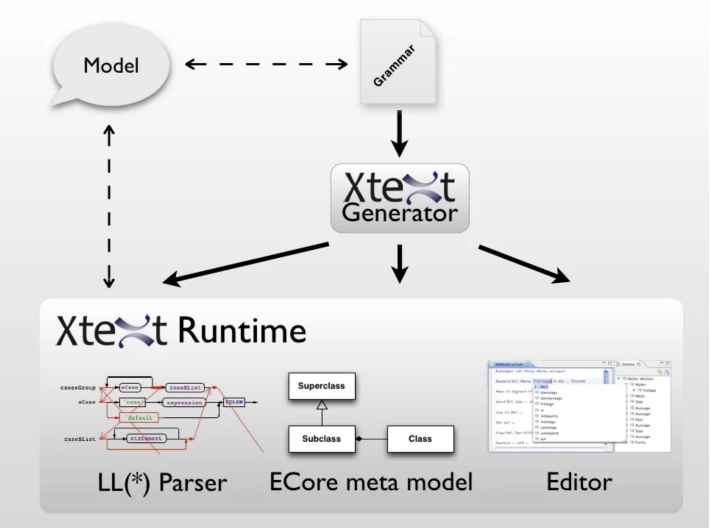

FIGURE 2.11 – Vue d'ensemble de l'outil Xtext.

Dans cette section, nous allons présenter la classifcation de la transformation des modèles proposée dans l'état de l'art (voir Figure 2.12).

Actuellement, il existe plus de 30 techniques de transformation de modèles dans la littérature. Parmi ces approches, Czarnecki et Helsen distinguent

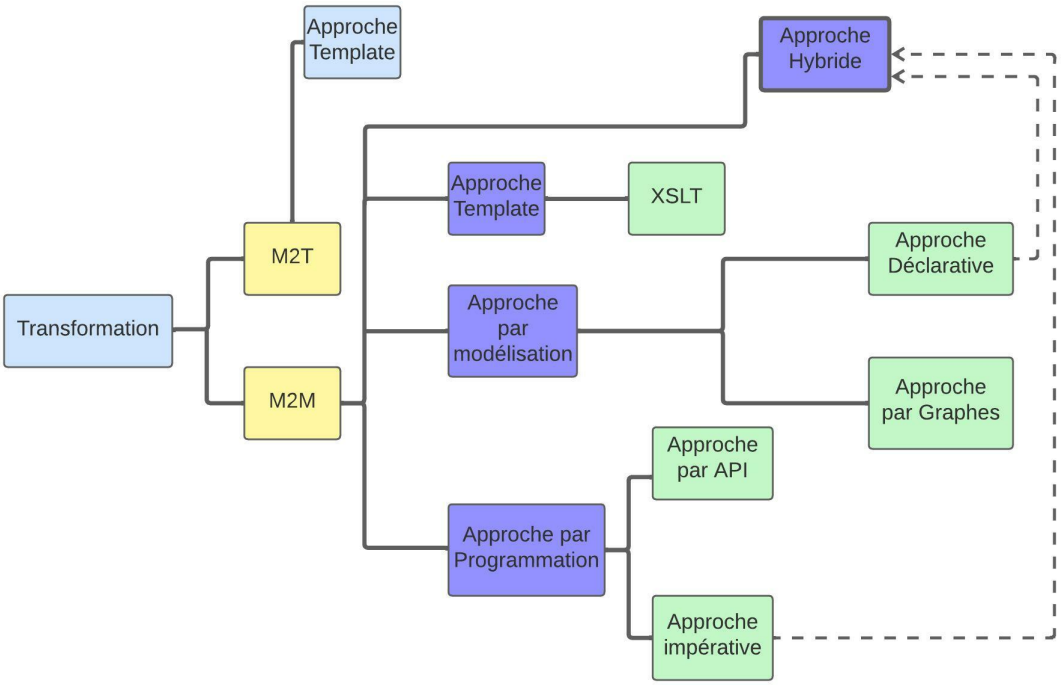

FIGURE 2.12 – Approches de transformation de modèle.

Visitor-based :

Un modèle de visiteur implémenté est dans un langage de programmation traverse le modèle dans un framework orienté objet[5] . Cela devient programme-ming plutôt que modelage.

Template-based :

Les modèles sont généralement exprimés dans la syntaxe concrète du modèle cible, pour donner des exemples avec des annotations de méta-code pour accéder au modèle **[sourcesyriani2018systematic]**. Cette approche est souvent utilisée par des générateurs de code (par exemple, Enterprise Architect)

Direct-Manipulation :

les modèles proposent une API [5]pour les exploiter et manipuler des modèles .(décrit dans le méta-méta-langage).Cela reste de la programmation tout en étant conscient qu'il s'agit de modèles avec une API dédiée.

Operational :

Ils consistent en des langages modélisés qui permettent de manipuler des modèles par exemple, déclaratifs et/ou impératif OCL [6] De plus, les méta-modèles sont complétés par une construction impérative offrant des méthodes/fonctions appellés dans les modèles eux-mêmes.

Graph transformation-based :

les modèles sont représentés sous forme graphiques, ainsi la théorie de la transformation de graphe est utilisée pour transformer les modèles. c'est une manière déclarative de décrire les opérations sur les modèles.

Relational :

Ils décrivent discrètement les correspondances entre la source et la cible.Souvent sous la forme de contraintes à résoudre.Ils sont implicitement multidirectionnels, mais la transformation sur place est plus difficile à réaliser.

Hybrid :

C'est une combinaison de deux ou plusieurs des approches précédentes.

5. Une API, ou interface de programmation d'application, est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications.
6. OCL : (Object Constraint Language) est un langage informatique d'expression des contraintes utilisé par UML.

2.2.8 La classifications des modèles de transformation

Ce qui différencie les différentes approches est l'élaboration des transformations de modèles et la façon dont sont spécifiées les règles des transformations. Plusieurs approches sont répertoriées et qui permettent la spécification de ces règles de correspondances.

- **Approche par programmation :** Consiste à utiliser les langages de programmation orientée objet[15]. L'idée est de programmer une transformation de modèles de la même manière que l'on programme n'importe quelle application informatique. Ces transformations sont donc des applications informatiques qui ont la particularité de manipuler des modèles. Ces applications informatiques utilisent les interfaces de manipulation de modèles. Cette approche est la plus utilisée car elle est très puissante et fortement outillée.
  - **Approche Impérative :** [26] Décrit ce que l'on fait mécaniquement, il n'y a aucune part de choix qui n'appartient pas au développeur. Par exemple une requête SQL n'est pas impérative, le SGBD fait comme il veut pour récupérer les données, de même pour le HTML, le navigateur internet est libre d'interpréter la page comme il veut .
  - **approches hybride :** Les approches hybrides sont une combinaison des différentes techniques. On peut notamment retrouver des approches utilisant à la fois des règles à logique déclarative et des règles à logique impérative.
  - **Approche par API :** Certaines APIs (Application Programming Interface) de transformation sont décrites dans des langages de programmations impératifs et sont ensuite fournies comme des bibliothèques permettant la description du processus de transformation suivant la syntaxe du langage utilisé. Les résultats sont alors d'une performance considérable. Cependant, le développeur a la charge de l'organisation et de la description de toutes les étapes de manière explicite en termes de déclarations impératives [43]
  - **Approche par Modélisation :** Consiste à appliquer les concepts de l'ingénierie des modèles aux transformations des modèles elles-mêmes. L'objectif est de modéliser les transformations de modèles et de rendre les modèles de transformation pérennes et productifs et d'exprimer leur indépendance vis-à-vis des plates-formes d'exécution .Cette approche est actuellement au stade de la recherche. Le standard MOF2.0 QVT' [7] en cours de définition à l'OMG a pour objectif de définir le méta modèle permettant l'élaboration des modèles de transformation de modèles. Actuellement, seuls quelques prototypes de recherche supportent cette approche.
  - **Approche par graphe :** Cette catégorie d'approches de transformation de modèles s'appuie sur la théorie de graphes. Une approche par transformation de graphes utilise usuellement des graphes typés, orientés et étiquetés où les règles de transformation de graphes possèdent un schéma LHS et un schéma RHS : le schéma LHS est repéré dans le modèle source et remplacé par le schéma RHS dans le modèle cible.
  - **Approche Déclarative :** [14] Dans cette approche, Une règle fait la correspondance entre un ensemble de concepts invoqué au niveau du modèle source et un ensemble de concept qui devrait être adopté au niveau du modèle cible. La mise en œuvre est réalisée par un moteur d'inférence. Cependant, l'un des principaux inconvénients de cette approche est la charge élevée de travail que doit effectuer le développeur qui doit spécifier toutes les contraintes d'appui à la transformation, tâche qui s'avère souvent fastidieuse

7. Query, View, Transformation

	Approches utilisée	Transformation (M2M)	Transformation(M2T)
AndroMDA	Template	+	+
ATL	Hybride	+	-
Accéléo	Template	-	+
Mola	Impératif	+	-

Tableau 2.1 – Approches de langages de la transformation des modèles.

	Horizontale	**Verticale**
Endogène	Refactroing	Formal refinement
Exogène	migration de langage	Génération de codes

Tableau 2.2 – Topologie de transformation.

— **Approche par template :** Consiste à définir les canevas des modèles cibles souhaités en y déclarant des paramètres [15]. Ces paramètres seront substitués par les informations contenues dans les modèles sources. On appelle ces canevas des « modèles cibles paramétrés » ou des « modèles Template». L'exécution d'une transformation consiste à prendre un modèle Template et à remplacer ses paramètres par les valeurs d'un modèle source. Cette approche nécessite un langage particulier permettant la définition des modèles Template. Sa puissance réside principalement dans le langage de définition des modèles Template. De tels langages sont en cours d'élaboration et n'ont pas encore la maturité des langages de programmation orientée objet utilisés dans la première approche . Le tableau suivantTableau 2.1 montre les approches de langages de la transformation des modèles.

2.2.8.0.1 Topologie de transformation

Une transformation de modèle génère un modèle cible à partir d'un modèle source[32]. ces deux modèles source et cible sont conformément aux même métamodèle ou à deux métamodèles différents aussi les modèles sources et cibles peuvent appartenir au même niveau d'abstraction (raffinement) ou à deux niveaux d'abstractions différents

Le Tableau 2.2 présente les topologies de la transformation.

2.2.8.0.2 Transformation endogène

Une transformation de modèles est dite endogène c'est lorsque les modèles source et cible sont conforme au même métamodèle. Donc les règles de transformation sont présentées sur un seul méta-modèle[36]. On utilise les transformations endogènes dans le cadre d'optimisation de modèle ou d'une simplification d'un modèle ou dans un refactoring.

2.2.8.0.3 Transformation exogène

Une transformation de modèles est dite exogène c'est lorsque le métamodèle du modèle source est différent du métamodèle du modèle cible [36] . Donc le formalisme qui va exprimer le modèle source est

différent du formalisme qui va exprimer le modèle cible. On utilise les transformations exogènes dans le cadre de migration de modèle d'une plateforme à une autre en restant au même niveau d'abstraction, ou pour la génération de code et aussi les opérations de rétro-ingénierie ou rétro-conception.

2.2.8.0.4 Verticalité et horizontalité

Une transformation de modèle peut aussi être classer 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" [38]

2.2.8.1 Exactitude de la transformation du modèle

L'extraction de modèle est une transformation de modèle exogène et verticale [34]. L'extraction est l'inverse de la génération de code et c'est l'une des activités essentielles dans l'ingénierie inverse et de la compréhension du programme . Elle permet de construire un modèle visuel à un niveau d'abstraction supérieur qui permettra de connaître la structure du code .

2.3 Conclusion

La transformation des modèles est l'élément coeur de l'ingénierie dirigée par les modèles (IDM). Dans ce chapitre, nous avons présenté les généralités sur la transformation des modèles. Ces notions de base nous permettront de nous familiariser avec ce domaine. La transformation des modèles est souvent exprimée avec des règles déclaratives qui contiennent des préconditions, des actions et des postconditions. Dans certaines situations la connaissance du domaine est immature, par conséquent il n'existe pas une règle prédéfinie. La transformation des modèles doit reposer sur une séries d'expériences, ce qu'on appel la transformation des modèles à base d'exemple (MTBE) qui sera l'objet du chapitre suivant.

.

Chapitre

3

Transformation de modèle par les exemples

« Demande conseil à celui qui en a de l'expérience et non au médecin. »

— Proverbe Marocain

Sommaire

3.1 Introduction

L'objectif de ce chapitre est d'explorer les travaux qui traitent l'apprentissage de transformations dans le cadre de la transformation des modèles par l'exemple (TMPE, en anglais, MTBE : Model Transformation By Example). Ce chapitre est consacré à l'apprentissage des transformations de modèles. Nous allons passer en revue les différentes contributions qui s'inscrivent dans le cadre de TMPE (*revue de littérature non exhaustive*), et Enfin, la dernière partie de ce chapitre prend la forme d'une synthèse qui résume les caractéristiques clés et les limites des contributions identifiées.

3.2 Concepts de base de TMPE

Dans cette section, nous allons présenter les concepts de base de la transformation des modèles a base d'exemples.

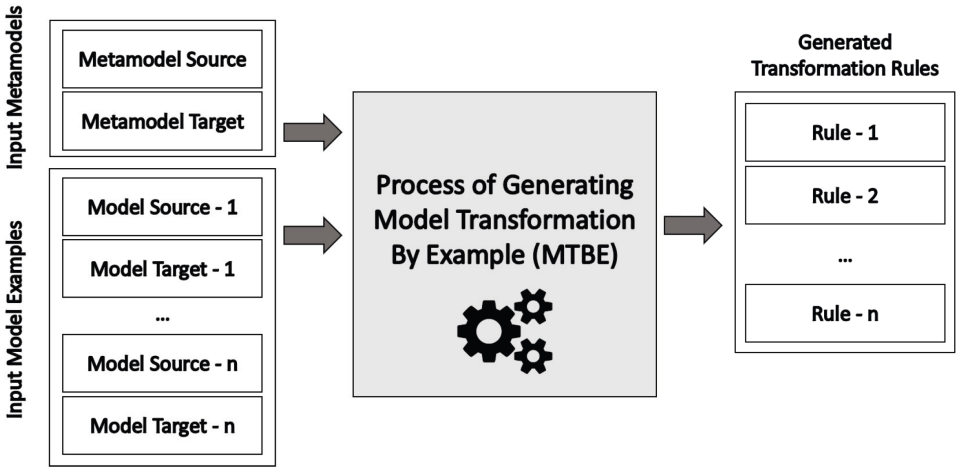

FIGURE 3.1 – Principe de transformation de modèle par l'exemple [20].

Les travaux de TMPE oeuvrent à dériver automatiquement un programme de transformation à partir d'un ensemble d'exemples fournis en entrée. Chaque exemple est une paire de modèles constituée d'un modèle source et d'un modèle cible. En plus des exemples, la majorité des approches de TMPE exploitent des traces fines de transformation. Ces traces sont des liens, de plusieurs-à-plusieurs, qui associent un groupe de n éléments sources à un groupe de m éléments cibles.

3.2.1 Pourquoi l'exemple ?

Les exemples jouent un rôle clé dans le processus d'apprentissage humain. Il existe de nombreuses théories sur les styles d'apprentissage dont certaines exemples doivent être considérés comme des artefacts majeurs.

Pour une description des théories de style d'apprentissage populaires d'aujourd'hui, voir par exemple. Toutes ces théories, en particulier celles référencées, sont assez similaires en ce qui concerne leur notion

d'une dimension d'apprentissage utile à la déduction de règles générales, comme illustré dans la Figure 3.2. Aussi, les exemples sont des outils courants dans l'enseignement de l'informatique. Surtout dans les domaines fortement influencés par les algorithmes et les mathématiques.

La Figure 3.2 montre une sorte de transformation basée sur une serie des exemples. Qui consiste en à un exemple de paires avec trois traces identifiées (toutes les traces ne sont pas affichées). Pour chaque trace, le fragment cible (en bas) a été déterminé comme correspondant au fragment source (en haut). Les cartographies de transformation peuvent être identifiées par un expert au cours du processus de conception ou récupérées (semi-) automatiquement à l'aide d'un traitement informatique basé sur l'exemple.

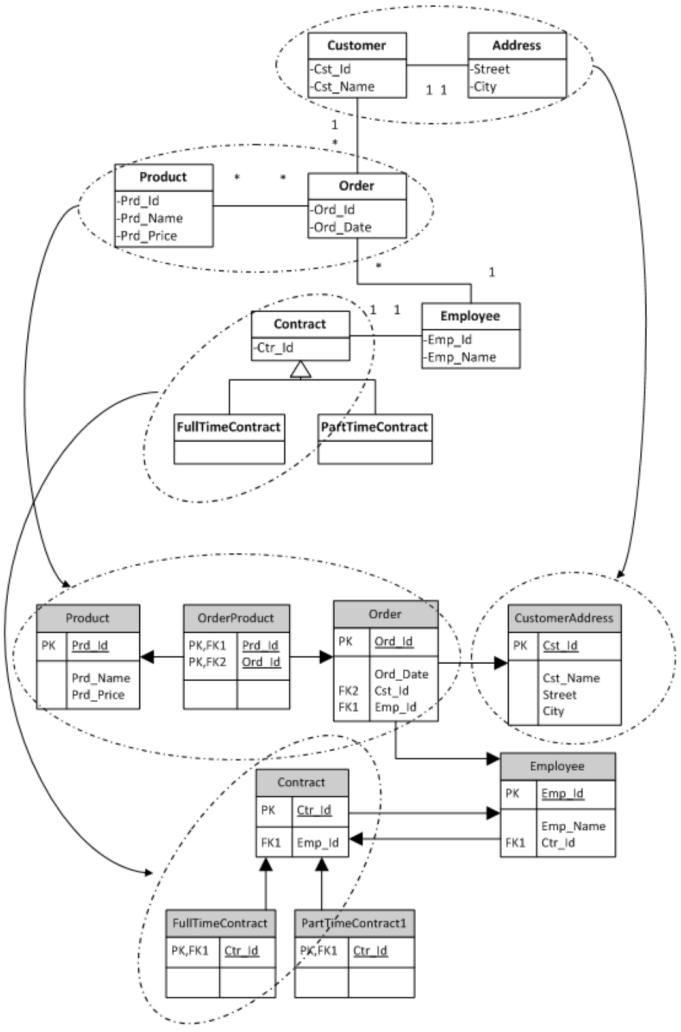

FIGURE 3.2 – Exemple d'un exemple de paire source-cible avec trois traces identifiées

3.3 Derivation de la règle de transformation

Dans cette section, nous allons présenter une classification des approches transformations adoptées par la communauté de l'IDM. Nous distingons deux grandes famille (i) Approches basées sur les règleset (ii) Approches basées sur les exemple.

1. **Approches basées sur les règles :** Les Transformation à base de règles sont souvent décrites comme des règles en premier lieu. Ce type de transformation est un langage de programmation déclaratif à base de règles basé sur le mode impératif ou décratatif. Voici un exemple de transformation inplace (UML 2 UML). La Figure 3.3 represente une classe UML nommée `Person` qui doit être transformer en classe nommée `Contact`. Le modèle d'entré qui contient la classe `Person` est stocker dans un fichier XMI (`Ma.xmi`). La transformation exprimé en langage ATL genere un fichier de sortié nommé `Mb.xmi` qui contient la description de classe `Contact`.

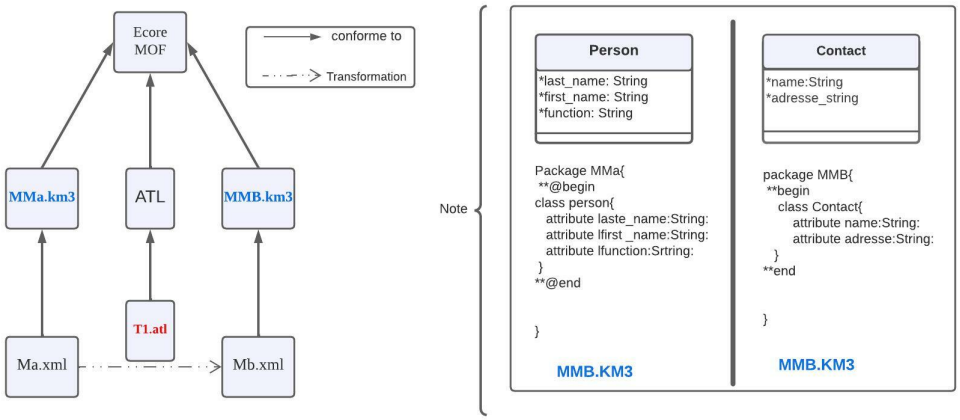

FIGURE 3.3 – Illustration de la transformation

Listing Liste 3.1 montre le programme de la transformation qui est stocké aussi dans un fichier ATL `T1.alt`.

```
1  T1.alt
2  ---------
3  module Person2Contact
4  create OUT : MMb from IN : MMa;
5
6  rule Start {
7  from
8  p : MMa!Person (
9  p.function = 'Boss'
10 )
11 to
12 c : MMb!Contact (
13 name ←p.first_name + p.last_name
14 )
15 }
```

Listing 3.1 – Exemple d'une règle de transformation inplace (UML2UML) exprimée en ATL

2. **Approches basées sur les exemples :** Cette catégorie regroupe deux types de méthodes qui se trouve dans la littérature : (i) les approches basées sur des algorithmes de recherche (par ex.

algorithme génétique) et les algorithmes de l'apprentissage automatique (par ex. *arbre de décision*).

— **Approches basées sur la recherche :** ces approches considèrent le problème de transformation des modèles comme un problème d'optimisation sous contraintes. La strategie de recherche utilisée est basées sur une fonction objective qui doivent etre definie par le concepteur de modèle de transformation, prenant on considération des contraintes de l'application et les besoins des utilisateurs. Nous pouvons citer deux algorithmes utilisées : l'algorithme génétique et les algorithmes d'optimisation à base de PSO (Particle Swarm Optimization). La Figure 3.4 illustre le principe des algorithmes de recherche dans notre contexte MTBE.

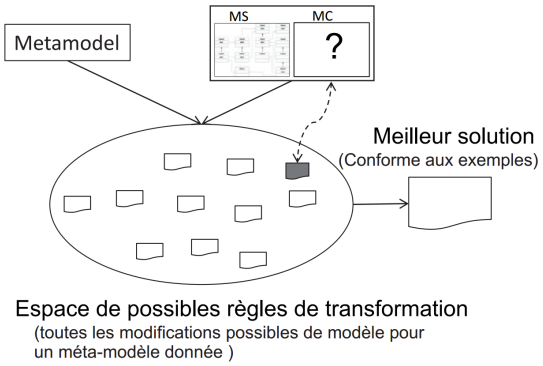

FIGURE 3.4 – Illustration de Principe des algorithmes de recherche dans notre contexte MTBE.

Souvent les auteurs adoptent des algorithmes d'optimisation de recherche basées les heuristiques. Une heuristique est une méthode qui cherche une bonne solution en un temps de réponse raisonnable sans garantir l'optimalité. Notons qu'une heuristique est une méthode, conçue pour un problème d'optimisation donné, qui produit une solution non nécessairement optimale lorsqu'on lui fournit une instance de ce problème. Nous pouvons classer les méta-heuristiques selon le nombre de solutions traitées simultanément en deux grandes catégories (voir) : (1) méthodes basées sur une population et (2) méthodes basées sur une seule solution (voir Figure 3.5).

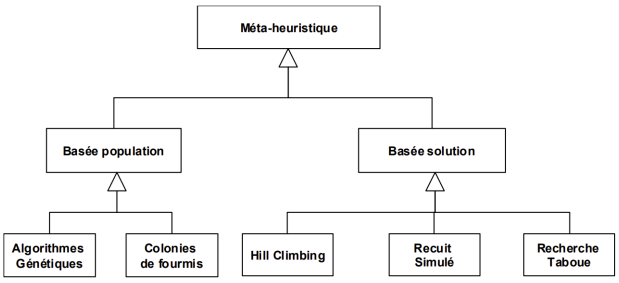

FIGURE 3.5 – Classification des principales méta-heuristiques.

— **Approches basées sur l'apprentissage :** dans cette catégorie on distngues deux types de techniques : implicite et explicite. Les approches explicite permet de générer les règles de transformation avec une description détaillé qui conduit à la génération de modèle cible. Les algorithmes les plus utilisées arbre de decision, la programmation en nombre de entier

etc.. Pour les techniques implicites, nous pouvons citer les algorithmes de deep learning ou apprentissage profond, Natural language processing (NLP).

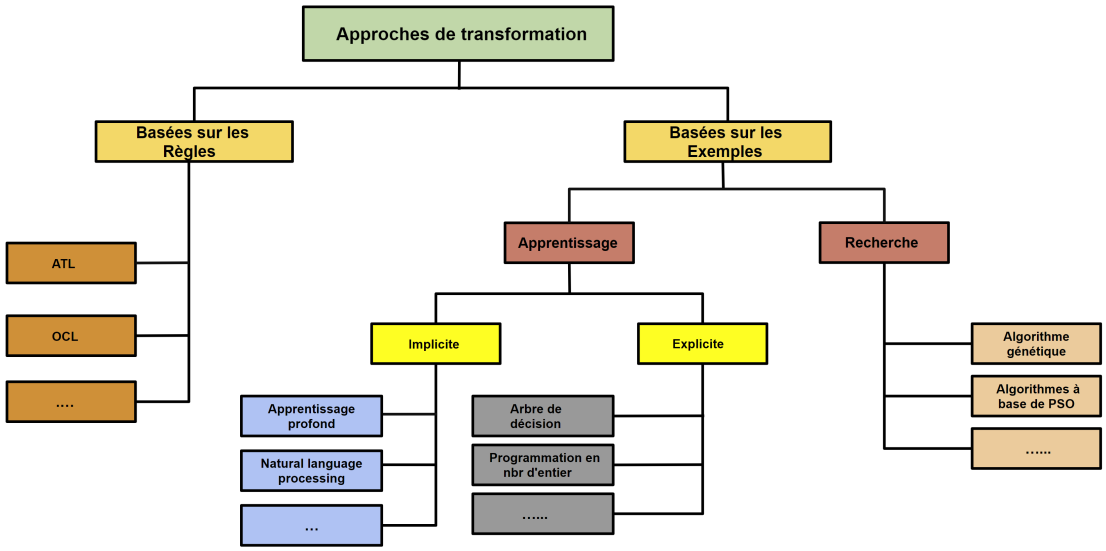

FIGURE 3.6 – Classification des approches de transformation des modèles.

Dans la section suivante, nous allons présenter quelques travaux de la littérature qui sont basés sur le MTBE.

3.4 Les approches de Transformation de modèle a base d'exemples

Dans la literature, nous avons recencé un certain nombre de travaux qui traitent la transformation des modèle à base l'exemple.

D'aprés ces travaux, nous avons essayé de décliner les dimensions illustrées dans la Figure 3.7. Ces dimensions montrent les différentes alternatives considérées dans chaque contribution, notez que certaines dimensions sont bien expliquées et certains sont noyés dans le texte de discussion des articles.

On considère les septs dimensions suivantes :

- **Approche utilisée :** Cette dimension détermine la catégorie de l'approche utilisée : (i) basée recherche ou bien (ii) basée apprentissage machine.
- **Algorithme utilisé :** Cette dimension détermine le type d'algorithme utilisé en fonction de sa catégorie.
- **Entrées considérées :** Cette dimension explicite les entrées considérées pour traiter le problème TMBE.
- **Sorties :** Cette dimension montre le résultat retournée par l'approche adoptée : exemple, exemple + les règles..
- **Explicabilité :** Cette dimension détermine le degré de l'applicabilité de l'algorithme, c-à-d est ce que l'algorithme montre le raisonnement est les règles conduisent à la déduction de la règle..
- **Implication des utilisateurs :** Cette dimension détermine si l'utilisateur final est impliqué dans le processus de la transformation, par exemple le cas de transformation semi automatique(

MTBD).

— **Type de règle :** cette dimension explicite le type de règle de mapping : Il existe 4 types de relation : OneToOne, OneToMany, ManyToOne, ManyToMany.

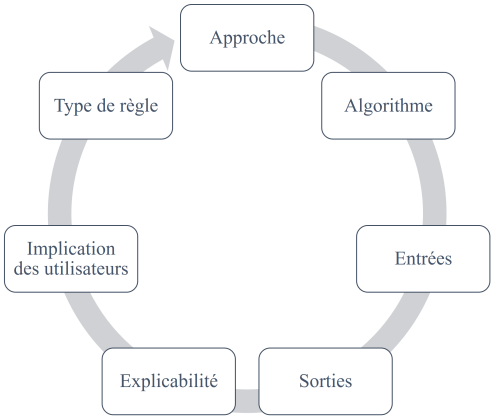

FIGURE 3.7 – Dimensions considérées dans l'analyse des travaux TMPE.

3.4.1 Travaux de Varro

En 2006, Varró [44] propose une première approche pour l'apprentissage de transformations à partir de paires d'exemples en utilisant un algorithme ad-hoc basé sur les graphes. L'algorithme prend en entrée des paires de modèles interreliés (accompagnés de traces) et dérive un ensemble de règles de transformation de type 1 1. Le processus de dérivation est itératif et interactif, à chaque itération, les règles produites sont ranées par l'utilisateur. La contribution est ensuite automatisée davantage par Balogh et al.[33] en utilisant la programmation logique inductive (PLI). Cette nouvelle approche, bien que toujours itérative et incrémentale, permet de dériver des règles de transformation plus complexes (de type n, m).

3.4.2 Travaux de Wimmer

Durant la même période Wimmer et al. [21, 1] propose une approche similaire pour dériver des transformations sous la forme de règles de type 1 1 exprimées en ATL. La contribution en question est également itérative et incrémentale, elle dière cependant des deux contributions précédentes sur deux aspects. D'abord, les traces de transformation exploitées sont spéciées dans une syntaxe concrète plutôt-qu'abstraite. Ensuite, le processus de dérivation se fait selon une approche orientée objet, contrairement à celui de Varró où des graphes sont utilisés. La contribution de Wimmer est également améliorée dans des contributions subséquentes, par Strommer et al.[27], où un langage pour la dénition de correspondances de type (n , m) est présenté, jumelé à un algorithme de raisonnement pour la dérivation de règles (nm également) à partir des correspondances exprimées. L'usage d'opérateurs de chaînes tel que la concaténation est également brièvement mentionné dans l'une des deux contributions.

La Figure 3.9 montre un méta-modèle simplifiés de UML et Entité-Relation utilisé dans Wimmer .

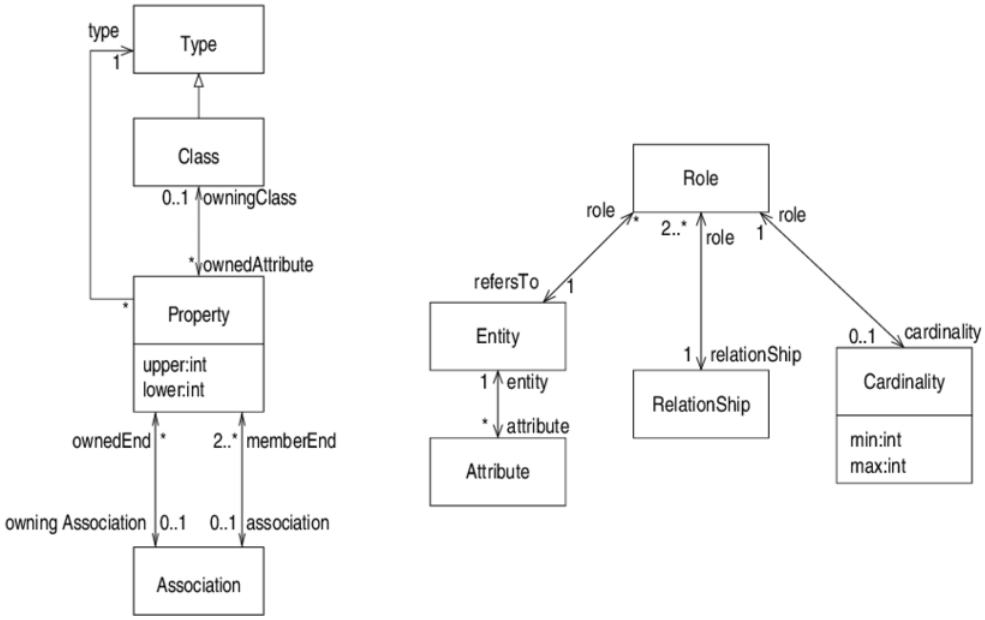

FIGURE 3.8 – Méta-modèles simplifiés de UML (à gauche) et Entité-Relation (à droite) utilisés dans wimmer et al 2007[17].

3.4.3 Travaux de Garcia-Magarino

Un autre travail qui s'inscrit dans le contexte des transformations de modèles par l'exemple est celui de Garcia-Magarino et al.[30] qui propose un algorithme permettant de générer des règles de transformation de type (n ,m), vraisemblablement dans plusieurs langages de transformation, à partir d'un ensemble de paires de modèles sources et cibles interconnectés (agrémentés par des traces). À l'instar de [44], l'approche utilise des graphes. Afin de pallier le fait que certains langages de transformations ne permettent pas d'écrire des règles de plusieurs à plusieurs, les règles sont d'abord dérivées dans un langage de transformation générique avant d'être transformées dans le langage de transformation souhaité (ATL dans l'article).

3.4.4 Travaux de Kessentini

Kessentini et al.[44] utilise des analogies pour effectuer une transformation. Contrairement aux contributions citées plus haut, cette approche ne produit pas de règles de transformation, mais dérive plutôt le modèle cible directement à partir du modèle source en considérant la transformation de modèles comme un problème d'optimisation. Le problème tel que posé est abordé en utilisant l'optimisation par essaims particulaires (OEP) dans la première contribution et une combinaison de OEP et du recuit simulé (RS) dans la deuxième. L'approche d'apprentissage est ensuite améliorée dans [23] où des règles de transformation sont produites à partir de modèles sources et cibles qui ne sont pas accompagnés de traces de transformation. L'approche est validée sur une transformation de modèles comportementaux (diagramme de séquence vers réseau de Petri colorés).

3.4.5 Travaux de Dolques

Une autre contribution de TMPE qui ne produisait pas de règles de transformation initialement est celle de Dolques et al.[16]. Ce travail se base sur l'analyse relationnelle de concepts (ARC), une variante de l'analyse formelle de concepts, pour classier les éléments sources et cibles ainsi que les correspondances fournies en entrée. Les patrons identiés sont organisés dans des treillis partiellement ordonnés et sont ensuite analysés pour ltrer les plus pertinents. L'approche est également étendue par Saada et al.[40] où des règles exécutables sont produites à partir des patrons ltrés. Dans cette dernière contribution es règles sont exprimées en Jess. [8] Il est possible ensuite de raisonner sur ces connaissances à l'aide de règles déclaratives. Jess applique les règles sur la base de faits en utilisant un ltrage par motifs basé sur l'algorithme Rete [39]. Il est possible d'associer la notion de fait en Jess au concept d'objet dans le paradigme orienté objet (POO). Chaque fait peut contenir plusieurs attributs, appelés « slots ». Jess ore également la possibilité de dénir des gabarits de faits (template) qui correspondent à la notion de classe dans le POO. An d'utiliser Jess pour la transformation de modèles,nous exprimons chaque modèle sous la forme d'un ensemble de faits. Chaque élément du modèle est un fait dont les slots sont des attributs ou des références vers d'autres éléments. Les méta-modèles sont exprimés, quant à eux, sous la forme de gabarits de faits. La Liste 3.1 présente un exemple trivial, où sont représentés, un méta-modèle et un modèle comportant un seul élément.

3.4.6 Travaux de Baki

[3] cet approche prend en entrée un ensemble de pairs d'exemples de modèles sources et cibles accompagnés de traces de transformations capturant la correspondance entre les diférents fragments sources et cibles. Le processus d'apprentissage proposé dans cette dernière permet alors de dériver un programme de transformation en trois étapes principales. Dans un premier temps, les traces fournies sont complétées et analysées afin de construire des pools d'exemples cohérents. Ensuite, un ensemble de règles est dérivé pour chaque pool en utilisant un programme génétique. Durant cette étape, les règles sont également traitées pour éliminer d'éventuelles erreurs ou incohérences. Enfin, la troisième et dernière étape consiste à fusionner les groupes de règles apprises en un programme de transformation qui est affiné en utilisant la méthode du recuit simulé. Le processus d'apprentissage de cette approche est validé sur sept (7) cas de transformations pré- sentant divers caractéristiques et degrés de complexité. Les résultats obtenus indiquent que les transformations les plus communes sont parfaitement apprises. Par ailleurs, l'apprentissage des cas de transformations les plus complexes permet de générer des modèles cibles très similaires à ceux attendus.

3.4.7 Travaux de Faunes

Les travaux les plus récents dans le contexte de la TMPE sont ceux de Faunes et al.[19, 4] dans les quels la programmation génétique (PG) est utilisée pour faire évoluer une population de transformations sur plusieurs générations jusqu'à produire la transformation attendue. Le processus de dérivation prend en entrée des paires d'exemples de modèles sources et cibles uniquement (sans traces de transformation)

8. Jess :C'est un environnement de transformation de modèles.,Jess est un moteur de règles qui permet de stocker en mémoire des connaissances exprimées sous la forme de faits (facts).

Approche	Algorithme	Entrée	Sortie
Varró	Ad-hoc	Exemples et Traces	Règles
Wimmer	Ad-hoc	Exemples et Traces	Règles
Balogh	PLI	Exemples et Traces	Règles
Kassentini	OEP/OEP-RS	Exemples	MC
Deloques	ARC	Exemples et Traces	Règles
Faunes	PG	Exemples	Règles
Baki	PG	Exemples	Règles

Tableau 3.1 – Caractéristiques des approches TMPE actuelles.

et produit en sortie des règles exécutables de type (n, m). L'approche est ensuite améliorée par la contribution de Baki et al.[3] où le programme génétique tente d'apprendre simultanément les règles de transformation ainsi que le contrôle d'exécution qui doit être exercé sur cellesci pour former un programme de transformation correct. Cette seconde version permet également de dériver des règles de transformations plus complexes en incluant la négation de conditions, l'usage de primitives de navigation ainsi que la considération des types et des domaines de dénition lors de la construction du modèle cible.

Le tableau suivant Tableau 3.1 illustre les caractéristiques des approches MTPE actuelles.

3.5 Carte conceptuelle des approches MTBE (Approches Timeline)

Les travaux de TMPE ont dérivé automatiquement un programme de transformation à partir d'un ensemble d'exemples fournis en entrée. Chaque exemple est une paire de modèles constituée d'un modèle cible (MS) et d'un modèle source (MC). En plus des exemples, la majorité des approches de TMPE exploitent des traces de transformation. Ces traces sont des liens, de plusieurs-à-plusieurs, qui associent un groupe de n éléments sources à un groupe de m éléments cibles.

Nous avons essayé de tracer l'évolution de ce domaine sous forme d'un carte pour montrer les travaux de bases qui constituant l'état de l'art. La Figure 3.9 montre la carte conceptuelle des approches MTBE (Approches Timeline).

3.6 Synthèse

La Figure 3.10 montre une comparaison des travaux. Les approches MTBE existantes ne résolvent que partiellement le problème de dérivation des règles de transformation (voir Figure 3.10). La plupart d'entre eux nécessitent des mappings détaillés (traces de transformations) entre les modèles source et cible, Les exemples sont difficiles à fournir dans certaines situations. D'autres peuvent difficilement dériver des règles de transformation qui testent de nombreuses constructions dans le modèle source et/ou produisent de nombreuses constructions dans le modèle cible. Cependant, les règles de transformation "n-m", avec leur complexité, sont une nécessité dans les problèmes de transformation complexes. Enfin, certaines approches produisent des règles de transformation non exécutables et abstraites qui doivent être complétées et traduites manuellement dans un langage exécutable.

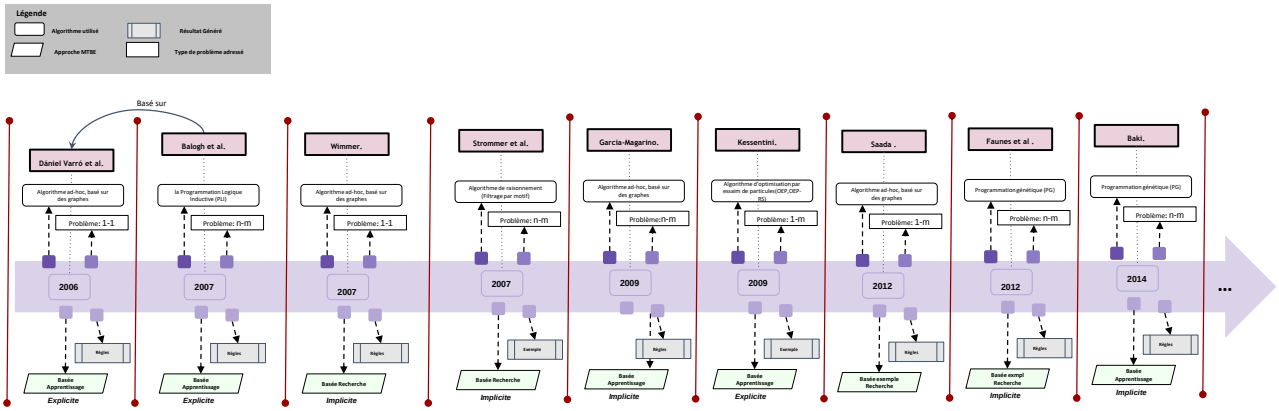

FIGURE 3.9 – Carte conceptuelle des approches MTBE (Approches Timeline)

.

Dans certain cas, au lieu de déduire des règles MTBE, Model Transformation By Demonstration (MTBD) on demande aux utilisateurs de démontrer comment une transformation de modèle doit se comporter en modifiant directement (ajout, suppression, mise à jour, etc.) un modèle d'entrée donné pour simuler le processus de transformation de modèle étape par étape. marcher. Un moteur d'inférence et d'enregistrement capture toutes les opérations de l'utilisateur dans une tâche de transformation pour déduire l'intention d'un utilisateur.

	Input				
	Exemple	Exemple & Traces	Semi-automatique	Règle	
Approaches	Fauness *et al.*. [2012; 2013] Al-Jamimi and Ahmed [2017]	Strommer *et al.*. [2007; 2008] Balogh *et al.*. [2009] García-Magariño *et al.*.[2009] Baki *et al.*. [2014; 2016]	Langer et al. [2010] Kehrer *et al.*. [2017]	n-m	Output
	Kassentini *et al.*. [2010]	Saada *et al.*. [2012a; 2012b]		1-m	
		Varró *et al.*. [2006] Wimmer *et al.*. [2007]		1-1	
		Kassentini *et al.*. [2008; 2012] Deloques *et al.*. [2010]	Brosch *et al.*. [2009a; 2009b] Sun *et al.*. [2009; 2011]	-	
	MTBE		MTBD		

FIGURE 3.10 – Comparaison des approches de transformation de modèle par l'exemple

.

Nous avons synthétiser les travaux qui traitent le problème de transformation de modèles par l'exemples. Ces travaux sont basés sur des approches heuristiques comme par exemple les algorithmes génétiques [3, 4] ,des approches basées sur les graphes et des approches basées sur l'apprentissage automatique . Ces travaux traitent la problématique liée à la transformation de modèles comme l'absence des règles.

Les approches MTBE existantes ne résolvent que partiellement le problème de dérivation des règles

de transformation. La plupart d'entre eux nécessitent des mappings détaillés (traces de transformations) entre les modèles source et cible des exemples, difficiles à fournir dans certaines situations. D'autres peuvent difficilement dériver des règles de transformation qui testent de nombreuses constructions dans le modèle source et/ou produisent de nombreuses constructions dans le modèle cible. Cependant, dans certains domaines comme *la robotique* et *l'automobile*, la connaissance est immature, cette nature de connaissance rend la transformation du modèle plus complexe et on peut trouver plusieurs points de vue. De plus, le manque de données au départ peut être un autre défi qui doit être relevé. Dans ce mémoire, nous abordons ce problème pour générer les règles de transformation en commençant sans données, avec des règles qui vont être affinées lors de l'utilisation du système en cours de conception ou de simulation.

Cette situation nous a motivé à voir la possibilité de proposer une approche de transformation inspirée de l'apprentissage par renforcement (*Reinforcement Learning*) qui raffine au fur et à mesure les règles de transformation.

3.7 Conclusion

Dans ce chapitre, nous avons présenter les approches de MTBE qui représente un bref état de l'art des différentes techniques et algorithmes utilisés pour permettre de déduire si un modèle source peut être transformer par un modèle cible dans le cas de l'absence de la règle. Le chapitre suivant présente notre approche de transformation de modèle dans la cadre MTBE.

Troisième partie

Contribution

Chapitre

4

Approche MTBE proposée

« Chaque individu apporte au monde sa contribution unique. »
— Jack Kornfield

Sommaire

4.1 Introduction

La transformation de modèle par l'exemple (TMPE) semble être une solution appropriée du problème de transformation des modèles. De manière analogue à la programmation par l'exemple et l'interrogation par l'exemple , les approches de MTBE ont pour but d'apprendre automatiquement un programme de transformation à partir d'artéfacts fournis en guise d'exemples. La machine peut utliser une série d'exemples avec une trace détaillée, une signature et des composantes du modèle. Dans ce chapitre nous allons formaliser notre problème de recherche en spécifiant les entrées et les sorties, et nous allons proposer des algorithmes d'apprentissage automatique (*machine Learning*) et de trouver le modèle qui transforme une nouvelle instance de modèle source (SM) vers le model cible (TM) correspond.

4.2 Exemple de modèle dynamique (comportementale)

Cette section présente un DSML simple, à savoir réseaux de Petri, machines à états finis et PacMan, suivant cette présentation :

- La structure du DSL est spécifiée à l'aide d'un métamodèle exprimé en MOF ; nous spécifions également de manière informelle une syntaxe concrète possible, puis fournissons un modèle témoin simple.
- La sémantique du DSL est esquissée, sans détails des implémentations qui dépendent fortement du MTL, et du style de modélisation du concepteur MT. Nous montrons cependant pour l'exemple PacMan comment une approche similaire dans divers MTL peut conduire à une structure MT similaire, indépendamment du MTL sous-jacent.
- Un ensemble de transformation sont ensuite décrites en détail.

Notez que nous supposons que la structure DSL suit le modèle DSML exécutable (ou XDSL), qui se compose de deux métamodèles distincts pour définition des DSML exécutables :

- Le métamodèle de définition de domaine capture la structure statique du DSL, qui sert généralement à définir des modèles valides à l'aide de diverses syntaxes textuelles et/ou visuelles ;
- Le métamodèle de définition d'état ajoute de nouvelles informations au-dessus du métamodèle de définition de domaine pour permettre une exécution basée sur l'état, définissant ainsi les éléments qui sont modifiés pendant l'exécution.

Les deux métamodèles doivent généralement être fusionnés, en utilisant différentes techniques (par exemple, en utilisant l'approche de "fusion de packages" de MOF, ou d'autres technique de composition. À des fins de clarté et de simplification de la présentation, nous décrivons les deux métamodèles de manière fusionnée, bien que nous distinguions visuellement chaque partie : les éléments du métamodèle de définition de domaine sont représentés à l'aide de la police normale avec des flèches simples pour références ; tandis que les éléments de définition d'état utilisent des polices courbes et des références en pointillés.

4.2.1 FSM : Un DSL pour la FSM

Les machines à états finis (FSM :Finite State Machines) représentent un DSL commun pour capturer le comportement basé sur l'état de divers domaines biologiques, mais aussi informatiques (par exemple, les grammaires régulières de Chomsky, entre autres). Nous considérons FSMs qui sont simplifiés de diverses manières : en particulier, nous ciblons un sémantique mot-acceptation (*word-acceptance*), où les transitions ne contiennent pas de gardes et les déclencheurs sont réduits à leur expression la plus simple, à savoir des chaînes simples La figure 4.1 spécifie le métamodèle de la Finite State Machine DSL. Un FSM est composé de States identifiés par un name, et de Transitions qui contiennent un simple trigger.

La figure 4.2 décrit un modèle simple composé de trois States et trois Transitions, pour reconnaître l'expression régulière $(\mathtt{a} \cdot \mathtt{b})^\star\ \mathtt{b}$.

Nous supposons la représentation visuelle classique pour FSM : un State est représenté par un cercle étiqueté avec son name ; et une Transition est représentée par une flèche pointant vers sa cible et étiquetée par son déclencheur. Nous représentons le courant State en superposant une forme arrondie rouge (que nous appelons *token*) sur le State correspondant (cf. figure 4.2).

La sémantique acceptant les mots est encodée dans une transformation appelée accept qui lit un Word et traverse le FSM. Un Word est accepté si l'State atteint lorsque le Word devient vide est FINAL.

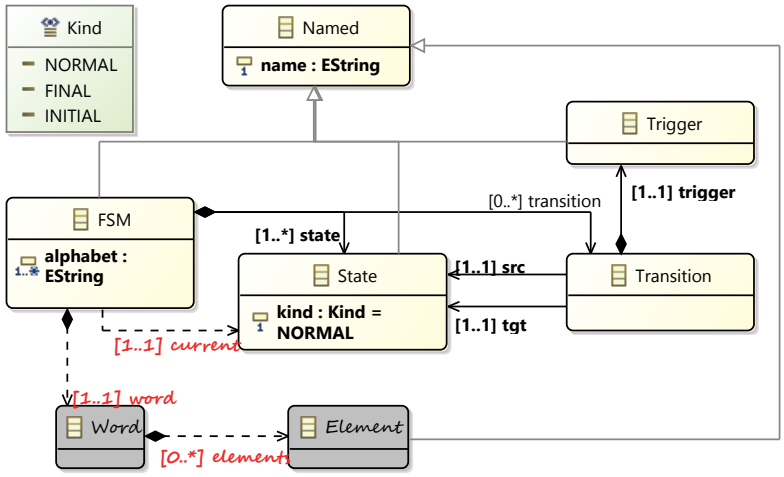

FIGURE 4.1 – Un métamodèle pour FSM.

4.2.2 PN : Un DSL pour le PN

Le formalisme des réseaux de Petri (PN : Petri Net) est populaire pour la modélisation de divers systèmes physiques et informatiques qui incluent la concurrence, car il permet la vérification formelle des propriétés intéressantes (généralement, l'accessibilité, la sécurité et la vivacité). Nous considérons ici des PN Place/Transition simples avec des arcs pondérés avec leur sémantique de tir (fire) traditionnelle.

La figure 4.3 spécifie le métamodèle d'un PN DSL. Un PN est un graphe bipartie dont les noeuds sont des Places et les Transitions, sont des arêtes appelé *pondéré* (*weighted*) Arcs. Le marquage (marking)

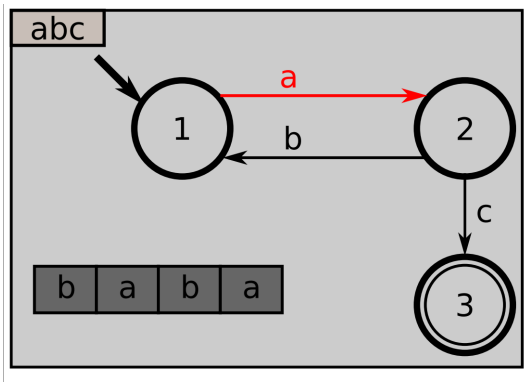

FIGURE 4.2 – La syntaxe concrète pour FSM.

représente les jetons hébergés par un Lieu ; il changera pendant l'exécution lors du tir de Transition.

Nous supposons la syntaxe concrète visuelle traditionnelle des PN : un Lieu est représenté par un cercle, un Transition par une boîte noire fine et un Arc par une flèche dirigée portant une étiquette représentant sa poids (omis si égal à 1).

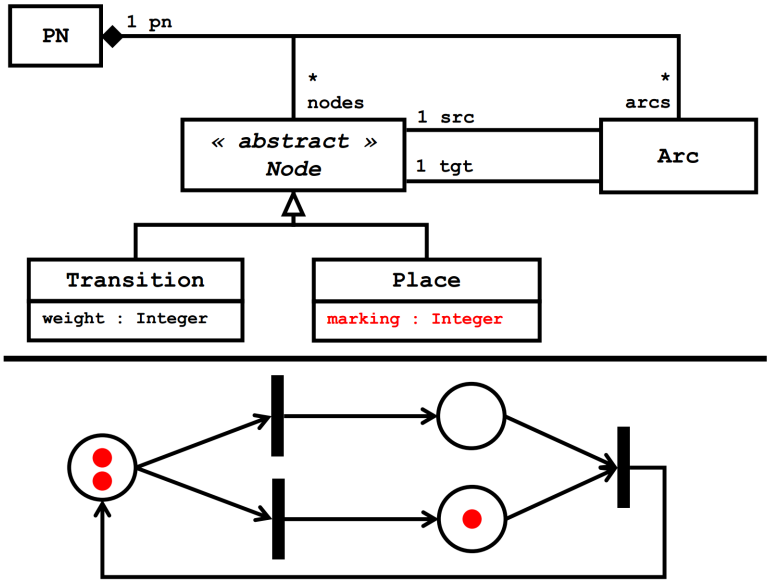

FIGURE 4.3 – Un métamodèle (en haut) et un modèle (en bas) pour les réseaux de Petri.

En partant d'un marquage initial, c'est-à-dire en fixant les valeurs de marquage pour chaque Lieu apparaissant dans un modèle, l'exécution d'un PN consiste à activer séquentiellement les *tirs* Transitions , jusqu'à ce qu'aucune Transition ne soit plus activée. Une Transition t est *enabled* ssi chaque entrée Place (c'est-à-dire une Place connectée à t par un Arc dirigé vers textsft) est marqué avec au moins le poids de l'Arc. Le déclenchement de t consomme des jetons de tous les Places d'entrée et crée des jetons sur les Places de sortie, selon le weight des Arcs de sortie correspondants.

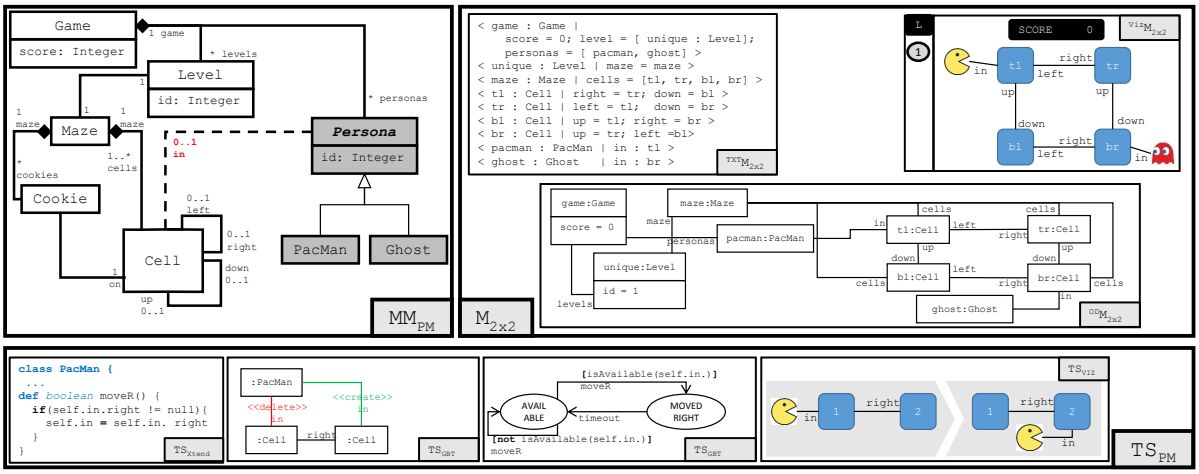

FIGURE 4.4 – Spécification d'un DSL pour le jeu PacMan.

L'ingénieur DSL crée le métamodèle MM_{PM} (suivant le modèle DSML exécutable ; un concepteur MT spécifie une transformation TS_{PM} (composé de plusieurs unités de transformation telles que *moveR* représenté ici).

4.2.3 PacMan : Un DSL pour le jeu PacMan

Le jeu PacMan est un jeu d'arcade populaire qui a suscité l'intérêt de la communauté MDE car il capture un DSL réactif simple et bien connu avec une syntaxe concrète facilement compréhensible et présente des fonctionnalités intéressantes pour l'exécution en temps réel.

4.2.3.1 Spécification

Sur le compartiment en haut à gauche de La est représenté un métamodèle MM_{PM}, tel que créé par un ingénieur DSL. Un Game consiste en une séquence de Levels, chacun affichant un Maze où les Personas évoluent. Un Level se termine lorsque PacMan est mangé par un Ghost, ou lorsqu'il a mangé tous les Cookies. Plusieurs joueurs peuvent s'affronter pour le Score le plus élevé.

Dans le compartiment supérieur droit de La , un modèle de base M_{2x2} avec un unique Niveau de taille 2x2, éventuellement créé par un modélisateur, est représenté à l'aide de trois syntaxes concrètes différentes : textuel pour ${}^{TXT}M_{2x2}$ (inspiré par eMotions) ; basé sur le diagramme d'objets UML pour pour ${}^{OD}M_{2x2}$; et librement inspiré du vrai jeu pour ${}^{Viz}M_{2x2}$, les deux derniers étant graphiques.

4.2.3.2 Exécution

PAcMan est un DSL *réactif*, c'est-à-dire qu'il réagit aux stimuli externes donnés par un joueur qui décide de la direction à prendre. Le compartiment inférieur de La Section 4.2.3.1 décrit une règle simple *moveR* (déplacer Pac-Man sur un Cell à sa droite, si disponible) dans le cadre de la spécification MT de simulation TS_{PM}, dans le cadre de MM_{PM}'s sémantique exécutable. Le premier MT, TS_{Xtend}, utilise la métaprogrammation (basée sur Xtend avec GeMoC). Les suivants et derniers sont basés sur les transformations de graphe : TS_{GBT} repose sur ${}^{OD}M_{2x2}$ pour exprimer la réécriture (comme serait exprimé

par exemple. dans Henshin) ; tandis que TS_{VIZ} repose sur ${}^{Viz}M_{2x2}$ (comme on l'exprimerait par ex. dans AtoMPM). Pour terminer, TS_{FSM} présente un fragment MT exprimé avec une machine à états finis d'UML. Notez que toutes les spécifications MT (fragments) sauf TS_{Xtend} présentent une représentation graphique, montrant que les spécifications MT peuvent également être visualisées graphiquement.

4.2.3.3 Visualisation de la transformation

Le jeu PacMan utilise généralement trois types d'animations pour différentes situations :

- Persona moves. En règle générale, une animation spécifique peut être jointe à chaque mouvement, car ils sont traditionnellement encodés dans des règles/opérations différentes (par exemple.*moveR* comme spécifié dans La Section 4.2.3.1, mais aussi *moveL*, *moveU* et *moveD* pour se déplacer à gauche, de haut en bas). Selon que le MTL autorise ou non l'utilisation de classes abstraites (qui seraient Persona ici) pour la spécification MT, ces règles/opérations peuvent être dupliquées.
- PacMan mange un Cookie. Cela se produit lorsque PacMan se trouve sur une Cell contenant un Cookie, le faisant disparaître et déclenchant une mise à jour de Score.
- PacMan est mangé par un Ghost. Cela se produit lorsque PacMan se trouve sur la même Cell qu'un Ghost, ce qui peut se produire lorsque Persona se déplace vers une Cell déjà occupée.

Pour résumer, nous aurions à définir quatre actions différentes pour transformer complètement le DSL PacMan :

- **PM.1** Un Persona disparaît d'une Cell et réapparaît sur une autre Cell (adjacente)
- **PM.2** En supposant que PacMan se trouve sur une Cell contenant un Cookie, le Cookie disparaît.
- **PM.3** Le Score est mis à jour par un incrément prédéfini.
- **PM.4** En supposant que PacMan et un Ghost soient sur la même Cell, PacMan disparaît.

Notez que ces animations ne sont pas totalement indépendantes. Tout d'abord, **PM.2** et **PM.3** doivent être menés séquentiellement assez rapidement pour ne pas remarquer de décalage temporel. Deuxièmement, **PM.2** et **PM.4** semblent être de nature très similaire : ils supposent tous deux que deux objets sont situés sur la même Cell avant de faire disparaître l'un d'entre eux.

4.3 Vue Globale de notre approche MTBE

Pour résoudre le problème de transformation des modèles à base des exemples, nous avons proposer une approche basées sur la réutlisation des traces de transformation (voir La Figure 4.5).

Notre approche prend en entrée un modèle source ($\mathbf{M}_{src}$) conforme à un méta-modèle source pour avoir un modèle cible conforme à méta-modèle cible aprés une spécification de notre transformation qui est écrite avec un langage de transformation (ensemble de règle). Durant cette execution en vas utilisé un listner qui interrompre l'execution pour qu'il prend le modèle avant et le modèle aprés et l'etape de l'execution pour les stocké dans un fichier CSV (Traces de transformation),avant de prendre une dicision on vas utiliser un Analyseur qui vas faire le chargement des traces de transformation pour prendre une dicision .

Le principe de cette solution est inspiré de concept de l'apprentissage par renforcément (Reinforcement Learning) [9]. La Figure 4.6 illustre le principe de de l'apprentissage par renforcement. Ce type d'apprentissage est une technique qui permet à un agent (par exemple Pacman) d'apprendre dans un

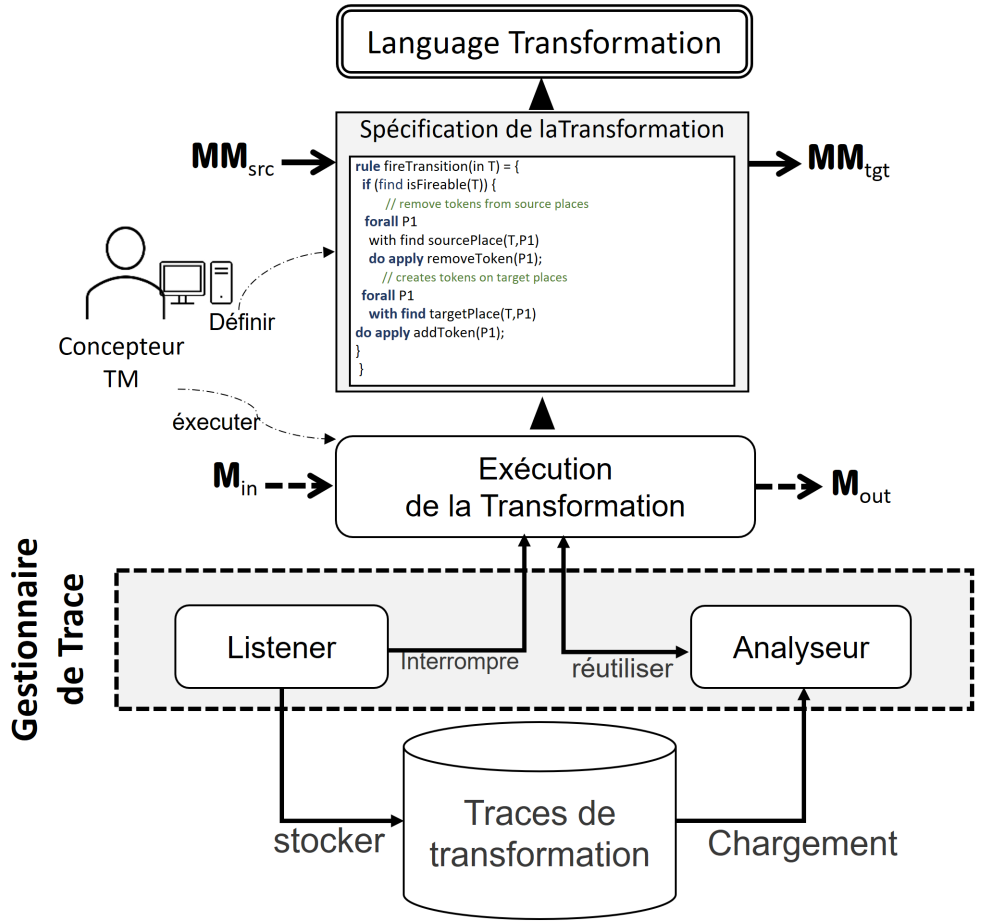

FIGURE 4.5 – Vue Globale de notre approche.

environnement interactif par essais et erreurs en utilisant les commentaires de ses propres actions et expériences.

Apprentissage par renforcement est une classe d'algorithmes d'apprentissage automatique (*machine learning*) qui apprennent des solution suite à des problèmes, dans lesquels il y a peu de rétroaction (*feedback*). Le processus d'apprentissage est piloté par un agent qui exécute actions dans son environnement ; et traite ensuite la rétroaction de l'environnement : le nouvel état de l'agent, et un signal de récompense (*reward signal*).

4.4 Vue conceptuelle de notre approche

D'un point de vue conceptuel, comme illustre La Figure 4.7 notre approche est organisé en trois package :

- **La syntaxe abstraite :** ce package décrit les éléments statique du métamodel utilisé,
- **Execution de métamodel :** Ce package représente la partie dynamique de DSL. Cette partie représente les elements dynamique qui peuvent être changés durant l'exécution de la transforma-

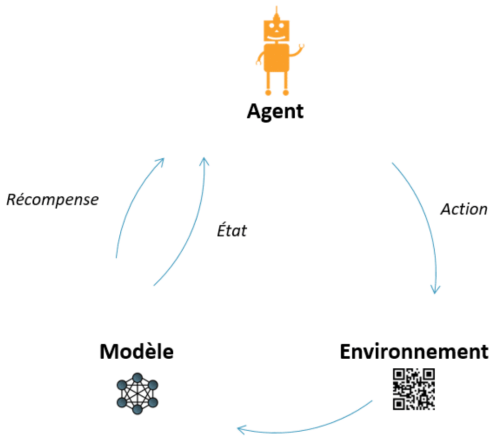

FIGURE 4.6 – Principe de l'apprentissage par renforcément (Reinforcement Learning).

tion.

— **L'historique (Journal) :** Ce package montre la structure de l'historique de l'exécution qui doit être capturé pour stocker les états de modèle et les étapes de l'exécution.

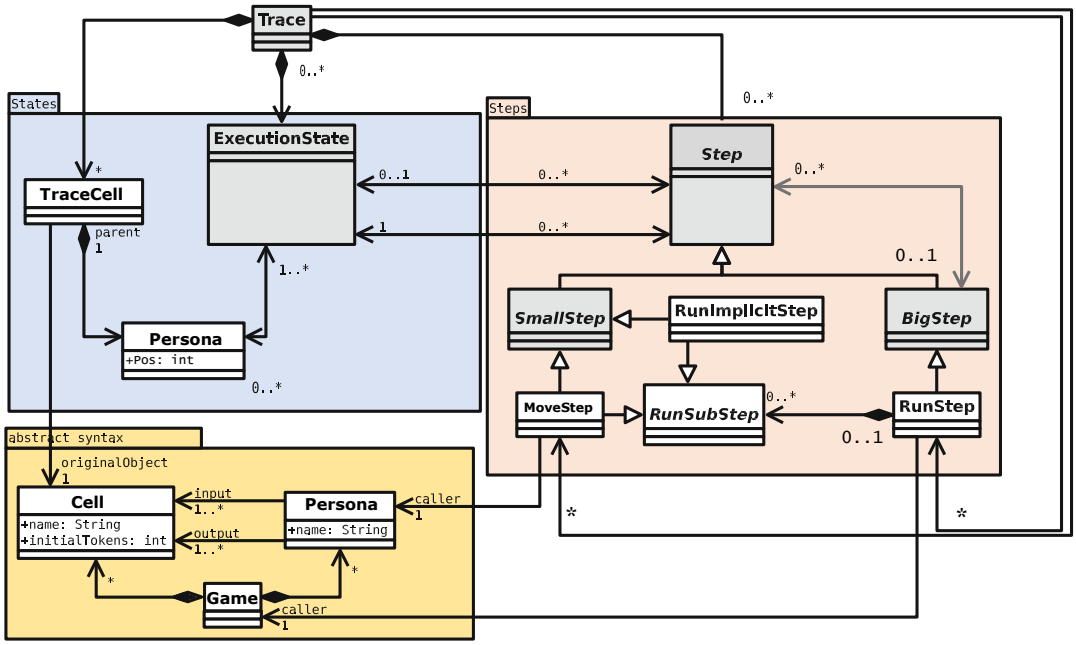

FIGURE 4.7 – Vue conceptuelle de notre approche MTBE (adaptée à partir des travaux de [8]).

4.5 Etapes de notre approche

Dans cette section, nous allons détailler les étapes de notre approche qui sont principalement : (1) Définition de DSL et (2) Définition de la sémantique Opérationnelle (Transformation M2M). Noter que la partie visuelle du DSL (la syntaxe concrète) est non traité dans ce travail.

4.5.1 Définition de DSL

Cette étape traite de la définition de la syntaxe du DSL avec les deux notations : (i) Métamodélisation via l'éditeur visuel et (ii) Métamodélisation via l'éditeur textuel.

4.5.1.1 Métamodélisation via l'éditeur visuel

Le méta-modèle source et cible peuvent etres représenter sous forme visuelle en utilisant EMF-Ecore. Ce modèle est basé sur la spésification de la structure des méta-modèles,ce dernier joue un rôle clé pour les développements de DSML car c'est à partir des informations issues des méta-modèles écrits en Ecore que seront construits les outils qui manipulent les modèles. Pour une édition plus avancée

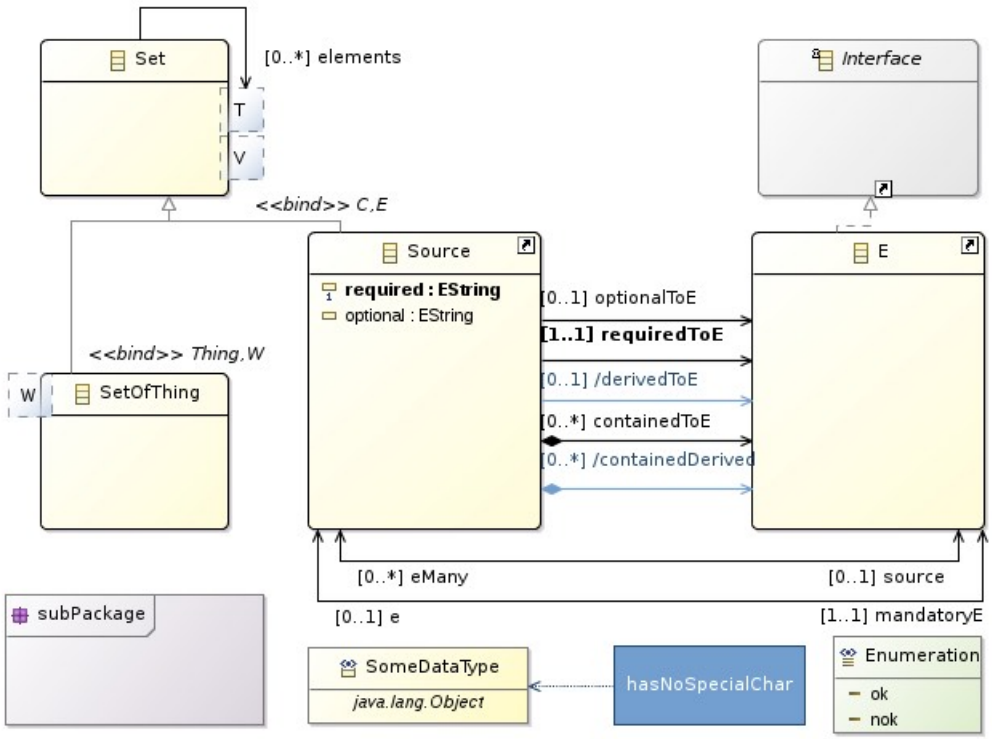

Figure 4.8 – EcoreTools-Modélisation graphique pour Ecore.

des métamodèles, on utilisera de préférence l'éditeur graphique fourni par le projet Eclipse Ecore Tools [9].Comme c'est ulistrée dans La Figure 4.8 Cet éditeur offre la possibilité d'éditer graphiquement

9. https://www.eclipse.org/ecoretools/

un métamodèle sous la forme d'un diagramme de classe. Pour cela cet éditeur complète le métamodèle sérialisé au format Ecore avec un second fichier ayant l'extension (.ecore). (les informations permettant de distinguer les différents états du modèle au cours de son exécution, en fonction du niveau d'abstraction choisi), Elle considère donc comme domaine sémantique une extension du domaine syntaxique ecorediag et stockant les informations du diagramme, c-a-d., de la vue graphique.

4.5.1.2 Métamodélisation via l'éditeur textuel

Le langage Xtext consiste à définir une syntaxe concrète textuelle pour un DSML à travers une grammaire qui doit pouvoir être analysée en LL(k). En particulier, elle ne doit pas récursive à gauche. Généralement,xText est utilisé aussi pour engendrer le méta-modèle correspondant. C'est ce que nous allons voir dans un premier temps. Cependant,xText peut aussi travailler à partir d'un méta-modèle existant. Cependant, il ne faut pas que la distance soit trop grande entre le méta-modèle existant et la syntaxe textuelle souhaitée. Dans ce cas défavorable, il est conseillé de définir un méta-modèle adapté à la syntaxe concrète (par exemple engendré automatiquement) puis d'écrire une transformation de modèle pour obtenir un modèle conforme au méta-modèleexistant du DSML.

4.5.2 Définition de la sémantique Opérationnelle (Trans. M2M)

La sémantique opérationnelle s'exprime sur une représentation abstraite similaire à celle de la syntaxe abstraite du langage considéré, à laquelle est rajouté les informations que l'on souhaite capturer sur l'exécution (i.e., les informations permettant de distinguer les différents états du modèle au cours de son exécution, en fonction du niveau d'abstraction choisi). Elle considère donc comme domaine sémantique une extension du domaine syntaxique. Cela permet alors d'exprimer la sémantique comportementale (i.e., comment évolue l'état du modèle) directement sur les concepts du langage dédié au domaine, instanciés à partir des paradigmes utilisés pour la définition de la syntaxe abstraite (dans notre cas, les concepts de métamodélisation).

Ces concepts sont dénués de sémantique mais un langage d'action permet de l'exprimer et ainsi de définir les outils support à l'exécution. Un tel langage permet de décrire l'évolution du modèle et de produire à partir d'un état donné un ou plusieurs autres états (dit successeurs). Dans le contexte des langages naturels, une sémantique opérationnelle revient à manipuler une phrase (le plus souvent mentalement), dans la même langue que celle utilisée pour l'écrire initialement. On lui donne dans ce cas un sens selon l'expérience acquise (propre à chacun).

De manière plus formelle, notons qu'une sémantique opérationnelle est, par définition, fortement bisimilaire au système de transition de référence du système.

4.5.2.1 Définition des règles en utilisant Méta-Programmation (MP)

Les règles de transformation dans notre système sont exprimées avec le paradigme de metaprogramming. Le Listing Liste 4.1 montre un aperçu d'un pseudo code de la règle `moveRight` de DSL Pacman.

```
class PacMan {
...
def boolean moveR() {
if(self.in.right != null){  self.in = self.in. Right}
```

Listing 4.1 – Exemple de rèle MoveR de jeux de Pacman

4.5.2.2 Définition des règles on utilisant Réécriture de graphes (GBT)

Comme nous l'avons indiqué dans le chapitre 1, les règles de transformation peuvent être exprimées avec différents langages d'action comme Kermeta, Xtend et les langages de transformation graphe. Ces langages expriment les règles par NAC, LHS et RHS. Dans notre approche, on considère que la transformation est basée sur les langages d'actions (dans notre cas Xtend/JAVA). Voici un exemple des règles de transformations liées au metamodel de jeux de PacMan (voir La Figure 4.9).

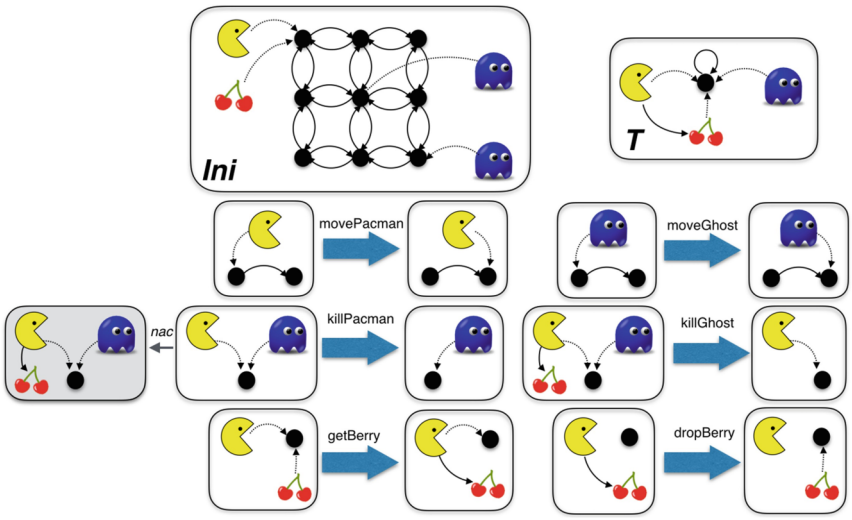

FIGURE 4.9 – Examples des règles de transformation de Jeux de Pacman exprimées en langage graphique (NAS, LHS, RHS).

4.6 Composantes de notre approche

Nous avons analysé les exigences de notre solution proposée dans La Figure 4.5.

- Comment déduire la règle à partir de l'exemple ? -> Adoptée une approche MTBE
- A quoi sert l'exemple de transformation ? -> Trace d'exécution
- Comment représenter les exemples ? -> Besoin d'une structure de stockage
- Besoin de manipulation des traces ? -> Besoin d'un Gestionnaire (ensemble de services)
  - **Mécanisme pour interrompre l'exécution :** Notre approche nécessite un mécanisme pour interrompre l'execution de la transformation en utilisant une instrection de blocage (Breakpoint) pour stocker les traces .

- **Comparaison :** Pour sélectionnerla règle à executer il nous faut une comparaison des modèles (modèle avant et le modèle aprés).
- **Encodage :** Pour manupiler un programme Chaque modèle doit etre encodé
- **Calcule de distance :** Une fois deux modèles sont encodés en peut calculer la distance euclidienne en se basant sur une métrique .
- **Similarité :** le calcul de similarité repose sur le calcule de la distance (Similarité = 1-Distance).
- **Comment implémenter notre solution ?**
  - IDE Eclipse : Xtext, EMF, Xtend, CSV etc.

Nous pouvons diviser notre approche MTBE en deux composantes principales :

1. Représenter des traces de modèles exécutables
2. Fournir un gestionnaire de gestion de trace d'exécution

4.6.1 Structure des traces d'exécution

La Figure 4.10 montre un diagramme de classe UML de l'historique des traces de l'executions de la transformation. Les éléments corps de ce modèle sont : l'historique (classe **history**), les étapes d'exécution (classe **Step**), les changements atomique durant les étapes d'exécution (classe **Change**) et les grandes étapes de l'exécution (class **MacroStep**). Une étape d'exécution est l'application d'une règle d'étape. Une étape d'exécution qui n'est pas composée d'autres étapes est appelée une petite étape, tandis qu'une étape d'exécution composée de plusieurs étapes est appelée une grande étape. Une trace d'exécution est une séquence d'états d'exécution et d'étapes d'exécution (petites et grandes étapes) responsables des changements d'état. Un état d'exécution est l'ensemble des valeurs de tous les champs dynamiques d'un modèle à un certain moment de l'exécution. L'état d'exécution d'un modèle est modifié par l'application des règles de la transformation d'exécution.

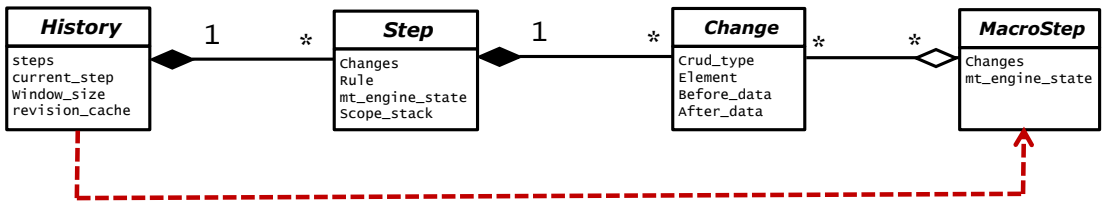

FIGURE 4.10 – historique des traces de l'executions de la transformation.

Les changements atomiques (les changements intermédiaires) qui peuvent être apportés à l'état d'exécution, c'est-à-dire les changements pendant lesquels l'état d'exécution n'est pas cohérent (c'est-à-dire supprimer les jetons des places d'entrée et ajouter des jetons aux places de sortie). Les grands pas de ce changement sont composés de plusieurs étapes d'exécution, si la règle d'étape appelle une autre règle d'étape, alors l'exécution des premieres constitue un grand pas.

Exemple : Les traces d'exécution dans le "Jeux de Pacman"

Dans cet example, nous utiliserons le jeu Pacman comme exemple courant, en particulier pour discuter de la transformation de graphe présenté dans le chapitre 1. L'exemple est représenté sur La

Figure 4.11. Nous utilisons un système de transformation de graphe ayant 4 types de noeuds et 5 types d'arêtes (graphe T). Les règles décrivent comment Pacman et les fantômes peuvent se déplacer (règles *movePacman* et *moveGhost*) ; comment un fantôme peut tuer Pacman (règle *killPacman*, qui a un *nac* - graphique sur fond gris indiquant que Pacman ne peut être tué que s'il ne porte pas de baie) ; comment Pacman peut tuer un fantôme (règle *killGhost*) ; et comment Pacman peut obtenir et déposer des baies (règles *getBerry* et *dropBerry*). Notez que seuls les côtés gauche et droit des règles sont affichés puisque dans ces exemples tous les éléments qui sont affichés des deux côtés sont conservés

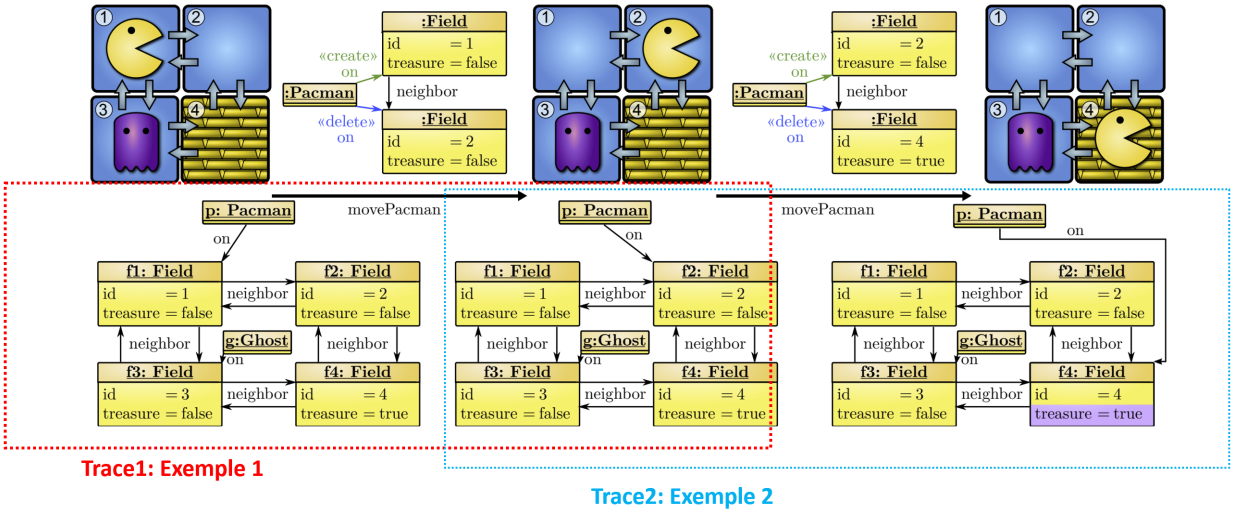

FIGURE 4.11 – Exemples des traces de transformation : Jeux de Pacman.

4.6.2 Gestionnaire des traces d'exécution

Le Gestionnaire des traces d'execution est compose par les services suivants : (*Interruption* (Break-Point), *Creation*, *Chargement*, *Recherche*, *Comparaison* basée sur la différenciation des modèles et le stockage).

4.6.2.1 Listener sur l'éxecution de transformation

A chaque action élémentaire exécuté dans le programme de la transformation, le Listener récupère les deux états du modèle avant et après, aussi l'étape d'exécution procédé par la transformation. La Figure 4.13 montre une illustration des points d'arrêt pour récupérer les états de modèle et l'opération d'exécutions. Le point d'arrêt peut être implémenté techniquement par l'instruction `Thread.sleep` les langages java-like.

4.6.2.2 Analyseur des traces d'exécution

Avec notre approche, nous pouvons contrôler l'exécution de la transformation qui signifie pouvoir la mettre en pause et la reprendre à volonté. Historiquement, les langages de transformation ont fourni un

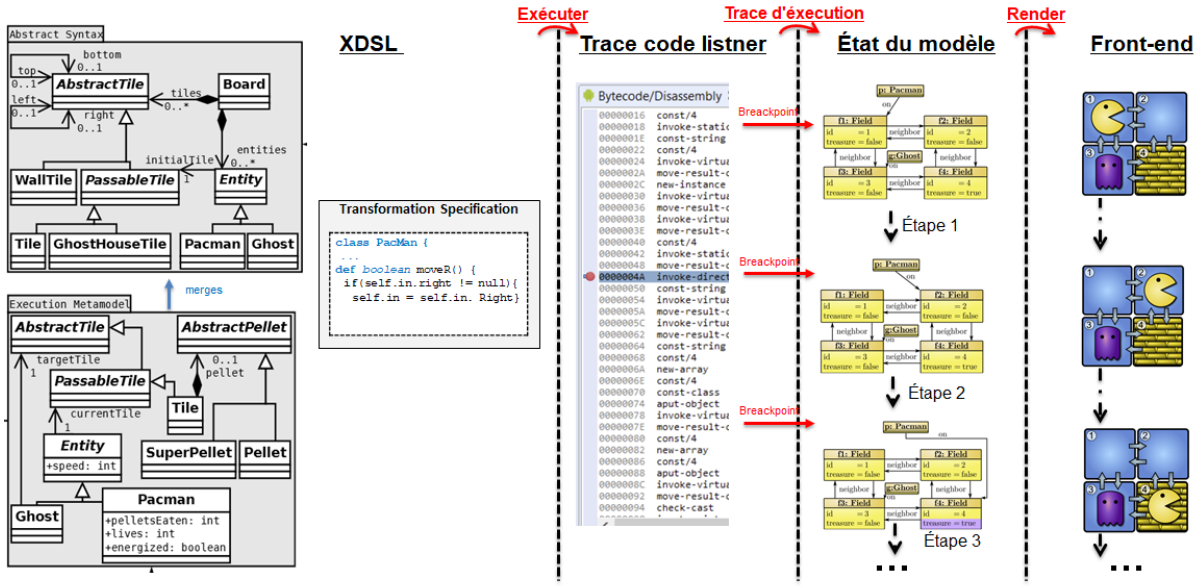

FIGURE 4.12 – Listener sur l'éxecution de transformation.

tel contrôle via des points d'arrêt, qui sont des conditions dans lesquelles l'exécution doit être interrompue (par exemple, atteindre une instruction spécifique). Cela implique que l'exécution est toujours explorée dans un sens ou vers l'autre. Lorsqu'elle est en pause, l'observation est traditionnellement fournie sous la forme de vues, telles que la pile actuelle d'appels de méthode ou les valeurs de toutes les variables existantes.

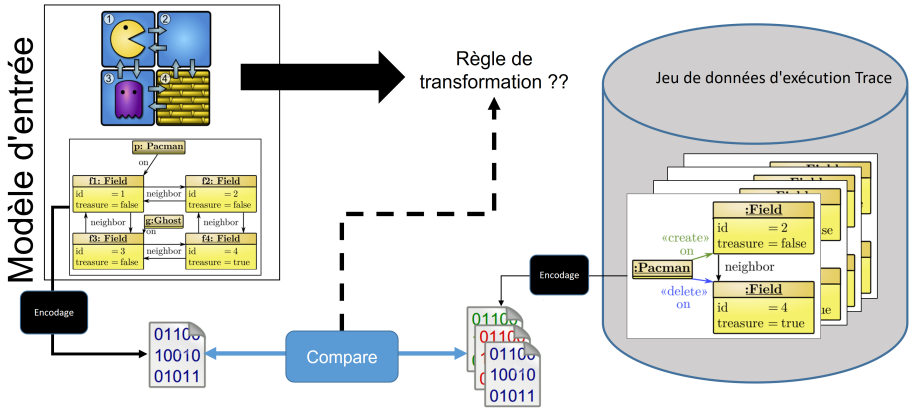

FIGURE 4.13 – Analyseur des traces de transformation.

Principe de codage

Notre approche MTBE nécessite d'abord que les MT soient encodés sous la forme d'un vecteur de caractéristiques capable de manipuler. Formellement, nous représentons le codage d'un TM_i donné comme une structure de Coding Sequence : $CDS_i = < E_i, \mathcal{S}_i^E, \mathcal{R}_i^E, l^{\mathcal{S}_i^E} >$, où nous identifions trois sous-ensembles d'objets $E = \{Class, Attribute, Association\}$ correspondant aux objets modifiables du TM

(i.e. objets pouvant être ajoutés, supprimés et remplacés). Ces objets sont divisés en propriétés/sous-propriétés $\mathcal{S}_i^E$(e.g multiplicités contraintes d'association). La relation $\mathcal{R}_i^E$ représente la relation (c'est-à-dire l'association, l'agrégation, la composition, la dépendance et l'héritage) entre les éléments de E, ($l^{\mathcal{R}^E}$: $E_i \longrightarrow E_j$, E_i, $E_j \in ExE$),par exemple. la classe B est une sous-classe de la classe A. La fonction d'étiquette $l^{\mathcal{S}_i^E}$ attribue une étiquette de chaîne unique à chaque sous-propriété $\mathcal{S}^{E_i}$, par exemple on peut encoder "le type d'un attribut" par la valeur "1" s'il s'agit d'un attribut Key et la valeur "0" sinon. De plus la relation $\mathcal{R}_i^E$ doit conserver les traces de transformation du modèle qui produisent TM_idu donné SM_i. Enfin, notre idée d'encodage permet de générer les CDSde chaqueTM_i solution selon son TMM. Par conséquent, la méthode utilisée pour coder une TM solution doit être adaptée à chaque DSL. En effet, notre cadre sert de recommandations au développeur qui doit affiner manuellement le codage pour qu'il soit spécifique au DSL en question. La Figure 4.14 montre un exemple de CDS (CDS_{y3}) correspondant à un fragment de TM_{y3} de l'exemple motivant.

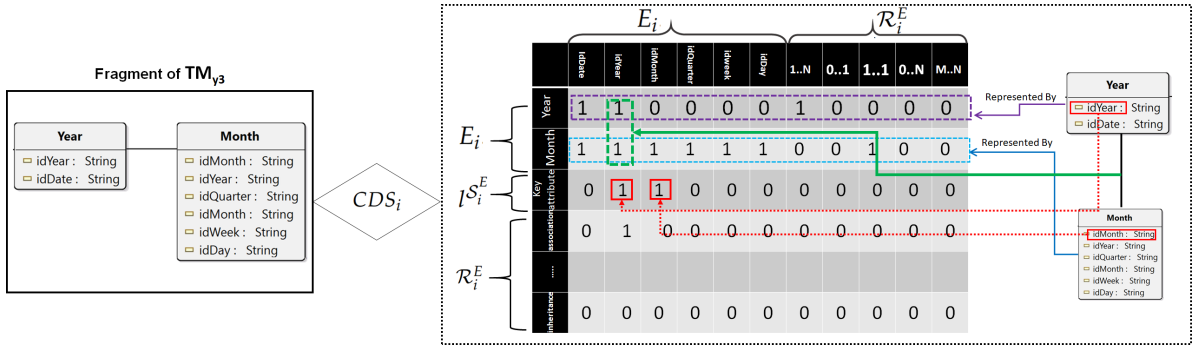

FIGURE 4.14 – Example CDS_{y3} corresponding to a fragment of TM_{y3} .

Calcule de similarité

Notre approche génère la population initiale de l'algorithme génétique en utilisant le CDS de chaque TM_i correspondant à un SM_i égal ou similaire au modèle source (SM_x) à transformer. La génération des candidats TMs est basée sur la similarité entre l'entrée SM_i et le *top-k SM*s similaires dans l'ensemble de données en utilisant la comparaison de modèles (M_i) ou les fonctions de distance spécifiques au domaine proposées dans la littérature telles que EMF Compare [10],Mesures de distance DSL [41] et DSDiffMM [45].

$$Similarity(SM_x, SM_i) = 1 - Dist_{M_i}(SM_x, SM_i) \tag{4.1}$$

La $Similarit$ (SM_x,SM_i) est définie comme une fonction permettant de comparer la similarité entre tous les SM sur la base du CDS de chaque SM. La similarité est calculée sur la base de la distance des données binaires, elle est représentée par une proportion en pourcentage (voir l'équation Équation (4.2)).

$$Dist(CDS_i, CDS_j) = Distance(E_i, E_j) + Dist(\mathcal{S}_i^E, \mathcal{S}_j^E) + Dist(\mathcal{R}_i^E, \mathcal{R}_i^E) + Dist(l^{\mathcal{S}_i^E}, l^{\mathcal{S}_j^E}) \tag{4.2}$$

10. https://www.eclipse.org/emf/compare/

La distance entre chaque paire de composants du *CDS* est calculée sur la base des coefficients de similarité pour les données binaires telles que *Russel and Rao coefficient*, qui est simplement une proportion de correspondances positives[**russell1940habitat**] (voir l'équationÉquation (4.3)).

$$Dist(O_i, O_j) = \frac{a}{a+b+c+d} \tag{4.3}$$

Les valeurs a, b, c et d sont calculées en utilisant *le tableau de contingence*[11], avec (resp. a, b, c, d) = nombre de positions où se trouve i (resp."1", "1","0","0") et j est à (resp. "1", "0", "1", "0"), c'est-à-dire le nombre de fois que les deux objets ont les mêmes occurrences positives.

Exemple 1. Similarity calculation *Considérons deux objets O_i et O_j, nous devons calculer le tableau de contingence pour : O_i = (1, 0, 1, 1, 1) et O_j = (1, 0, 1, 0, 0). Le tableau de contingence est calculé comme illustré à La Figure 4.15. En utilisant le Russel and Rao coefficientformule pour calculer la distance entre les deux objets : a=2, b=2, c=0, d=1, $Distance(O_i, O_j) = 0,4$.*

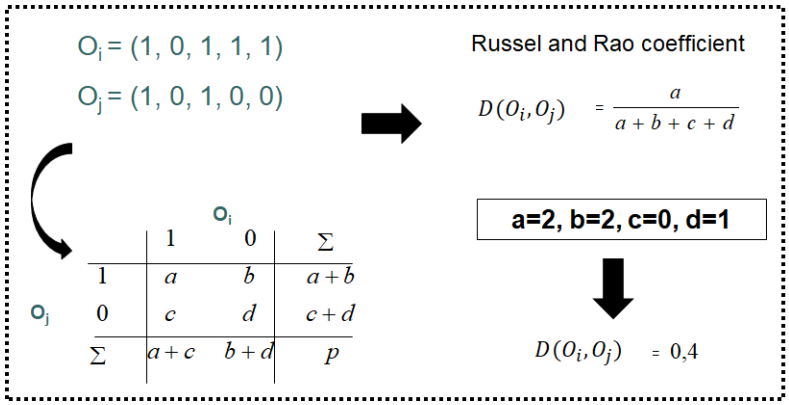

FIGURE 4.15 – Exemple de coefficients de similarité pour les données binaires

4.6.2.3 La base de traces transformation

Nous collectons des données historiques à partir de la transformation des modèles en utilisant le mécanisme d'interruption pour stocker le modèle avant, le modèle après et l'opération d'exécution. Nous effectuons ensuite des séries d'observations dans lesquelles les traces de transformation sont bien choisies en fonction de notre étude de cas du chapitre suivant. Les valeurs de performances consommées par le système sont collectées à l'aide d'un fichier CSV lors de l'exécution de ces transformations de modèle.

11. Le tableau de contingence est un tableau montrant la distribution d'une variable en lignes et d'une autre en colonnes, utilisé pour étudier la corrélation entre les deux variables.

4.7 Conclusion

Dans ce chapitre, nous avons expliqué la démarche suivie et la conception préconisée pour le système selon les sections : conception de la partie structurelle de l'approche, stockage des traces de la transformation, et les services de la manipulation des traces de transformation à base de principaux services (*Interruption* (BreakPoint), *Creation*, *Chargement*, *Recherche*, *Comparaison* basée sur la différenciation des modèles et le stockage). Maintenant que notre système a été conçu, nous pouvons dorénavant entamer la partie réalisation et tests de notre solution, qui feront l'objet du chapitre suivant.

Chapitre

5

Implémentation de l'approche proposée : Application sur "Pacman"

« The true method of knowledge is experiment. »

— William Blake

Sommaire

5.1 Introduction

Aprés avoir présenté notre approche MTBE dans le chapitre précédent, dans ce chapitre nous allons présenter une étude de cas qui sur un jeu de découverte PacMan (DSL Pacman). Notre choix est motivé parla nature de ce modèle qui est dynamique. Les différents états de ce jeux peuvent étres modeliser par un diagramme état transition.

Nous présentons ce chapitre en utilisant des captures d'écran pour montrer les principales fonctionnalités. Nous commençons par présenter les outils et technologies utilisées pour le développement, ensuite nous passons à l'architecture technique de la solution puis l'architecture du code.

5.2 Etude de cas : Jeux de Pacman

Nous avons choisis comme une étude de cas le jeux de Pacman. Dans le jeu Pacman original, les fantômes alternent entre les deux modes de jeu (attaque et dispersion) plusieurs fois jusqu'à rester finalement dans un état permanent d'attaque. Chacun des quatre fantômes a sa propre stratégie d'attaque unique, et chacun se dirige vers son propre coin "maison" quand diffusion. Lorsqu'un fantôme atteint une intersection, il décide d'une cellule cible. Le fantôme se déplace ensuite dans une direction pour minimiser sa distance horizontale ou verticale à cette cible (selon la plus grande). Lors de l'attaque, la cible du fantôme est la position actuelle de Pacman. La cible du fantôme rose est la cellule quatre positions devant Pacman. La cible du fantôme bleu est la position telle que la cellule deux positions devant f Pacman est le point médian entre Pacman et la cible du fantôme. Les fantômes reprennent un comportement pseudo-aléatoire pendant leurs déplacement.

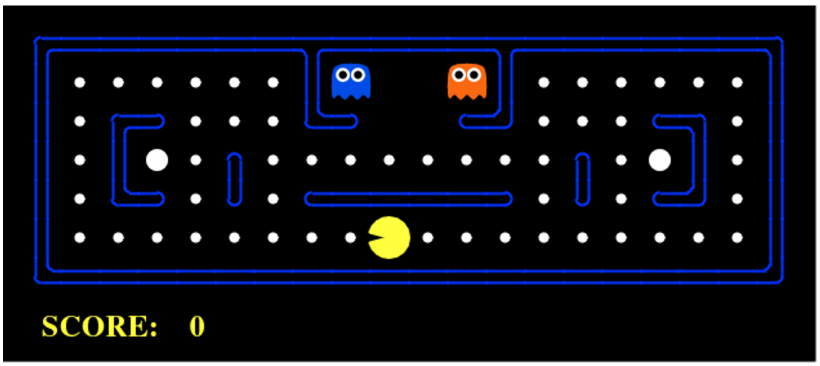

FIGURE 5.1 – Etude de cas : Jeu de Pacman

5.3 Technologies utilisées

Nous avons rencontré des difficultés pour se familiarise avec les outils eclipse EMF, dans cette section nous allons présenter les technologies utilisées avec une description collectée à partir de la documentation. Le détail présenté est pour les lecteurs et les prochaine étudiants de master qui veulent travailler sur ce domaine.

Nous avons utiliser plusieurs outils (Figure 5.3), voici les outils utilisé :

- Xtext :
  - Pour comparer les modèles et la distances entres eux.
  - Il est permet d'implémenter notre propre DSL textuel ou même de mettre sur pied un langage de programmation générique (un : general-purpose programming language).
  - Il est constitué de plusieurs API qui nous permettent de décrire les différents aspects du langage que nous voulons créer et propose une implémentation complète de ce langage.
- Sirius :
  - Pour implementer la syntaxe concrète (l'interface graphique).
- Xtend :
  - Utiliser pour faire notre transformation .
- CSV :
  - Pour stocker toutes les trace d'execution de notre approche .

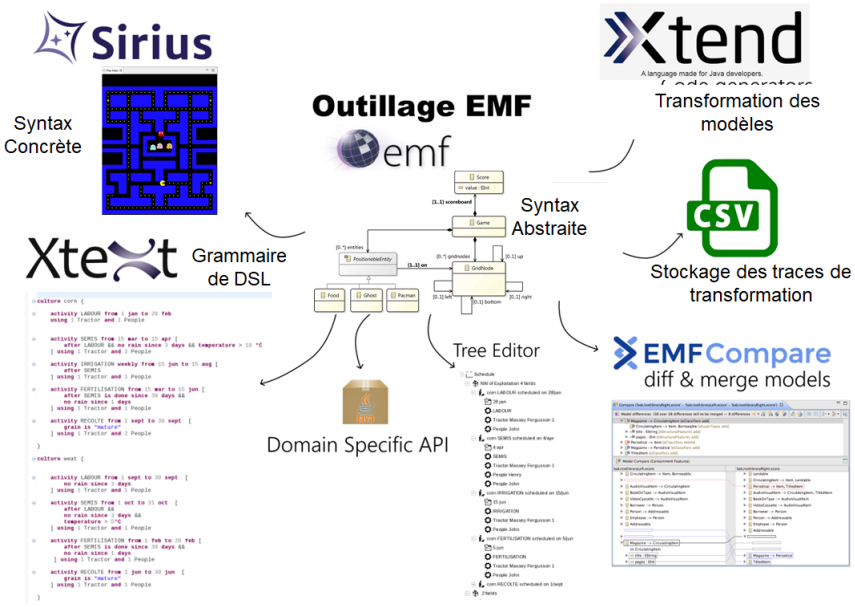

FIGURE 5.2 – Panoplie de l'outillage EMF

5.3.1 Architecture IDE

Comme déjà mentionné, l'IDE lui-même est basé sur la plate-forme Eclipse. Il est indispensable de savoir quelle est l'architecture de la plateforme (voir Figure 5.3).

Lorsqu'un utilisateur démarre une instance Eclipse, la plate-forme runtime recherche dans le dossier du plug-in Eclipse et inspecte plug-ins qu'il contient. Chaque plug-in Eclipse possède un manifeste XML. Un manifest est un fichier avec des dépendances déclarées,Les points de connexion définissent les connexions requises par un plug-in et les connexions fournies par le plug-in.

Lorsque l'environnement d'exécution de la plate-forme examine les plug-ins, il décide lequel charger. Il est bon de savoir que tous les plug-ins sont chargées au démarrage. Cette stratégie de chargement a des avantages multiples.

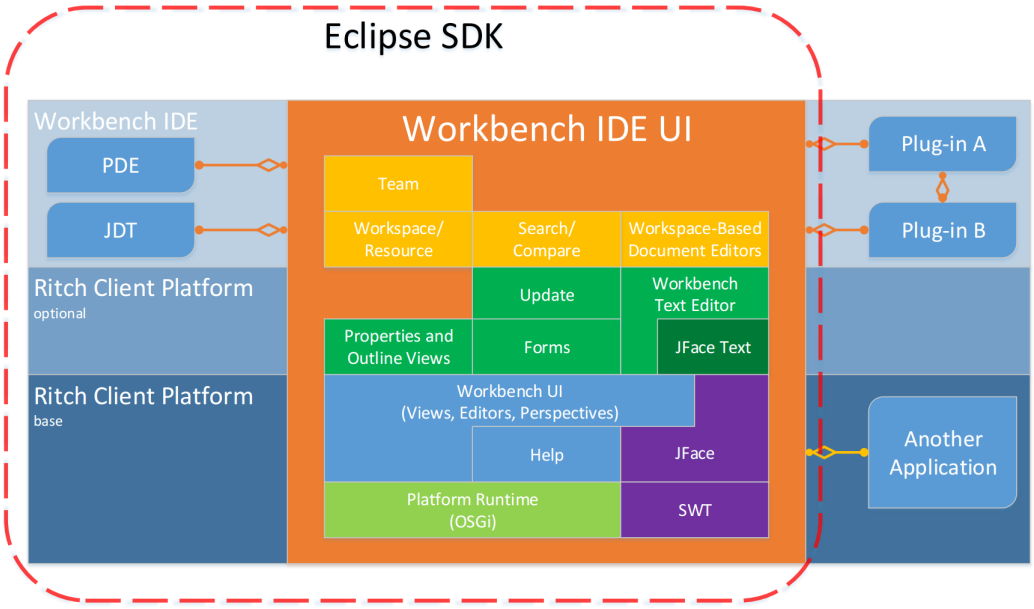

FIGURE 5.3 – Architecture de IDE Eclipse.

5.3.2 Architecture concrète de IDE

La Figure 5.7 montre l'Architecture concrète de l'IDE eclipse, dans cette architecture, le plug-in CTRL a sa propre perspective, qui donne aux utilisateurs la mise en page par défaut, adaptée aux projets (WinCC OA) développement, et il apporte de nouvelles vues et éditeurs de code.

Le framework Xtext, il offre une excellente interopérabilité avec les Plate-forme Éclipse. Par défaut, il donne aux développeurs standard une mise en oeuvre de la vue générale, coloration de la syntaxe, formatage du code, assistance de code, outil de recherche et bien d'autres. Le framework lui-même (Xtext) est profondément intégré à Eclipse, et en même temps, il sépare le développeur de l'écriture du code " code Eclipse". Par conséquent, les développeurs n'ont pas besoin d'en savoir beaucoup sur la plate-forme Eclipse, jusqu'à ce qu'ils doivent développer quelque chose que Xtext ne supporte pas.

5.3.3 La pile logicielle IDE (IDE Software Stack)

Le logiciel lié à Eclipse a une certaines dépendances qui doivent être prises en compte lors du développement et de la distribution. Connaître ces dépendances est également important non seulement pour la programmation, mais aussi pour la planification future. Afin de planifier le développement futur, les développeurs et les dirigeants de ce projet, devraient être en mesure de suivre les technologies et composants utilisés dans la pile logicielle IDE. Si un composant est sur le point d'être obsolète, non pris en charge ou la compatibilité avec les futures versions va se rompre, le processus de planification devrait tenir compte de ces changements.

Dans le cas de la pile logicielle IDE (illustrée à La Figure 5.5), la plupart des composants dépendent de la machine virtuelle Java.

La machine virtuelle Java ou JVM (Java Virtual Machine) utilisée a être compatible avec Java 8 et

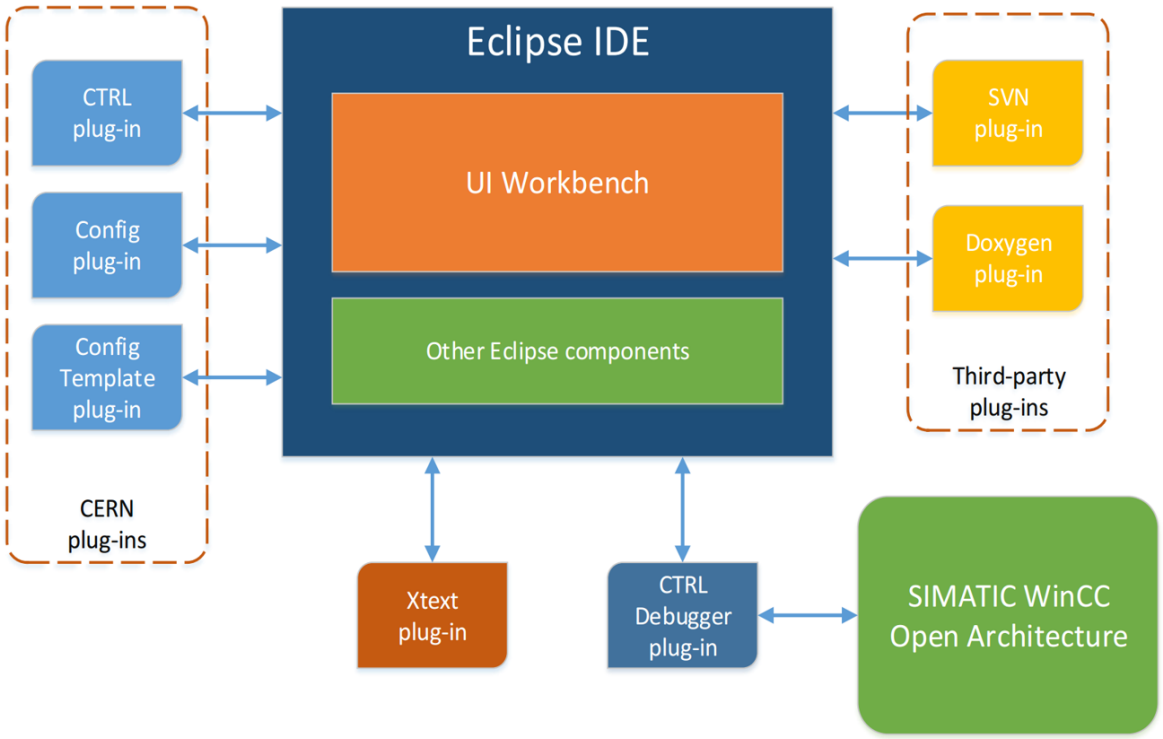

FIGURE 5.4 – Architecture concrète de l'IDE.

son objectif principal est de fournir un environnement d'exécution pour la plate-forme Eclipse. Outre la dépendance de la plate-forme Eclipse à la JVM, il existe également une dépendance cachée au système d'exploitation sous-jacent. C'est parce que l'interface graphique de la plate-forme utilise SWT, et SWT doit être implémenté pour chaque système d'exploitation Eclipse doit fonctionner. Jusqu'à la version 2.9, Xtext dépendait uniquement d'Eclipse. Actuellement la dépendance à Eclipse n'est pas obligatoire, puisque Xtext 2.9 est également disponible pour IntelliJ IDEA.

Au niveau des plug-ins du CERN, il y a une forte dépendance à Xtext. Cette dépendance est là car la plupart des plug-ins utilisent Xtext/ANTLR pour générer un analyseur, et le framework Xtext lui-même est utilisé pour implémenter des fonctionnalités spécifiques à DSL.

5.3.4 Structure de travail (The Workbench)

Le Workbench est la partie graphique d'Eclipse, autrement dit c'est l'interface utilisateur basée sur SWT/JFace. Il affiche des barres d'outils, des menus, des perspectives, des vues et des éditeurs.

La structure de l'éclipse Workbench est la suivant : au niveau le plus élevé, il y a la fenêtre Workbench. Ceci est la fenêtre d'application de Instance en cours d'exécution d'Eclipse. Si l'instance s'exécute sur Microsoft Windows, il y a des boutons de réduction, de maximisation/restauration et de fermeture en haut dans le coin droit. Au niveau inférieur, il y a la WorkbenchPage. Son but est de fournir le contenu de la fenêtre. Théoriquement, il peut y avoir un nombre arbitraire de pages Workbench dans une fenêtre Workbench. Pratiquement, il n'y a presque toujours qu'une seule instance de WorkbenchPage. Si nous

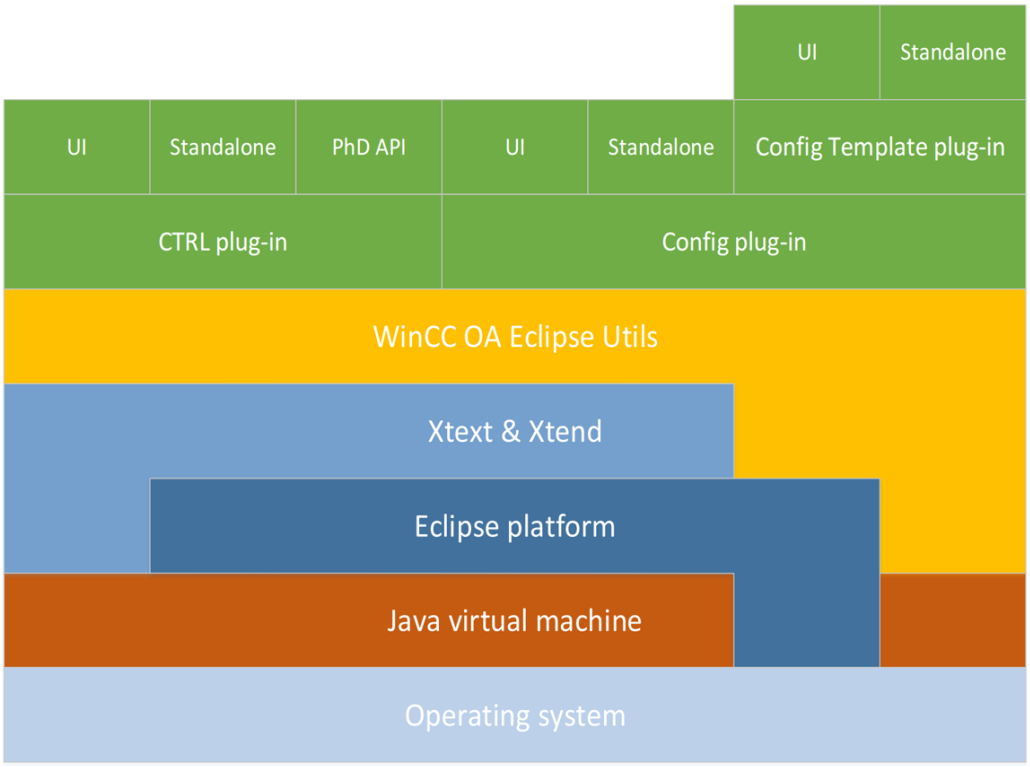

FIGURE 5.5 – Dépendances logicielles de l'IDE. Le logiciel des couches supérieures dépend du logiciel des couches inférieures.

descendons dans la hiérarchie , on voit que le WorkbenchPage contient une instance de ViewFactory et un ensemble de perspectives. Pour l'instant, On le fera sur les perspectives Eclipse. Du point de vue de l'utilisateur, une perspective contient un éditeur (par ex. éditeur de code Java, éditeur de code CTRL, etc.) et des vues (vue Plan, Explorateur de packages, etc.).

5.3.5 Le métamodèle Ecore

Au coeur d'EMF se trouve le (méta-)métamodèle Ecore. C'est le métamodèle qui doit etre conforme aux bases d'EMF. Selon le chef de projet d'EMF Ecore, c'est l'implémentation de référence la norme Essential MOF (EMOF), qui est une version réduite de la norme Complete MOF (CMOF). Ecore utilise un sous-ensemble de la syntaxe concrète du diagramme de classes UML (par exemple, il n'y a pas de concept de ('association'). Comme le (C)MOF (méta-)métamodèle, Ecore est auto-défini, ce qui signifie qu'il est son propre métamodèle.

5.3.6 Utilisation du Xtext et Xtend

Xtext et Xtend sont des composants fondamentaux du processus de développement. Les deux outils ont déjà été montioné dans les chapitres précédents,mais ils ont été décrits principalement à partir de perspectives théories . Dans ce chapitre,on vas voir Xtext et Xtend d'un point de vue pratique et on vas décrire notre expérience avec les deux outils.

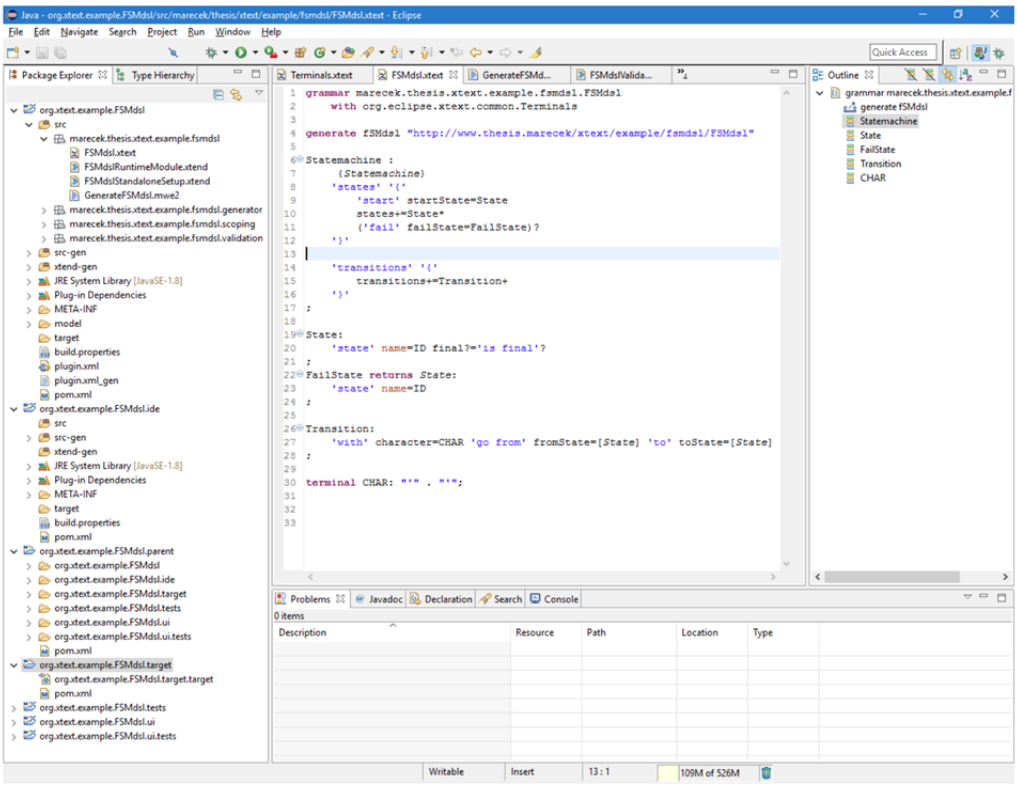

FIGURE 5.6 – Structure d'accueil (Extrait de notre projet)

5.3.6.1 Xtext

Ce cadre est vraiment remarquable et il accélère tout le processus de développement de langages spécifiques à un domaine. Quand il y a un simple DSL qui a besoin d'être intégré avec Eclipse et java,Xtext est le bon outil à utiliser. Par défaut, il fournit de nombreuses fonctionnalités de base :AST, références croisées, aide au code formatage, coloration du code, pliage du code, etc. Tous dérivent automatiquement de la grammaire de la langue. C'était particulièrement pratique lorsque je travaillais sur les plugins, Config et Config Template. Il est pas nécessaire de toucher directement à l'API Eclipse ou de travailler avec SWT. Toutes ces choses sont cachées, donc la courbe d'apprentissage est relativement raide. Un débutant Xtext a pu apprendre rapidement les concepts de base.

5.3.6.2 Xtend

Xtend est un dialecte à typage statique du langage de programmation Java. Il s'intègre bien avec les bibliothèques Java existantes et il se compile en code Java lisible par l'homme, et non en bytecode Java. Selon les options du compilateur, il peut être compilé vers Java version 5 et plus récente. Xtend est livré avec le plug-in Xtext et pour une bonne raison. Comparé à Java,Xtend fournit des fonctionnalités avancées utiles pour le développement DSL :

5.4 Les étapes de notre implémentation

5.4.1 Architecture Technique

La Figure 5.7 présente l'architecture Technique de notre approche. Les étape de cette architecture se divise en trois partie fondamentale :

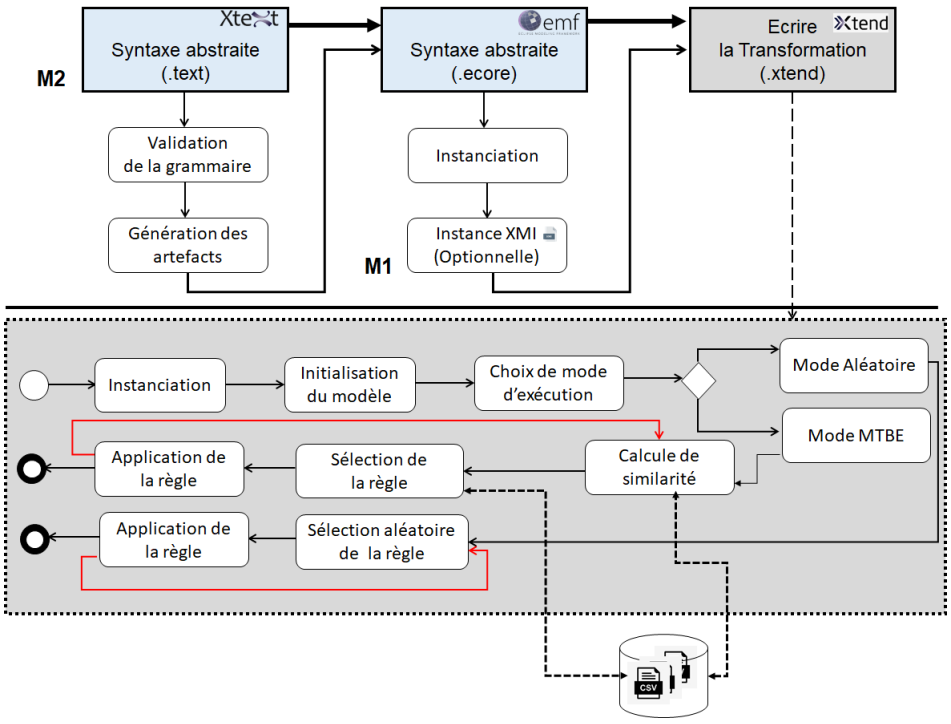

FIGURE 5.7 – Architecture Technique de notre implémentation

5.4.1.1 Définition de la syntaxe abstraite

Le processus commence par la définition de la grammaire de notre DSL (Métamodèle de Pacman). Après la définition de la grammaire, on doit passer à la validation de cette grammaire afin d'identifier les erreurs syntaxiques.

L'interface de cet envirenement est organise en quatre parties (A) :fichiers de projet, (B) : Apercu les classes et des méthodes de projets, (C) :Le code, et (D) :Le console.

La Figure 5.8 illustre l'environnement de l'implémentation Eclipse [12].

5.4.1.2 Grammaire de Pacman

Voici la grammaire de DSL Pacman.

12. https://www.eclipse.org/

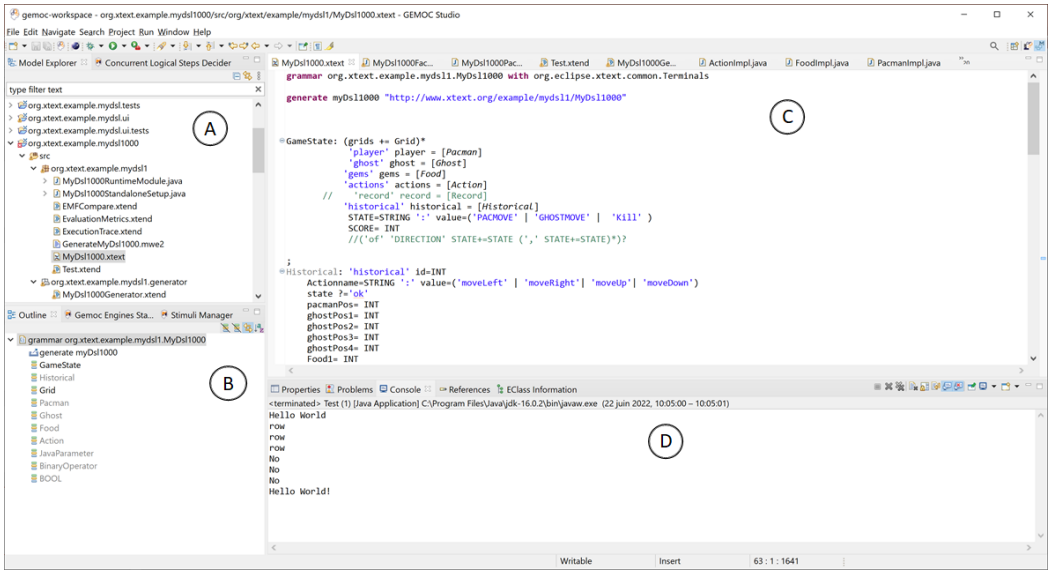

FIGURE 5.8 – Environment de l'implémentation

```
1
2  org.xtext.example.mydsl1.MyDsl1000 with org.eclipse.xtext.common.Terminals
3
4  generate myDsl1000 "http://www.xtext.org/example/mydsl1/MyDsl1000"
5
6  GameState: (grids += Grid)*
7          'player' player = [Pacman]
8          'ghost' ghost = [Ghost]
9           'gems' gems = [Food]
10          'actions' actions = [Action]
11     //      'record' record = [Record]
12          'historical' historical = [Historical]
13          STATE=STRING ':' value=('PACMOVE' | 'GHOSTMOVE' |  'Kill' )
14          SCORE= INT
15           ....
16           ......
```

Listing 5.1 – Extrait de la grammaire xtext de jeux de Pacman

5.4.1.3 Syntaxe graphique de la grammaire

Après l'avoir visualisé la grammaire en mode texte, cette dernière peut être visualisée par un arbre syntaxique. Un arbre de syntaxe peut être généré à partir de la grammaire : (Window/Show View/Other.../Xtext/Xtext Syntax Graph).

5.5 Validation et génération des artefacts

L'un des principaux avantages des DSL est la possibilité de valider statiquement des contraintes spécifiques à un domaine. Comme il s'agit d'un cas d'utilisation courant, Xtext fournit un hook dédié pour ce type de règles de validation.

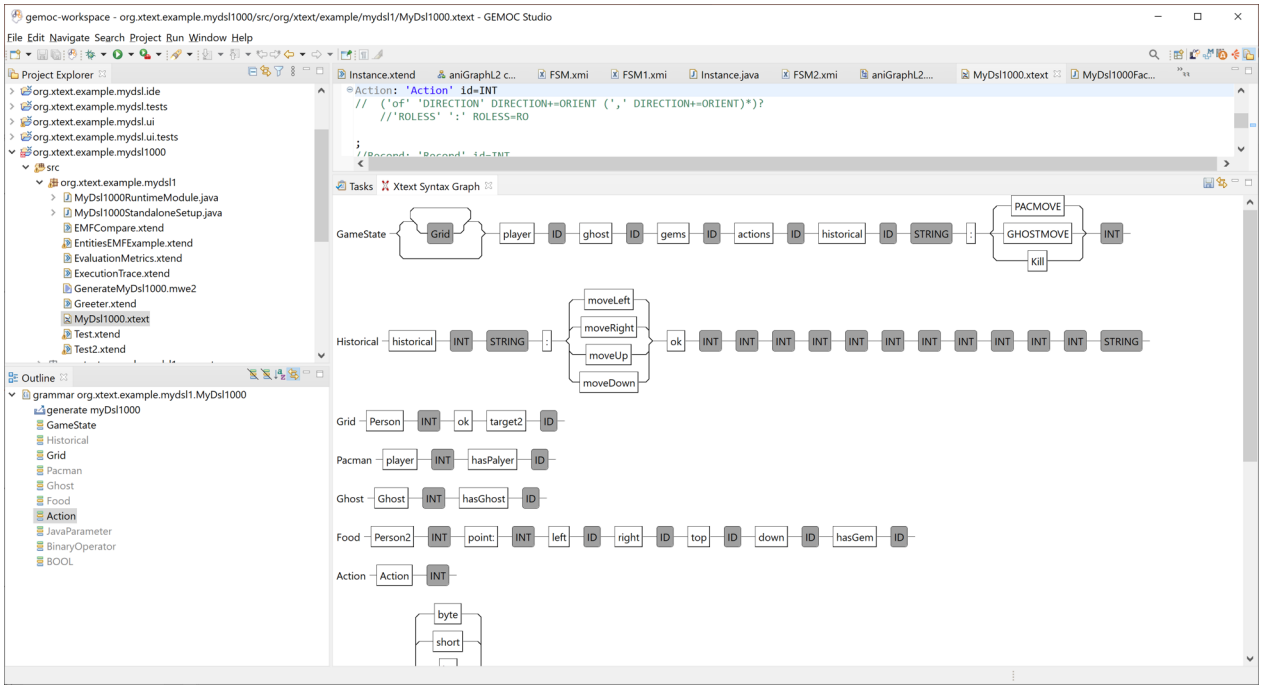

FIGURE 5.9 – Syntaxe graphique de la grammaire.

Localisez la classe `DomainmodelValidator` dans le package *org.example.domainmodel.validation* du projet de langage. Définir la contrainte elle-même n'est qu'une question de quelques lignes de code.

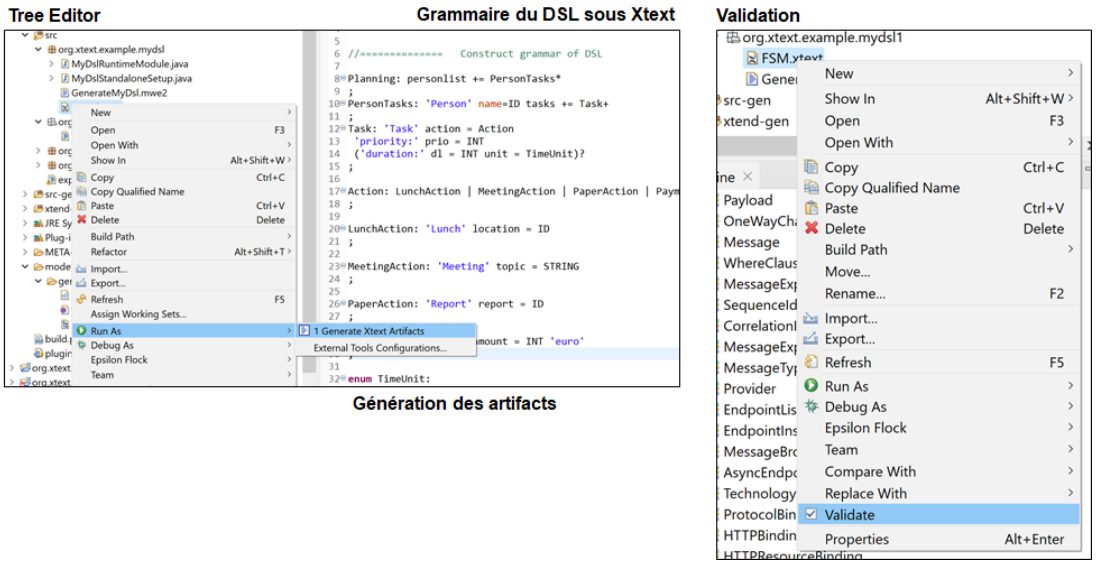

FIGURE 5.10 – Validation et génération des artefacts

5.5.1 Représentation visuelle de Métamodèle de Pacman

Nous utilisons le jeu `Pacman` qui est introduit en tant que syntaxe abstraite. Le jeu est basé sur le métamodèle `Pacman` comme illustré sur La Figure 5.12. Le jeu se compose d'un seul Pacman, d'un fantôme et de zéro ou plusieurs gemmes sur un plateau de jeu (*composé de plus de zéro grilles*). Chaque grille peut contenir `Pacman`, un fantôme et une gemme en même temps. La Le jeu Pacman est contrôlé par le `GameState`, qui enregistre des attributs importants tels que `STATE`, `SCORE` et `GridNode`. Il contient également une liste d'actions. Chaque action définit les mouvements autorisés par Pacman ou le fantôme (`Ghost`) et est exécutée lorsqu'elle a le même cadre que le `GameState`.

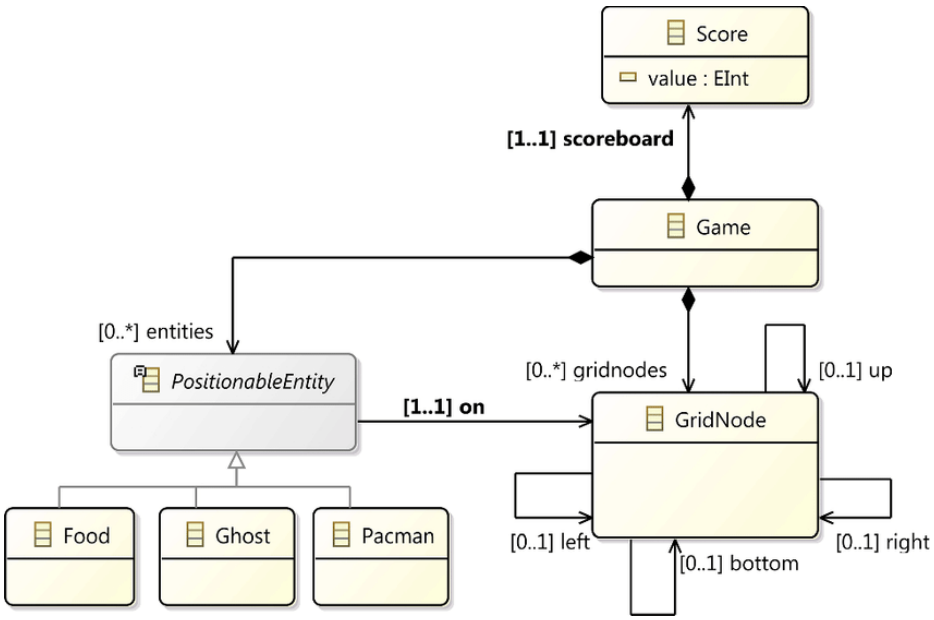

FIGURE 5.11 – Métamodele de Jeux de Pacman.

5.5.2 Implémentation de la transformation

Pour implémenter notre programme de transformation, nous avons utilisé l'API Factory.

```
1 public static myDslPackage getPackage()
2  {
3    return myDslPackage.eINSTANCE;
4  }
```

Listing 5.2 – La classe Factory "eINSTANCE"

5.5.3 Intialisaion de modèle

Cette API offre la possibilité d'accéder au méta classe, les propriétés et les attributs de la grammaire Xtext.

```
1  public EObject create(EClass eClass)
2  {
3    switch (eClass.getClassifierID())
4    {
5      case myDslPackage.GAME_STATE: return createGameState();
6      case myDslPackage.HISTORICAL: return createHistorical();
7      case myDslPackage.GRID: return createGrid();
8      case myDslPackage.PACMAN: return createPacman();
9      case myDslPackage.GHOST: return createGhost();
10     case myDslPackage.FOOD: return createFood();
11     case myDslPackage.ACTION: return createAction();
12     default:
13       throw new IllegalArgumentException("The class '" + eClass.getName() + "' is not a
           valid classifier");
14   }
15 }
```

Listing 5.3 – Intialisation du modèle

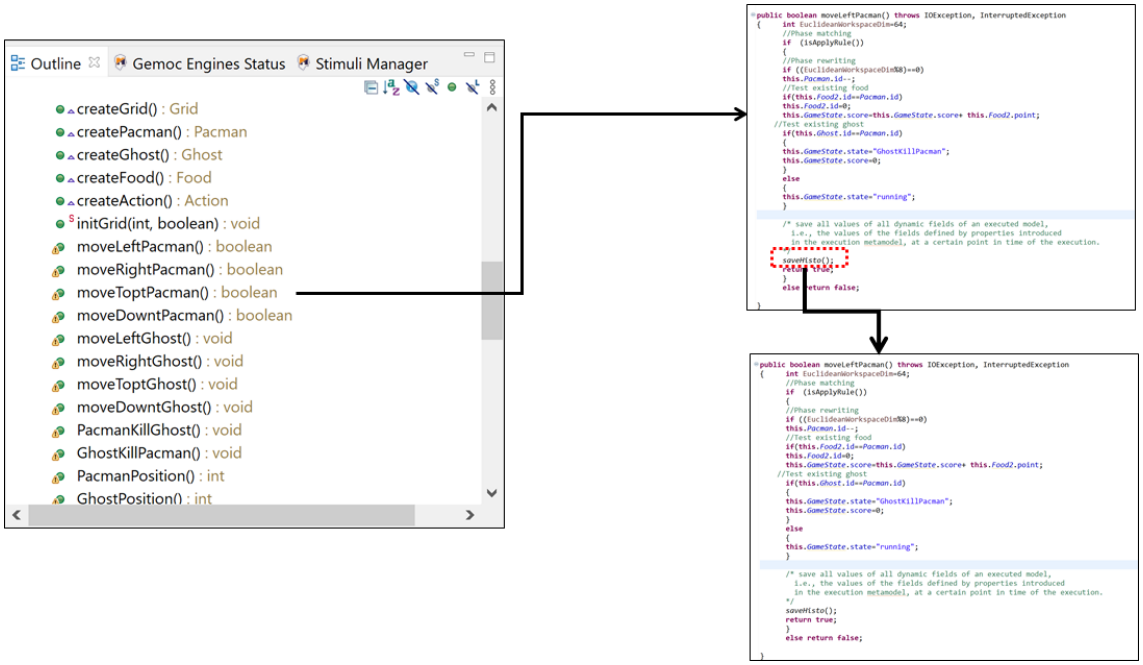

FIGURE 5.12 – Implémentation de la transformation

5.5.4 Implémentation de la sémantique opératinelle

Nous avons implémenté les regles de transformations de modèle pour les deux personnages de jeux : Pacman et son adversaire Ghost.

Le listing Liste 5.4 montre le code source de l'opération `movePacman` qui se déroule comme suit :

1. **La phase de matching (Pré-condition) :** Dans cette phase on vas vérifier si la règle(LHS) ou est confirmé dans notre modèle, ce matching peut étre totale,partiel ou bien non verifier (no matching).

2. **La phase de reécriture :** Dans cette phase on vas retirer,supprimer ou mettre à jour de notre modèle .

3. **La sauvgarde des traces :** Dans cette phase on vas stocker toutes les traces dans un fichier CSV .

```
public boolean moveLeftPacman() throws IOException, InterruptedException
{      int EuclideanWorkspaceDim=64;
    //Phase matching
    if  (isApplyRule())
    {
    //Phase rewriting
    if ((EuclideanWorkspaceDim%8)==0)
    this.Pacman.id--;
    //Test existing food
    if(this.Food2.id==Pacman.id)
    this.Food2.id=0;
    this.GameState.score=this.GameState.score+ this.Food2.point;
  //Test existing ghost
    if(this.Ghost.id==Pacman.id)
    {
    this.GameState.state="GhostKillPacman";
    this.GameState.score=0;
      }
    else
    {
    this.GameState.state="running";
    }

    /* save all values of all dynamic fields of an executed model,
      i.e., the values of the fields defined by properties introduced
      in the execution metamodel, at a certain point in time of the execution.
      */
    saveHisto();
    return true;
    }
    else return false;

}
```

Listing 5.4 – Exemple de l'opération de la transformation du modèle qui correspond au *MoveLeftPacman*

```
1
2  public static double getProcessCpuLoad() throws MalformedObjectNameException,
       ReflectionException, InstanceNotFoundException {
3
4      MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
5      ObjectName name = ObjectName.getInstance("java.lang:type=OperatingSystem");
6      javax.management.AttributeList list = mbs.getAttributes(name, new String[]{"ProcessCpuLoad
           "});
7
8      if (list.isEmpty()) {
9          return 0.0;
10     }
11    // Attribute att = (Attribute) list.get(0);
12   //  Double value = (Double) att.getValue();
13     Double value = 0.5;
14     if (value < 0.0) {
15         return 0.0;  // usually takes a couple of seconds before we get real values
16     }
17     return ((int) (value * getWindows()) / 10000.0);        // returns a percentage value with
           1 decimal point precision
18 }
```

Listing 5.5 – Code de calcule de l'usage de processeur

```
1  Runtime runtime = Runtime.getRuntime();
2
3  NumberFormat format = NumberFormat.getInstance();
4
5  StringBuilder sb = new StringBuilder();
6  long maxMemory = runtime.maxMemory();
7  long allocatedMemory = runtime.totalMemory();
8  long freeMemory = runtime.freeMemory();
9
10 sb.append("free memory: " + format.format(freeMemory / 1024) + "<br/>");
11 sb.append("allocated memory: " + format.format(allocatedMemory / 1024) + "<br/>");
12 sb.append("max memory: " + format.format(maxMemory / 1024) + "<br/>");
13 sb.append("total free memory: " + format.format((freeMemory + (maxMemory - allocatedMemory)) /
       1024) + "<br/>");
```

Listing 5.6 – code de calcule de l'usage de la mémoire centrale

5.5.5 Execution de la transformation

L'exécution de la transformation génère un fichier journal sous forme CSV pour stocker toutes les étapes de l'exécution.

5.5.6 La sauvegarde des traces de transformation

Pour le fichier historique, on capture uniquement les objets dynamiques de grandes traces de manière efficace,c'est-à-dire avec une bonne évolutivité à la fois en mémoire et en temps de manipulation.

5.5.7 L'historique de l'exécution

Toutes les traces de notre transformation sont stocké dans un fichier CSV, La Figure 5.13 illustre une cliché sur l'historique de l'exécution de la transformation.

A	B	C	D	E	F	G	H	I	J	K	L	M
N° Trace	Time	Food Position	Pacman Positio	Ghost 1	Ghost 2	Ghost 3	Ghost 4	Treasures Pos	Action	Is Score increa	is pacmankillgc	NO
1	2022/07/14 01:0		food-2	pacman-3	food-2	ghost-4	ghost-3	ghost-3	ghost-3	treasures-3	moveleft	YES
2	2022/07/14 01:0	food:0	pacman4	food:4	Ghost:08	Ghost:02	Ghost:09	Ghost:05	Treasures:null	move_down()	NO	YES
3	2022/07/14 01:0	food:5	pacman8	food:5	Ghost:00	Ghost:08	Ghost:01	Ghost:05	Treasures:null	move_down()	NO	YES
4	2022/07/14 01:0	food:2	pacman6	food:2	Ghost:05	Ghost:03	Ghost:03	Ghost:09	Treasures:null	move_down()	YES	NO
5	2022/07/14 01:0	food:5	pacman3	food:6	Ghost:03	Ghost:02	Ghost:03	Ghost:04	Treasures:null	move_down()	YES	NO
6	2022/07/14 01:0	food:7	pacman8	food:8	Ghost:09	Ghost:01	Ghost:06	Ghost:00	Treasures:null	move_down()	NO	YES
7	2022/07/14 01:0	food:5	pacman3	food:0	Ghost:04	Ghost:09	Ghost:02	Ghost:01	Treasures:null	move_down()	NO	YES
8	2022/07/14 01:0	food:1	pacman2	food:3	Ghost:04	Ghost:04	Ghost:00	Ghost:02	Treasures:null	move_down()	YES	NO
9	2022/07/14 01:0	food:7	pacman1	food:2	Ghost:01	Ghost:05	Ghost:05	Ghost:09	Treasures:null	move_down()	NO	YES
10	2022/07/14 01:0	food:1	pacman7	food:0	Ghost:05	Ghost:08	Ghost:00	Ghost:04	Treasures:null	move_down()	YES	NO
11	2022/07/14 01:0	food:1	pacman5	food:4	Ghost:02	Ghost:05	Ghost:05	Ghost:05	Treasures:null	move_down()	NO	YES
12	2022/07/14 01:0	food:2	pacman3	food:2	Ghost:02	Ghost:08	Ghost:01	Ghost:08	Treasures:null	move_down()	YES	NO
13	2022/07/14 01:0	food:2	pacman8	food:6	Ghost:00	Ghost:09	Ghost:03	Ghost:00	Treasures:null	move_down()	NO	YES
14	2022/07/14 01:0	food:5	pacman3	food:6	Ghost:02	Ghost:04	Ghost:02	Ghost:08	Treasures:null	move_down()	YES	NO
15	2022/07/14 01:0	food:5	pacman7	food:5	Ghost:03	Ghost:03	Ghost:03	Ghost:04	Treasures:null	move_down()	NO	YES
16	2022/07/14 01:0	food:3	pacman4	food:1	Ghost:09	Ghost:05	Ghost:08	Ghost:02	Treasures:null	move_down()	YES	NO
17	2022/07/14 01:0	food:5	pacman4	food:1	Ghost:00	Ghost:01	Ghost:07	Ghost:06	Treasures:null	move_down()	YES	NO
18	2022/07/14 01:0	food:9	pacman6	food:6	Ghost:00	Ghost:09	Ghost:02	Ghost:02	Treasures:null	move_down()	YES	NO
19	2022/07/14 01:0	food:6	pacman0	food:7	Ghost:00	Ghost:08	Ghost:01	Ghost:07	Treasures:null	move_down()	YES	NO
20	2022/07/14 01:0	food:3	pacman4	food:0	Ghost:00	Ghost:07	Ghost:07	Ghost:02	Treasures:null	move_down()	YES	NO

FIGURE 5.13 – Cliché sur l'historique de l'execution de la transformation.

5.5.8 Mesure de performance de notre approche MTBE

A fin de l'exécution de notre transformation selon le mode MTBE ou aléatoire, on calcule les métriques de l'évaluation pour vérifier la qualité de notre solution. Voici l'exemple :

`Nb-RL :` Le nombre de règles à executer.

`Usage-RAM :` Le taux de l'utilisation de la mémoire centrale (RAM).

`Usage-CPU :` Le taux de l'utilisation de processeur (CPU)

`TempEXE` temps d'execution de programme de transformation

`Nb-RL-Correct :` nombre des règles appliquées correctement (le score augmente)

5.5.8.1 Métriques d'évaluation et discussion des résultats

Afin de calculer les métriques d'évaluation telles que *F-Measure* ($F1_{trans}$), *Accuracy* (ACC_{trans}), *Precision* (P_{trans}) et *Recall* (R_{trans}), nous avons utilisé deux paramètres recommandés, à savoir, les instances de MT correctes de notre approche et les instances de MT incorrectes par rapport aux MT conçues collectivement par les experts en termes d'application correcte et incorrecte des opérations de transformation de modèle (c'est-à-dire fractionner, MoveLeft, MoveRight, MoveUp, MoveDown, PacmanEat).

En particulier, pour chaque type d'opération qui représente le changement effectué sur le *MS* (par exemple, moveLeft, moveRight ou IncreaseScore), nous comptons le nombre d'instance du *MC* qui ont été correctement transformés (TP/TN) et mal prédits (FP/FN). Formellement, nous définissons ces quatre paramètres de métrique comme suit :

1. **Vrai positif (TP) :** Ce paramètre fait référence au nombre de prédictions où notre approche

utilise un type d'opération Op_x (par exemple, MoveRight) pour le processus de transformation de modèle, et l'opération correcte est le même type d'opération (Op_x).

2. **Vrai négatif (TN) :** Ce paramètre fait référence au nombre de prédictions où notre approche utilise un type d'opération différent de Op_x ($\neg Op_x$), et l'opération correcte également un Opération ($\neg Op_x$).
3. **Faux Positif (FP) :** Fait référence au nombre de prédictions où notre appoche utilise un type d'opération différent de Op_x ($\neg\ Op_x$), et l'opération correcte est le type d'opération Op_x.
4. **Faux négatif (FN) :** Fait référence au nombre de prédictions où notre approche utilise un type d'opération (Op_x) alors que les experts en transformation de modèle recommandent un type d'opération différent de Op_x ($\neg Op_x$).

Enfin, nous pouvons calculer les métriques mentionnées ci-dessus en considérant toutes les opérations utilisées.

$$ACC_{trans} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$P_{trans} = \frac{TP}{TP + FP} \tag{5.2}$$

$$R_{trans} = \frac{TP}{TP + FN} \tag{5.3}$$

$$F1_{trans} = \frac{2 * P * R}{P + R} \tag{5.4}$$

Le tableau Tableau 5.1 résume les résultats de la précision, du rappel et de la mesure F1 pour les deux scénarios de transformation du modèle. La précision avec Ref est de 0,96 et le rappel et la mesure F1 atteignent respectivement 0,75 et 0,71, tandis que la précision Sans Ref atteint 0,75 et le rappel et la mesure F1 sont estimés à 0,70 et 0,69 respectivement. Ces résultats démontrent que l'utilisation de notre approche MTBE est avantageuse dans la transformation de modèles et a un impact sur l'amélioration par rapport à une approche aléatoire.

Modèle	Precision	Recall	F-measure	Règles de transformation correctes/incorrectes
Aléatoire.	0.75	0.705	0.692	(82.88%, 7.12%)
MTBE	0.96	0.755	0.713	(81.08%, 18.91%)

Tableau 5.1 – Comparaison des valeurs de métrique *MTBE*

5.5.9 Syntax concrete de notre DSL

Dans notre cas, nous n'avons pas développé la syntaxe concrète du DSL. La syntaxe concrète est une visualisation du DSL qui associé pour chaque objet ou bien une instance des éléments graphiques. Le développement de cette syntaxe est une tâche difficile dans le cadre de MDE. Il existe plusieurs outils dédiés pour le développement de cette syntaxe, nous pouvons citer l'outil GEMOC Siruis. La Figure 5.14 montre le schéma global qui illustre la relation entre la syntaxe abstraite, la syntaxe concrète et la sémantique (transformation).

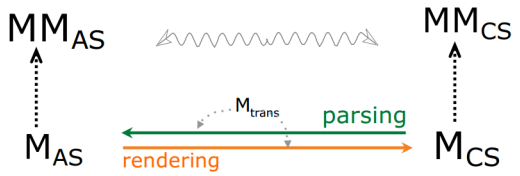

FIGURE 5.14 – La syntaxe abstraite, la syntaxe concrète et la sémantique.

5.6 Conclusion

Dans ce chapitre, nous avons présenté les exigences techniques pour le développement du système. Ensuite, nous avons présenté l'étape de réalisation de notre solution en passant par l'implémentation de chaque composant du système. A la n du chapitre nous avons déni l'architecture technique de la solution pour déployer notre système et nous avons aussi présenté les fonctionnalités et les interactions des différents composants du système.

Quatrième partie

Conclusion et perspectives

Chapitre

6

Conclusion générale et Perspectives

« To accomplish great things, we must not only act, but also dream ; not only plan, but also believe. »
— Anatole France (1844-1924)

Sommaire

6.1 Conclusion et Perspectives

Dans cette section, nous allons présenter la conclusion générale et les travaux futures.

6.1.1 Conclusion

La transformation de modèles (MT) est souvent décrite comme le « cœur et l'âme » de l'ingénierie pilotée par les modèles (MDE). C'est un catalyseur clé pour apporter des capacités de calcul dans MDE. En particulier, le comportement des langages spécifiques au domaine (DSL) peut être facilement capturé et automatisé avec MT. De manière simplifiée, un modèle peut être défini comme un ensemble de données représentant un système et écrites dans un langage bien défini. La transformation des modèles (TM) est une étape primordiale pour l'IDM, elle consiste à transformer d'un modèle source (MS) vers un modèle cible (MC) en se basant sur des règles de transformation , connues appropriées ou bien déduites à partir des traces de transformations. Cette déduction est basée sur les approches de recherches et d'apprentissages machine.

L'intérêt majeur de ce mémoire est d'assister le concepteur/développeur pour effectuer la transformation des modèles. Nous avons effectué une recherche d'informations pour se familiariser avec le sujet, ce qui nous a permis de produire une synthèse bibliographique sur le cœur de ce travail. Nous avons présenté une nouvelle approche basée sur l'apprentissage machine an de faciliter et d'aider le concepteur dans les tâches de transformation du modèle. une approche de transformation inspirée de l'apprentissage par renforcement (*Reinforcement Learning*) qui raffine au fur et à mesure les règles de transformation. Ce type d'apprentissage est une technique qui permet à un agent (par exemple Pacman) d'apprendre dans un environnement interactif par essais et erreurs en utilisant les feedbacks de ses propres actions et expériences.

Le travail présenté dans ce rapport de projet de fin d'étude s'articule dans cinq points essentiels :

1. Proposer une architecture de la solution qui montre les différente briques utilisées.
2. Clarifier les étapes de notre solution proposée.
3. Définition de deux composante principale de la solution : la structure de fichier historique et le gestionnaire des traces d'exécution.
4. Implémentation des algorithmes d'apprentissage machine.
5. Comparaison préliminaire de notre approche avec une exécution aléatoire.

6.1.2 Perspectives

Dans ce travail nous avons rencontré des difficultés de se familiariser rapidement avec les concepts et les outils technologiques de l'IDM. Une autre contrainte est liée à l'absence d'un module orienté vers ce paradigme de modélisation. Ce projet de fin d'étude nous a permis de confronter une nouvelle thématique et de découvrir les nouveaux outils technologiques comme Ecore, Xtext, Xtend, etc. Aussi on a pu avoir distinguer entre Ingénierie de domaine (Domain Engineering) qui concerne le domain specific language (DSL) et ingénierie d'application (Application Engineering) qui concene "a General-Purpose Programming Language" (GPL) comme C++, Java, Ruby, Paython, Rust etc.

Notre travail ouvre plusieurs perspectives, nous pouvons citer :

— Appliquer notre approche sur une grande base d'exemple.
— Implémenter d'autres approches d'apprentissage automatique.
— Développer un outil support à la base de cette approche proposée.
— Appliquer l'approche proposée sur d'autres formalise comme DEVES et le formalisme B.
— Développer la syntaxe concrète de notre DSL Pacman.

Bibliographie

[1] A. C. e. K. A. D. e. H. T. H. e. D. J. A. E. e. H. G. Al Monique DM et Van Houwelingen. « Modèles maternels d'acides gras essentiels pendant une grossesse normale et leur relation avec le statut néonatal en acides gras essentiels ». In : *British Journal of Nutrition* 74 (), p. 55-68 (cf. p. 35).

[2] M. M. e. C. C. André Étienne et Benmoussa. « Traduire des machines à états UML en réseaux de Petri colorés avec Acceleo : un rapport ». In : *arXiv preprint arXiv :1405.1112* () (cf. p. 20).

[3] « Apprentissage du contrôle implicite et explicite dans les transformations de modèle par exemple ». In : *Conférence internationale sur les langages et systèmes d'ingénierie pilotés par les modèles*, p. 636-652 (cf. p. 37-39).

[4] « Approche de programmation génétique pour apprendre les règles de transformation du modèle à partir d'exemples ». In : *Conférence internationale sur la théorie et la pratique des transformations de modèles*, p. 17-32 (cf. p. 37, 39).

[5] K. Asma et B. Meryem. « Une approche dirigée par les modèles basée sur UML pour la mise en œuvre de services web composés ». In : (2017) (cf. p. 24).

[6] B. Baldassari, F. Huynh et P. Preux. « De l'ombre à la lumière : plus de visibilité sur l'Eclipse. » In : *EGC*. 2014, p. 513-516 (cf. p. 19).

[7] B. Benamar et al. « La réingénierie des applications Web vers les données liées. » Thèse de doct. 2019 (cf. p. 16).

[8] E. Bousse et al. « Advanced and efficient execution trace management for executable domain-specific modeling languages ». In : *Software & Systems Modeling* 18.1 (2019), p. 385-421 (cf. p. 50).

[9] B. Bouzy. « Apprentissage par renforcement (3) ». In : *Cours de d'apprentissage automatique* (2005) (cf. p. 6, 48).

[10] *Cadre de modélisation Eclipse : guide du développeur* (cf. p. 19).

[11] C. Cocu. « THESIS/THÈSE ». In : () (cf. p. 20).

[12] B. Combemale. « Ingénierie Dirigée par les Modèles (IDM)–État de l'art ». In : (2008) (cf. p. 19).

[13] M. Dahchour et al. « Extension d'UML par les rôles ». In : *Proc. of the 9th Maghrebian Conference on Information Technologies (MCSEAI 2006), Agadir, Morocco*. 2006 (cf. p. 14).

[14] K. e. A. A. e. S. M. Dehayni May et Barbar. « Quelques approches de transformation de modèles : une revue critique qualitative ». In : *Journal de Recherche en Sciences Appliquées* 5 (), p. 1957-1965 (cf. p. 25).

[15] S. Diaw, R. Lbath et B. Coulette. « Etat de l'art sur le developpement logiciel base sur les transformations de modeles. » In : *Tech. Sci. Informatiques* 29.4-5 (2010), p. 505-536 (cf. p. 25, 26).

[16] M. e. N. C. e. R. P. Dolques Xavier et Huchard. « Apprentissage des règles de transformation à partir d'exemples de transformation : une approche basée sur l'analyse de concepts relationnels ». In : *2010 14 ateliers de la conférence internationale IEEE sur l'informatique distribuée pour les entreprises*, p. 27-32 (cf. p. 37).

[17] X. Dolques, M. Huchard et C. Nebut. « Génération de transformation de modèles par application de l'ARC sur des exemples ». In : *LMO : Langages et Modèles à Objets*. Cépaduès Editions. 2009, p. 61-75 (cf. p. 36).

[18] J.-R. Falleri, M. Huchard et C. Nebut. « Towards a traceability framework for model transformations in kermeta ». In : *ECMDA-TW'06 : ECMDA Traceability Workshop*. Sintef ICT, Norway. 2006, p. 31-40 (cf. p. 18).

[19] M. Faunes, H. Sahraoui et M. Boukadoum. « Generating model transformation rules from examples using an evolutionary algorithm ». In : *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE. 2012, p. 250-253 (cf. p. 37).

[20] I. García-Magariño et al. « A tool for generating model transformations by-example in multi-agent systems ». In : *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*. Springer. 2009, p. 70-79 (cf. p. 30).

[21] D. e. S. M. e. W. R. e. H. B. e. P. L. E. Gierlinger Notburga et Jacques. « Prédiction rapide de la durabilité naturelle du bois de cœur de mélèze par spectroscopie proche infrarouge à transformée de Fourier ». In : *Revue canadienne de recherches forestières* 33 (), p. 1727-1736 (cf. p. 35).

[22] J. e. D. P. Guerra Esther et de Lara. « Spécification visuelle des mesures et des refontes pour les langages visuels spécifiques à un domaine ». In : *Journal des langages visuels & informatique* 19 (), p. 399-425 (cf. p. 19).

[23] « Générer des règles de transformation à partir d'exemples de modèles comportementaux ». In : *Actes du deuxième atelier international sur la modélisation du comportement : fondements et applications*, p. 1-7 (cf. p. 36).

[24] M. e. E. S. e. K. B. Haase Arno et Völter. « Introduction à openArchitectureWare 4.1. 2 ». In : *Forum des implémenteurs d'outils MDD* (cf. p. 22).

[25] I. Hamzane. « Proposition d'une approche de transformation digitale de la composante gestion de projets de la gouvernance IT à base de modèles ». Thèse de doct. Université Hassan II Casablanca (Maroc), 2021 (cf. p. 20).

[26] G. Hegron. « Conception architecturale et modélisation déclarative ». In : *CERMA UMR CNRS* 1563 (2002) (cf. p. 25).

[27] « Immunosuppression et facteur de croissance transformant bêta dans le glioblastome. Production préférentielle de facteur de croissance transformant-beta 2. » In : *Le Journal d'Immunologie* 143 (), p. 3222-3229 (cf. p. 35).

[28] J.-M. Jézéquel, B. Combemale et D. Vojtisek. *Ingénierie Dirigée par les Modèles : des concepts à la pratique...* Ellipses, 2012 (cf. p. 13).

[29] S. Kent. « Model driven engineering ». In : *International conference on integrated formal methods*. Springer. 2002, p. 286-298 (cf. p. 12).

[30] S. e. G.-M. I. e. H. A. Khan Sulaiman et Nazir. « Fusion du big data urbain basée sur l'apprentissage profond dans les villes intelligentes : vers la surveillance du trafic et la fusion préservant les flux ». In : *Informatique & Génie électrique* 89 (), p. 106906 (cf. p. 36).

[31] A. G. Kleppe et al. *MDA explained : the model driven architecture : practice and promise*. Addison-Wesley Professional, 2003 (cf. p. 15).

[32] T. Le Calvar. « Exploration d'ensembles de modèles ». Thèse de doct. Université d'Angers, 2019 (cf. p. 26).

[33] « Modèle de transformation par exemple utilisant la programmation logique inductive ». In : *Modélisation de logiciels & systèmes* 8 (), p. 347-364 (cf. p. 35).

[34] M. NACERA. « UNE APPROCHE HYBRIDE POUR TRANSFORMER LES MODELES ». In : () (cf. p. 19, 27).

[35] G. Paquette et al. « Méthode d'ingénierie d'un système d'apprentissage ». In : *Revue informations in cognito* 8 (1997), p. 1997 (cf. p. 13).

[36] M. P. Y. Pillain. « Touil Amara touilam@ ensieta. fr Encadrants : M. Joël Champeau Et M. Pierre Yves Pillain ». In : (2008) (cf. p. 26).

[37] « Questions de (méta-)modélisation ». In : *Modélisation de logiciels & systèmes* 5 (), p. 369-385 (cf. p. 13).

[38] I. Ragoubi. « Motifs de transformation de métamodeles ». In : *Software Engineering* (2011) (cf. p. 27).

[39] « Rete : Un algorithme rapide pour le problème de correspondance de modèle plusieurs modèles/plusieurs objets ». In : *Lectures en Intelligence Artificielle et Bases de Données*, p. 547-559 (cf. p. 37).

[40] X. e. H.-M. e. N. C. e. S. H. Saada Hajer et Dolques. « Génération de règles de transformation opérationnelles à partir d'exemples de transformations de modèles ». In : *Conférence internationale sur les langages et systèmes d'ingénierie pilotés par les modèles*, p. 546-561 (cf. p. 37).

[41] E. Syriani, R. Bill et M. Wimmer. « Domain-Specific Model Distance Measures. » In : *J. Object Technol.* 18.3 (2019), p. 3-1 (cf. p. 57).

[42] H. V. Teguiak. « Construction d'ontologies à partir de textes : une approche basée sur les transformations de modèles ». Thèse de doct. ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique-Poitiers, 2012 (cf. p. 15).

[43] « Transformation de modèle en ingénierie Web et développement automatisé piloté par modèle ». In : *Journal international de modélisation et d'optimisation* 1 () (cf. p. 25).

[44] D. Varró. « Modèle de transformation par exemple ». In : *Conférence internationale sur les langages et systèmes d'ingénierie pilotés par les modèles*, p. 410-424 (cf. p. 35, 36).

[45] M. Zadahmad et al. « Domain-specific model differencing in visual concrete syntax ». In : *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering*. 2019, p. 100-112 (cf. p. 57).