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

UNIVERSITE IBN KHALDOUN - TIARET

MEMOIRE

Présenté à :

FACULTÉ MATHEMATIQUES ET INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

MASTER

Spécialité : [Génie Logiciel]

Par :

✓ **Ouakid raziqa hanaa**
✓ **Saad hanane**

Sur le thème

Transformation de modèles à l'aide d'Agent intelligent « BDI »

Soutenu publiquement le 28/ 10 /2021 à Tiaret devant le jury composé de:

Mr ABID Khaled	Grade Université UIK MAA	Président
Mr SIABDELHADI AHMED	Grade Université UIK MAA	Encadreur
Mr HATTAB Noureddine	Grade Université UIK MAA	Examinateur
Mr OUARED Abdelkader	Grade Université UIK MCB	CoEncadreur

Type your text

2020-2021

Résumé

L'ingénierie dirigée par les modèles ou « Model Driven Engineering » (MDE) est un paradigme qui permet de réduire la complexité du logiciel par l'utilisation intensive de modèles et des transformations automatiques entre modèles (TMs). L'automatisation des transformations de modèles demeure cependant une tâche ardue. La transformation de modèles à partir des exemples ou « *Model Transformation By Examples* » (MTBE) est une transformation de modèles à partir d'un ensemble de paires de modèles sources et cibles fournis à titre d'exemples . Dans le cadre de ce projet, nous présentons une approche qui prend en charge la transformation exogène est propose plusieurs solutions différentes par plusieurs experts et analyse pour sélectionner le mieux, ce qu'on appele (team work)shared conceptualisation et qui est actuellement utiliser dans les grandes entreprises. la meilleur façons de représenter les experts c'est les SMA (un système distribué composé d'un ensemble d'agents) ou les agents sont les experts.

Mots-clés : ingénierie dirigée par les modèles, transformations de modèles, SMA, MTBE, transformation exogène.

*Remérecimen*t

Tout d'abord, nous adressons notre sincère reconnaissance à Monsieur Siabdelhadi Ahmed qui nous a encadrés et soutenus avec ses conseils et suggestions précieuses durant toutes les étapes de ce projet.

Nous exprimons nos plus vifs remerciements à notre co-encadreur Monsieur Ouared Abd-elkader qui a initié et suivi le thème du présent mémoire. Ses conseils avisés, ses critiques constructives, son suivi permanent, et sa patience furent certes, très favorables à l'aboutissement de ce projet.

Nous remercions les membres du jury pour avoir accepté de juger et d'évaluer ce travail, ainsi que pour l'attention qu'ils nous ont accordée.

Toute notre gratitude et reconnaissance à nos très chers parents pour le soutien moral et affectif qu'ils nous ont apporté tout au long de cette expérience. Nous leur dédicaçons ce travail.

Enfin, nous remercions chaleureusement tous nos frères et nos sœurs et nos amis, ainsi que nos familles pour nous voir encouragés durant la réalisation de ce projet.

Table des matières

Liste des figures

Liste des tableaux

Première partie

Introduction Générale

Chapitre

1 Introduction Général

Sommaire

1.1 Introduction

La transformation de modèle est un élément clé de l'approche d'ingénierie dirigée par les modèles (MDE). Ce processus transforme un modèle d'entrée (modèle source SM) en un modèle de sortie (modèle cible), on peut citer par exemple, une transformation de modèle classique qui transforme un modèle UML en un modèle relationnel où un ensemble d'actions sont effectuées telles que (diviser, supprimer, créer ...). L'écriture de ce processus de transformation est une tâche ardue qui demande beaucoup de temps et beaucoup de connaissance du domaine, connaissance de la sémantique du langage de transformation et expertise approfondie de manière à ce que le résultat des auteurs supprimés en raison d'une longueur excessive doit répondre à une certaine qualité de service (par exemple : performance, stockage, convivialité, sécurité). Il convient de noter que la transformation du modèle est dirigée soit par des règles de transformation qui sont appliquées au modèle source pour obtenir un modèle cible, soit appliquer par un transformation de modèles par l'exemple (MTBE) qui permet de définir des transformations à l'aide d'exemples représentés en syntaxe concrète de langages de modélisation pour définir correspondances entre des éléments de modèle correspondant sémantiquement, Ce dernier mode de transformation a été initialement proposé par « Varr » , qui est basé sur d'autres approches par exemple la programmation par exemple ou la requête par exemple, vise à apprendre automatiquement les transformations convoitées à partir d'un ensemble de sources et paires de modèles cibles. Le processus de transformation doit refléter certains aspects du comportement humain, à savoir la collaboration et les différentes vues de conception pour collecter tous les besoins, résoudre les conflits de propositions et satisfaire les exigences des utilisateurs. Au cours des deux dernières décennies, un grand nombre d'efforts de recherche en cours ont été investi dans la transformation du modèle, et plusieurs approches, normes, langages et des outils liés au MTBE ont été proposés , alors que la plupart d'entre eux sont basé sur un référentiel d'exemples unique où chaque exemple est une vue d'expert de la conception. Cependant, nous nous pensons que l'idée de proposer des alternatives de conception de différents experts par exemple : la sélection peut améliorer la qualité du modèle cible et couvrir toutes les possibilités besoins de conception pour les modèles cibles. Cette idée a été principalement initiée dans les entreprises étendues au différents experts collaborent (intégrer et exploiter des exigences basées sur la sémantique

dans un contexte de partage de conception.) produire un business model dans un contexte de partage de conception .

Deuxième partie

Background

Chapitre

2

Les agent BDI (Belifs - Désire –Intentions)

« Research is to see what everybody else has seen, and to think what nobody else has thought. »
— Albert Szent-Gyorgyi

Sommaire

2.1 Introduction

L'une des techniques de modélisation les plus prometteuses pour étudier les systèmes est la modélisation à base d'agents qui permet de fournir des informations intéressantes sur la dynamique du système étudié. Malgré l'émergence de différentes plates-formes de modélisation facilitant le travail des modélisateurs, le problème de la définition des agents est toujours posé. Parce qu'en effet, définir des agents cognitifs capables de prendre des décisions complexes et d'interagir avec leur environnement biophysique et avec d'autres agents est une tâche difficile. Pour résoudre le problème de la définition et de l'implémentation d'agents de nombreuses architectures multi-agents ont été proposées. l'une de ces architectures qui peut être très utiles pour la définition d'agents cognitifs complexes est l'architecture BDI (Belief-Desire-Intention).

2.2 Définitions d'agent

Tout d'abord la notion d'agent est vaste ; Il n'existe pas de définition universelle de ce que peut être un agent. La définition de Russel et Norvig reste très générale.

Définition : "Un agent est tout ce qui peut être compris comme percevant son environnement à travers des senseurs et comme agissant sur cet environnement par l'intermédiaire d'effecteurs".

Un agent est donc défini comme une entité située dans un environnement dotée de capteurs et d'actionneurs, observant l'environnement et cherchant à modifier son évolution. De manière plus précise le fonctionnement général d'un agent est décrit par une boucle fermée appelée boucle sensori-motrice (ou perception-action).

L'exécution de cette boucle peut se décomposer en plusieurs étapes :
- Initialement, l'agent se trouve dans une certaine configuration interne.
- Il perçoit une partie de l'environnement dans lequel il est plongé grâce à ses capteurs.
- Il choisit une action à entreprendre en fonction de sa configuration interne et de ses perceptions. Ce choix sera considéré comme le résultat d'une fonction de prise de décision.
- Il effectue cette action dans le but de modifier l'état de l'environnement ou/et sa configuration interne [19].

Jennings Sycara et *Wooldridge* proposent une autre définition de la notion d'agent qui reste néanmoins proche de celle de Russel et norvig.

Définition 1

> *Un agent est un système informatique situé dans un environnement, qui est capable d'action autonome et flexible dans le but de répondre à ses objectifs de conception.*

Cette définition est fondée sur les trois mots clefs suivants :
- **Situé :** signifie que l'agent reçoit des données sensorielles directement à partir de l'environnement et peut effecteur des actions qui visent à le modifier.
- **Autonome :** signifie que l'agent peut agir de lui-même sans intervention directe extérieure.
- **flexible :** est lié à la notion d'objectif et d'intelligence. Un agent est flexible s'il possède les trois caractéristiques suivantes :
  - il répond à temps : son temps de réponse est court par rapport au temps d'évolution de l'environnement.
  - il est pro-actif : il n'agit pas uniquement en réponse à l'environnement mais peut prendre l'initiative lorsque cela est approprié pour atteindre son objectif.
  - il est social : lorsqu'il est mis en présence d'autres agents, il est capable d'interagir pour résoudre ses problèmes et d'aider les autres agents.

La notion de capacité sociale sera abordée de manière plus détaillée lorsque l'on traitera les systèmes multi-agents [7]

Définition 2

> *Une définition d'un agent donné par J Ferber est : "On appelle agent une entité physique ou virtuelle »:*
> - ***a.****qui est capable d'agir dans un environnement.*
> - ***b.****qui peut communiquer directement avec d'autres agents.*

- ***c.****qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie qu'elle cherche à optimiser).*
- ***d.****qui possède des ressources propres.*
- ***e.****qui est capable de percevoir (mais de manière limitée) son environnement.*
- ***f.****qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune).*
- ***g.****qui possède des compétences et offre des services.*
- ***h.****qui peut éventuellement se reproduire.*
- ***I.*** *Dont le comportement tend à satisfaire ses objectifs en tenant compte des ressources et des compétences dont elle dispose et en fonction de sa perception de ses représentations et des communications qu'elle reçoit [9].*

Un agent évolue dans un environnement, il doit être en mesure de recevoir des informations de cet environnement par des récepteurs et d'agir sur ce même environnement par des effecteurs suivant un comportement décidé selon :

- le raisonnement de l'agent.
- L'agent est caractérisé par son architecture et son comportement.
- L'architecture reste liée au point de vue du concepteur à la manière d'assembler les différentes parties de l'agent afin que ce dernier puisse accomplir ce qui est attendu de lui.

2.3 Catégories d'agent

Selon les architectures et les capacités nous distinguons deux grandes familles d'agents « les agents réactifs et les agents cognitifs ».

2.3.1 Agent réactif

Comme son nom l'indique, un agent réactif ne fait que réagir aux changements qui surviennent dans l'environnement. Autrement dit un tel agent ne fait ni délibération ni planification, il se contente simplement d'acquérir des perceptions et de réagir à celles-ci (Figure 2.1). Étant donné qu'il n'y a pratiquement pas de raisonnement ces agents peuvent agir et réagir très rapidement.

Ferber définit ce type d'agent comme suit : « Les agents réactifs définis par le fait même qu'ils n'ont pas de représentation de leur environnement et des autres agents sont incapables de prévoir ce qui va se passer et donc d'anticiper en planifiant les actions à accomplir » (voir Figure 2.1).

L'agent réactif présente les caractéristiques suivantes :

- Pas de mémoire.
- Pas de représentation explicite.
- Prise de décision se basant sur le fait du Stimulus/Réponse.
- Simple à mettre en œuvre [8].

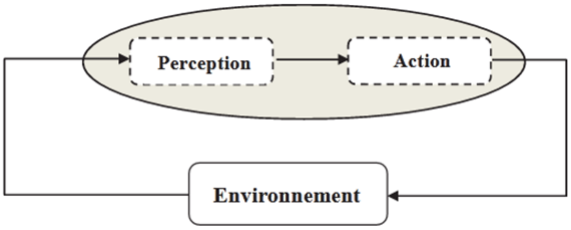


FIGURE 2.1 – Agent réactif.

2.3.2 Agent cognitif

L'agent cognitif est un agent qui dispose d'une base de connaissances comprenant l'ensemble des informations et de savoir-faire nécessaires à la réalisation de sa tâche et la gestion des interactions avec les autres agents et avec son environnement [38].

Ainsi, ces agents possèdent une représentation explicite de leur environnement, des autres agents et d'eux-mêmes. Ils sont aussi dotés de capacités de raisonnement et de planification ainsi que de communication. Le travail le plus représentatif de cette famille d'agent porte sur le modèle BDI (Belifs – Desirs – Intentions).

2.3.3 Agent hybride

Il est possible de combiner ces deux architectures pour obtenir une troisième architecture qui se base sur des agents cognitifs exhibant des capacités de réactions aux événements, on parlera alors d'agents hybrides. L'agent hybride est conçu pour combiner les capacités réactives à des capacités cognitives ce qui leur permet d'adapter son comportement en temps réel à l'évolution de l'environnement. Dans ce modèle, un agent est composé d'une architecture multicouche qui se base sur la hiérarchie de niveaux [11].

2.4 Agent BDI (Belief-Desire-Intention)

Cette architecture est basée sur les notions d'attitudes mentales que sont les Croyance (Belief), le Désir (Desir) et l'Intention (Intention).

Un agent BDI est composé de :

1. Un ensemble de croyances courantes : représentent les informations que l'agent a maintenant sur son environnement.
2. Une fonction de révision de croyances : permet de mettre à jour les croyances de l'agent en fonction de ses croyances précédentes et de l'état actuel de l'environnement.
3. Une fonction de génération des options : elle détermine les options disponibles actuellement à l'agent (ses désires, ou encore les choix des actions à entreprendre).
4. Un ensemble d'options courantes : qui sont les désires ou les alternatives possibles pour l'agent.

5. Une fonction de filtre : c'est le processus de délibération de l'agent qui en fonction de croyances, désires, et intentions actuels de l'agent génère les nouveaux intentions.
6. Un ensemble d'intentions courantes : C'est le centre d'attention actuel de l'agent c'est les objectifs qu'il essaye d'atteindre.
7. Une fonction de sélection des actions : cette fonction détermine les actions à effectuer en se basant sur les intentions actuelles de l'agent.

2.4.1 Les avantages des agents BDI

Les avantages des agents BDI :

1. Basé sur les perceptions des modifications dans l'environnement.
2. Plus performants dans la réalisation de leurs taches.
3. L'architecture BDI basé sur un courant philosophique.
4. L'architecture BDI peut être considérer comme la partie explicite et symbolique des architectures cognitives.
5. Se concentrent en effet sur le niveau conscient de l'esprit (manipuler les concepts et le communiquer aux autre).

2.4.2 Architecture BDI

Les composantes BDI :

Le B = Belief = Croyance

Les croyances d'un agent sont les informations que l'agent possède sur l'environnement et sur d'autres agents qui existent dans le même environnement. Les croyances peuvent être incorrectes, incomplètes ou incertaines. Les croyances peuvent changer au fur et à mesure que l'agent par sa capacité de perception ou par l'interaction avec d'autres agents.

Le D = Desire = Désir

Les désirs d'un agent représentent les états de l'environnement, et parfois de lui-même, que l'agent aimerait voir réalisés.

Le I = Intention = Intention Les intentions d'un agent sont les désirs que l'agent a décidé d'accomplir ou les actions qu'il a décidé de faire pour accomplir ses désirs. Même si tous les désirs d'un agent sont consistants, l'agent peut ne pas être capable d'accomplir tous ses désirs à la fois.

C'est cette idée même qui est au cœur de la théorie BDI de l'action rationnelle proposée pour la première fois par Michael Bratman. C'est une théorie du raisonnement pratique qui essaie de surprendre comment les gens raisonnent dans la vie de tous les jours, en décidant, à chaque moment, ce qu'ils ont à faire. En développant sa théorie, Bratman montre que les intentions jouent un rôle fondamental dans le raisonnement pratique, car elles limitent les choix possibles qu'un humain (ou un agent artificiel) peut faire à un certain moment.

Une architecture BDI est alors un bon candidat pour modéliser le comportement d'un agent intelligent car :

— elle s'appuie sur une théorie connue et appréciée de l'action rationnelle des humains ;
— la théorie a été formalisée dans une logique symbolique formelle, rigoureuse ;
— elle a été implémentée et utilisée avec succès dans beaucoup d'applications.

Ces deux systèmes sont les plus connus, mais bien d'autres systèmes multi-agents et applications sont réalisés dans une conception BDI.

La Figure 2.2 suivante présente les composantes principales d'une architecture BDI.

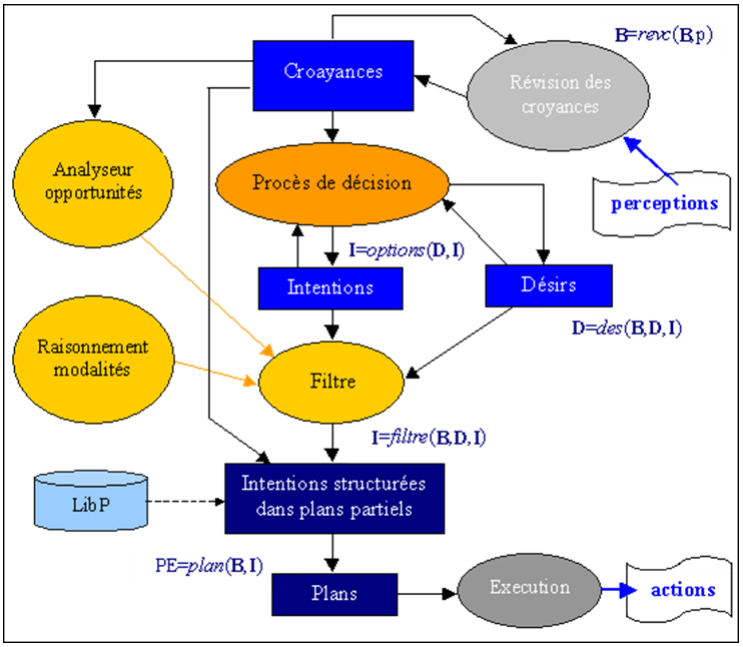


FIGURE 2.2 – : les composantes principales d'une architecture BDI.

L'agent a une représentation explicite de ses croyances, désirs et intentions. On dénote par B l'ensemble des croyances de l'agent, par D l'ensemble de ses désirs, et par I l'ensemble de ses intentions [28].

Les ensembles B, D et I peuvent être représentés au moyen de divers modèles de représentation de connaissances, par exemple en utilisant la logique des prédicats du premier ordre, une logique d'ordre supérieur, le modèle des règles de production ou bien comme de simples structures de données. L'agent doit produire des plans notamment une séquence d'actions qu'il va exécuter pour résoudre le problème.

La représentation des actions la plus commune consiste à représenter les effets de ces actions sur l'environnement.

Le résultat d'une action sera celui envisagé si l'environnement est déterministe.

Les plans conçus par l'agent sont toujours des structures de connaissances ou structures de données. LibP est une bibliothèque de plans. Cette composante peut être présente ou non dans l'architecture. Si elle existe, le processus de planification de l'agent est simplifié car il peut retrouver des plans adéquats à une certaine situation en parcourant cette bibliothèque de plans.

Par la suite, nous considèrerons les fonctions suivantes :
— **revc :** B x P ->B est la fonction de révision des croyances de l'agent lorsqu'il reçoit de nouvelles

perceptions sur l'environnement, où P représente l'ensemble des perceptions de l'agent ; elle est réalisée par la composante Révision des croyances ;

- **options :** D x I->I est la fonction qui représente le processus de décision de l'agent prenant en compte ses désirs et ses intentions courantes ; cette fonction est réalisée par la composante Processus de décision.
- des : B x D x I->D est la fonction qui peut changer les désirs d'un agent si ses croyances ou intentions changent, pour maintenir la consistance des désirs de l'agent (on suppose dans notre modèle que l'agent a toujours des désirs consistants) ; cette fonction est également réalisée par la composante Processus de décision.
- **filtre :** B x D x I->I est la fonction la plus importante car elle décide des intentions à poursuivre ; elle est réalisée par la composante Filtre.

La composante Filtre est la partie de l'architecture qui a la responsabilité de bâtir des plans partiels pour réaliser les intentions de l'agent, tout en tenant compte des nouvelles opportunités [20].

En conséquence de ce qu'il perçoit de son environnement et de sa révision des croyances, l'agent peut détecter des nouvelles opportunités qui favorisent la réalisation de ses intentions ou qui peuvent même empêch cette réalisation. Cette analyse est effectuée par la composante Analyseur opportunités. Ces nouvelles opportunités sont communiquées au Filtre.

Le Filtre construit des plans partiels pour aboutir aux intentions de l'agent avec l'aide de la composante Raisonnement modalités ; cette dernière a la responsabilité d'effectuer le raisonnement orienté action et la modalité de réalisation des plans exécutables, PE étant l'ensemble de ces plans elle peut utiliser, par exemple : une bibliothèque de plans représentée par le module LibP dans la (figure 2.2).

Un plan est une séquence d'actions à exécuter dans le temps. Un plan partiel est un plan dans lequel tous les détails de planication n'ont pas été spéciés. Par exemple : on s'est contenté d'un ordre partiel des actions dans le temps, ou certaines actions ne sont pas entièrement détaillées. La raison pour commencer par bâtir des plans partiels est que l'agent peut parfois être obligé de changer ses plans en fonction de nouvelles perceptions recueillies sur l'environnement, ainsi qu'il a été dit dans la section précédente.

L'approche des plans partiels peut apporter une solution à ce problème.

Une fois qu'on dispose d'un plan exécutable, le module Exécution va exécuter l'une après l'autre, les actions de ce plan dans l'environnement.

2.4.3 Le contrôle BDI

Nous avons jusqu'ici détaillé les composantes et les fonctions d'une architecture BDI. Cette architecture est assez proche de l'architecture du système PRS. Pour apporter plus de précision, nous allons décrire le processus de contrôle d'un agent BDI par l'algorithme suivant (algorithme conçu par Michael Wooldrige).

Soient B_0, D_0 et I_0 les croyances, désirs et intentions initiales de l'agent.

Algorithme de contrôle d'agent BDI

1. B =B0
2. D =D0
3. I = I0
4. répéter

(4.1) obtenir nouvelles perceptions p

(4.2) B = revc(B, p)

(4.3) I = options(D, I)

(4.4) D = des(B, D ,I)

(4.5) I = ltre(B, D, I)

(4.6) PE = plan(B, I)

(4.7) exécuter(PE) jusqu'à ce que l'agent soit arrêté fin

Si une intention a été choisie par la fonction filtre, on dit que l'agent s'est obligé à réaliser cette intention.

L'obligation d'un agent envers une intention, en anglais appelé "commitment", a une grande importance dans le modèle BDI d'un agent situé dans un environnement non déterministe.

Combien de temps va l'agent est-il considéré comme lié par une telle obligation ?

Il ne serait pas très rationnel de la part de l'agent d'abandonner une intention immédiatement après avoir décidé de la réaliser. On peut donc supposer que l'agent va maintenir, pour un certain temps, ses intentions choisies au pas (4.5).

Le cas le plus simple est, évidemment, de supposer que l'agent reste lié à une intention jusqu'à ce qu'elle soit réalisée. Mais qu'advient-il si la réalisation de cette intention devient impossible à cause d'un changement dans les conditions de l'environnement ?

Et si, au moment où l'agent en arrive à exécuter son intention (pas 4.7), les conditions de l'environnement ont changé par rapport aux perceptions effectuées au pas (4.1) et qu'en conséquence, d'autres intentions deviennent plus opportunes à réaliser ?

Revenons maintenant à la question qu'on a posée plus haut : " Combien de temps l'agent reste-t-il obligé par une intention ?"

La réponse à cette question est donnée par la stratégie d'obligation envers les intentions choisies par le concepteur pour les agents BDI. On peut identifier trois principales stratégies d'obligation :

- **Obligation aveugle** (ou obligation fanatique). Un agent suivant cette stratégie va maintenir ses intentions jusqu'à ce qu'elles soient réalisées, plus précisément jusqu'à ce qu'il croie qu'elle sont réalisées. Tenant compte de ce qu'on a déjà dit, cette stratégie n'est pas la meilleure si l'environnement change entre le moment où l'agent a choisi (filtré) ses intentions, et le moment où ces intentions doivent être accomplies.
- **Obligation limitée**. Cette stratégie dit que l'agent va maintenir ses intentions, ou bien jusqu'à ce qu'elles soient réalisées, ou bien jusqu'à ce qu'il croie qu'elles ne sont plus réalisables.
- **Obligation ouverte.** Un agent ayant une stratégie d'obligation ouverte maintient ses intentions

tant que ces intentions sont aussi ses désirs. Cela implique aussi que, une fois que l'agent a conclu que ses intentions ne sont plus réalisables, il ne les considère plus parmi ses désirs.

L'algorithme de contrôle d'un agent BDI décrit plus haut correspond à un agent avec une stratégie d'obligation aveugle envers ses intentions. Dans les deux autres cas, l'agent doit vérifier ses intentions après l'exécution de chaque action du plan qu'il a conçu au pas 4.6. On peut modifier l'algorithme de contrôle pour prendre en compte les deux autres stratégies ; pour cela, on définit première(PE) la première action d'un plan PE et reste(PE) le reste des actions du PE après l'exécution celle-ci.

Algorithme de contrôle d'agent BDI avec obligation limitée :

1. B = B0
2. D = D0
3. I = I0
4. répéter
   - 4.1 obtenir nouvelles perception p
   - 4.2 B = revc(B, p)
   - 4.3 I = options (D, I)
   - 4.4 D = des (B, D, I)
   - 4.5 I = filtre (B, D, I)
   - 4.6 PE = plan (B, I)
   - 4.7 Tanque (PE<> et non accompli(I,B)et possible(I,B))répéter
     - x = première(PE) ; exécuter(x) ; PE = reste(PE)
     - obtenir nouvelles perceptions p
     - B = revc(B, p)
     - fin tant que jusqu'à ce que l'agent soit arrêté fin.

Dans cet algorithme possible(I, B) est la fonction qui vérifie si après avoir obtenu la perception p sur l'environnement et révisé ses croyances , les intentions choisies sont encore réalisables non-accompli (I, B) est la fonction qui vérifie si par hasard les intentions de l'agent sont déjà accomplies avant même que l'exécution de son plan soit terminée.

Algorithme de contrôle d'agent BDI avec obligation ouverte

- 1. B = B0
- 2. D = D0
- 3. I = I0
- 4. Répéter
  - 4.1 obtenir nouvelles perception p
  - 4.2 B = revc(B, p)
  - 4.3 I = options (D, I)
  - 4.4 D = des (B, D, I)
  - 4.5 I = filtre (B,D,I)
  - 4.6 PE =plan(B, I)
  - 4.7 Tan que (PE<>et nonccompli(I,B) et possible (I,B))répéter
    - x = première(PE) ; exécuter(x) ; PE = reste(PE)
    - obtenir nouvelles perceptions p
      - B = revc(B, p)
      - D = des(B, D ,I)
      - I = filtre(B, D, I)
      - PE = plan(B, I)
      - fin tant que jusqu'à ce que l'agent soit arrêté fin

Dans cet algorithme, après avoir reçu des nouvelles perceptions sur l'environnement et révisé ses croyances, l'agent considère aussi ses désirs et en plus considère un possible changement dans l'ensemble de ses intentions par la fonction filtre. Dans ce cas l'agent s'engage dans un processus de re-planification car le résultat de la fonction filtre est un plan partiel.

Les stratégies d'obligation limitée et ouverte envers les intentions sont, d'un certain point de vue, meilleures que la stratégie d'obligation aveugle, car l'agent vérifie si ses intentions sont encore faisables ou appropriées. Par contre, et surtout dans la stratégie de l'obligation aveugle, il y a danger que l'agent change perpétuellement ses intentions en fonction des nouvelles conditions de l'environnement et arrive dans la situation de ne jamais voir ses désirs aboutir. Par exemple, si notre agent de voyage va tout le temps voir s'il y a de nouvelles offres des compagnies aériennes et modifie sans arrêt la réservation, il se peut qu'il n'arrive jamais à nous offrir les billets d'avion. Il faut alors trouver le juste rapport entre le temps passé par l'agent pour accomplir ses intentions et le temps pendant lequel l'agent vérifie si ces intentions sont toujours possibles ou désirables.

Les architectures BDI sont l'exemple le plus représentatif d'architecture permettant de bâtir un agent cognitif [39].

2.5 Conclusion

La simulation à base d'agents est maintenant largement utilisée pour étudier les systèmes complexes. Cependant, le problème de la définition des agents est toujours posé. Définir des agents complexes capables d'agir de manière réaliste est une tâche difficile. une architecture agent basée sur le paradigme BDI permet de répondre aux difficultés précitées. Cette architecture semble avoir les qualités nécessaires pour formaliser le comportement de tels agents. L'architecture BDI s'est d'ailleurs avérée comme l'une des architectures favorites dans la communauté des agents intelligents et des systèmes multi-agents.

Chapitre

3

Transformation de modèles par l'exemple

« Research is to see what everybody else has seen, and to think what nobody else has thought. »
— Albert Szent-Gyorgyi

Sommaire

3.1 Introduction

L'ingénierie dirigée par les modèles (Model Driven Engineering,IDM) est une approche qui s'applique au génie logiciel en procurant un rôle prépondérant à la notion de modèles. Elle implique ainsi un gain remarquable en productivité en maximisant la compatibilité entre les systèmes et en simplifiant le processus de conception tout en facilitant la communication entre les équipes chargées du développement [6].

L'IDM est donc une forme d'ingénierie générative par laquelle tout ou partie d'une application informatique est générée à partir de modèles. Dans cette nouvelle perspective, les modèles occupent une place de premier plan parmi les artéfacts de développement des systèmes et doivent en contre partie être suffisamment précis et riches afin de pouvoir être interprétés ou transformés par des machines. Le processus de développement des systèmes peut alors être vu comme une séquence de transformations

de modèles partiellement ordonnée, chaque transformation prenant un ou des modèles en entrée et produisant un ou des modèles en sortie ,jusqu'à l'obtention d'artéfacts exécutables.

L'IDM a pour principal objectif de relever un certain nombre de défis du génie logiciel (qualité, productivité, sûreté, séparation des préoccupations, coût, etc.) .

Le développement dirigé par les modèles (en anglais : model-driven software development) est l'application de l'ingénierie dirigée par les modèles au développement logiciel.

L'architecture dirigée par les modèles (MDA) désigne une initiative de l'Object Management Group pour mettre au point des standards non-propriétaires pour faciliter le développement dirigé par les modèles . [8][9]

3.2 Architecture dirigé par le modèle (Model Driven Architecture , MDA)

L'approche MDA a été présentée publiquement par l'OMG (Object Management Group) [1] (L'OMG est un consortium regroupant des industriels, des fournisseurs d'outils et des académics dont l'objectif principal est d'établir des standards pour résoudre les problèmes d'interopérabilité entre les systèmes d'information. Ces standards sur lesquels repose l'IDM sont centrés sur les notions de métamodèles et de métamétamodèles.) [23] en Novembre 2000, elle constitue une déclinaison du IDM en réunissant l'ensemble des standards proposés par l'OMG [1].

MDA est une approche qui se focalise sur l'utilisation des modèles pour la réalisation des systèmes informatiques, car elle part du principe que les modèles de conception sont pérennes dans le temps alors qu'une technologie de développement change constamment et peut rapidement devenir obsolète, les modèles deviennent alors des éléments productifs, contrairement à l'approche centrée sur le code, ou la conception du système ne représentait qu'une étape passagère pour arriver à la phase de l'implémentation, ce qui limitaient le rôle des modèles à un rôle contemplatif et purement descriptif.

L'approche MDA se base sur les trois notions suivantes [9] :

3.2.1 Modèle

Un modèle est une abstraction de la réalité [26], son objectif est de structurer les informations et activités d'une organisation : données, traitements, et flux d'informations entre entités pour permettre une étude plus simple dans un contexte maîtrisé autre que le contexte réel .

Au niveau du IDM les modèles manipulés sont de natures diverses : modèles d'objets métiers, de processus, de service, de plate-forme, de transformation et de tests. On peut distinguer trois classes de modèles :

- **a-** modèles indépendants de l'informatisation appelés (Computation Independent Model : CIM) : Un CIM modélise les exigences d'un système, son but étant d'aider à la compréhension du problème mais aussi de fixer un vocabulaire commun pour un domaine . .
- **b-** modèles indépendants des plates-formes appelés (Platform Independent Model : PIM) : Le PIM, connu aussi sous le nom de modèle d'analyse et de conception, est un modèle abstrait

indépendant de toute plate-forme d'exécution. Le PIM a pour but de décrire le savoir-faire ou la connaissance métier d'une organisation. Ayant isolé le savoir-faire métier dans des PIMs, on a besoin soit de transformer ces modèles en d'autres PIMs pour des besoins d'interopérabilité, soit de produire des modèles PSM ciblant une plate-forme d'exécution spécifique en se basant sur les PDMs ("Platform Description Model" il définit les différentes fonctionnalités de la plate-forme et précise comment les utiliser) pour améliorer la portabilité et augmenter la productivité [22].

— **c-** modèles spécifiques à une plate-forme appelés (Platform Specific Model : PSM) :un PSM définit les différentes fonctionnalités de la plate-forme et précise comment les utiliser [21].

3.2.2 Méta-modèle

Un métamodèle est un modèle qui décrit la structure de modèles. En particulier, il permet la construction de langages de modélisation, la création de relations entre les modèles et la définition de règles de modélisation. On dit que le métamodèle représente le modèle, tandis que le modèle instancie le métamodèle [37].

3.2.3 La transformation de modéle

Une transformation est la génération automatique d'un modèle cible à partir d'un modèle source [23].

L'initiative MDA de l'OMG a donné lieu à une standardisation des approches pour la modélisation sous la forme d'une structure en 4 niveaux de modélisation (dit encore Pile de modélisation) [24].

3.2.4 Standards de l'OMG

Nous présentons modèles standards de l'OMG utilise pour les techniques générales de métamodlisation :

MOF (Meta-Object Facility), le standard pour établir de nouveaux langages de modélisation ;

UML (Unified Modeling Language), le standard de modélisation des systèmes ;

OCL (Object Constraint Language), le standard pour exprimer la sémantique statique des modèles et des métamodèles ;

XMI (XML Metadata Interchange) qui permet de représenter les modèles sous forme de documents XML pour des besoins d'interopérabilité ;

DI (Diagram Interchange) qui permet la représentation au format XML des parties graphiques d'un modèle [33].

3.2.4.1 Meta-Object Facility (MOF)

MOF (OMG-MOF 2.0, 2006) se situe au sommet dans l'architecture à quatre niveaux de l'OMG (voir figure 2 ci-après). MOF est un méta-formalisme c'est-à-dire un formalisme pour établir des langages de modélisation permettant eux-mêmes d'exprimer des modèles. Dans sa version 2.0 le

métamodèle MOF est constitué de deux parties : EMOF (Essential MOF), pour l'élaboration des métamodèles sans association, et CMOF (Complete MOF) pour les métamodèles avec associations. Il faut souligner que MOF s'auto-décrit pour pouvoir limiter l'architecture pyramidale de l'OMG à quatre niveaux (Figure 3.1 ci-dessous). Dans cette architecture, le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les métamodèles permettant la définition de ces modèles (UML, SPEM, etc.) constituent le niveau M2. Enfin, le métamétamodèle (MOF), unique et métacirculaire, est représenté au sommet de la pyramide (niveau M3) [36].

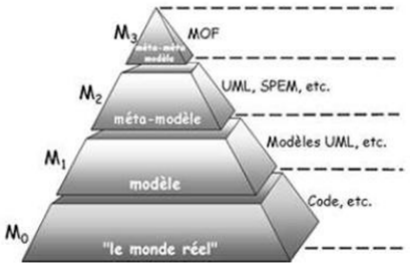


Figure 3.1 – Pyramide de modélisation de l'OMG .

3.2.4.2 Unified Modeling Language (UML)

UML est le standard de facto en matière de modélisation des systèmes logiciels, notamment dans les domaines d'application tels que l'aéronautique, le spatial ou l'automobile. Dans sa deuxième version (OMG-UML 2.2, 2009), le métamodèle UML est composé de deux paquetages : UML 2.2 Superstructure qui devient le standard en matière de modélisation orientée objet, et UML 2.2 Infrastructure qui décrit le noyau d'UML commun avec le MOF.

UML Infrastructure a été conçu de façon modulaire pour pouvoir être réutilisé par MOF et UML Superstructure. Il est important de noter qu'il est sans niveau fixe ; il peut appartenir au niveau M3 s'il est intégré dans MOF ou au niveau M2 s'il est intégré dans UML Superstructure [30].

3.2.4.3 Object Constraint Language (OCL)

Le but du langage OCL (OMG-OCL, 2005) est de permettre l'ajout de contraintes pour exprimer la sémantique statique des modèles et métamodèles. Un métamodèle n'a pas toujours l'expressivité suffisante pour décrire toutes les relations entre les méta-éléments qu'il contient. Dans le contexte des normes de l'OMG, OCL est utilisé pour décrire des contraintes non capturées dans le métamodèle, sous forme d'invariants. L'ajout de ces contraintes est fondamental pour obtenir des métamodèles clairement définis et donc outillables. En outre, OCL définit des pré et post conditions sur les opérations pour que le système modélisé reste dans un état cohérent quand on exécute ces opérations. Les constructions d'OCL ne permettent pas de créer, de détruire ou modifier les objets d'un modèle : la vérification des contraintes se fait sans effet de bord. En plus de la définition de contraintes, OCL peut être utilisé pour spécifier de manière déclarative le

comportement de propriétés dérivées et d'opérations, sans toutefois pouvoir exprimer impérativement des modifications de modèles. OCL fournit donc une bonne solution pour exprimer des contraintes sur les modèles mais pas pour décrire le comportement ou la sémantique des modèles. Cette limitation d'OCL a amené l'OMG à définir le Standard AS (Action Semantics) qui permet de modifier l'état d'un modèle. Cependant, il n'y a pas à ce jour de syntaxe concrète standardisée pour AS (Blanc, 2005). Depuis la deuxième version d'UML, le standard AS est intégré dans UML Superstructure. Cette limitation d'OCL ne remet pas en cause ses vertus, d'autant plus qu'il peut être appliqué sur tout type de modèle (MOF, Ecore, UML ...) et est utilisé dans plusieurs langages ou normes de transformation (QVT, ATL, Kermeta ...) [3].

3.2.4.4 XML Metadata Interchange (XMI)

Les modèles étant des entités abstraites au niveau conceptuel, l'OMG a décidé de standardiser XMI (Blanc, 2005), (OMG-XMI, 2007) qui offre une représentation concrète des modèles sous forme de documents XML. Cette représentation concrète des modèles se fait par des mécanismes appelés sérialisation et génération. La génération consiste à transformer un métamodèle en un DTD (Document Type Definition) alors que la sérialisation permet de représenter les modèles sous forme de document XML (voir Figure 3.2 ci-dessous) [10].

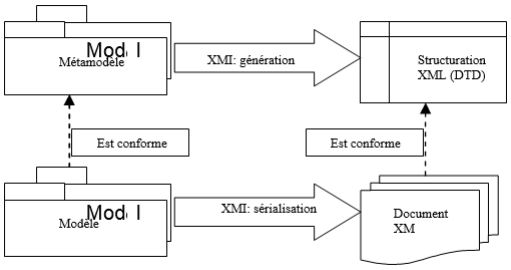


FIGURE 3.2 – XMI et la structuration de balises XML .

3.2.4.5 Diagram Interchange (DI)

Dans la sérialisation des modèles en documents XML, nous pouvons remarquer que la partie graphique des modèles n'est pas représentée. Cela est dû au fait que XMI ne permet de représenter sous forme de documents XML que les informations dont la structure est définie dans un métamodèle (voir figure 3 ci-dessus). Pour faire face à cette difficulté, l'OMG a décidé de définir un standard spécifique sous le nom de DI (Diagram Interchange) (OMG-DI, 2005). L'idée de ce standard est de définir un nouveau métamodèle DI lié à UML et représentant sous forme de métaclasses les idiomes graphiques nécessaires et suffisants à la représentation des parties graphiques des modèles UML. En plus de cela, DI définit une transformation de modèles permettant de générer automatiquement des documents SVG (W3C-SVG, 2003) à partir des documents XML, ceci dans le but de rendre visibles les parties graphiques des modèles UML dans n'importe quel outil supportant le format SVG [14].

3.2.4.6 Transformation de modèle

La définition de transformations a pour objectif de rendre les modèles opérationnels dans une approche IDM, ce qui augmente considérablement la productivité des applications.

La transformation de modèles est une opération très importante dans toute approche orientée modèle. En effet, les transformations assurent les opérations de passage d'un ou plusieurs modèles d'un niveau d'abstraction donné vers un ou plusieurs autres modèles du même niveau(transformation horizontale) ou d'un niveau différent (transformation verticale).

On peut citer comme exemple de transformation verticale, la transformation PIM vers PSM dans l'approche MDA. Le modèle transformé est appelé modèle source et le modèle résultant de la transformation est appelé modèle cible [32]. La transformation de modèles dans un cadre MDA est réalisée grâce à un moteur de transformation qui applique un ensemble de règles au PIM fourni en entrée pour produire le PSM en sortie .

Il faut noter que l transformations de modèles et caractérisée par certaines propriétés dont :

- **La Réversibilité :** une transformation de modèle est inversible s'il existe une transformation permettant d'aller dans le sens inverse.
- **La Réutilisabilité :** consiste en la capacité d'étendre les transformations existantes afin de produire de nouvelles transformations de modèles. L'aspect de la modularisation des règles de transformations joue un rôle primordial à ce niveau.
- **L'Ordonnancement des règles :** cet aspect détermine l'ordre de l'exécution des règles de transformation, en effet, ceci implique la représentation des niveaux d'imbrication de celle-ci.
- **La Traçabilité :** précise la façon qui permet la journalisation des étapes d'une transformation de modèles et le suivi des liens permettant la correspondance des éléments du modèle source aux éléments du modèle cible.

La notion de méta-modèle est omniprésente dans ce cas, dans la mesure où chaque modèle (PIM ou PSM) s'appuie sur un méta-modèle qui sert à le décrire. Lorsque les deux modèles s'appuient sur le même méta-modèle, il s'agit alors d'une transformation «endogène», dans le cas contraire, nous parlons d'une transformation « exogène ».

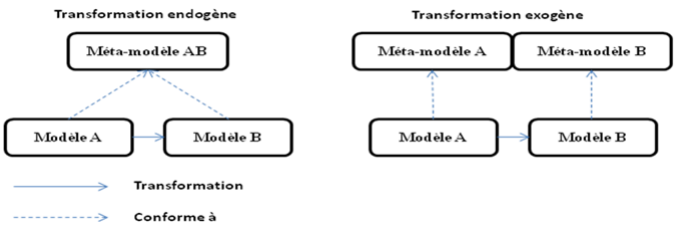


FIGURE 3.3 – Transformation endogène et exogène .

3.2.5 Type de transformation de modèles

Nous comptons deux types de transformations de modèles [13] :

— **Les transformations M2M (Model to Model) :** Il s'agit des transformations de modèles vers modèles, ces transformations concernent l'ensemble des tâches à exécuter afin d'aboutir à un modèle respectant les spécifications technique de l'environnement ciblé à partir d'un autre modèle.
— **Les transformations M2T (Model to Text) :** il s'agit des transformations visant la génération du code ou de la documentation.

3.2.6 Approche de transformation de modèles

Il existe différentes approches pour effectuer une transformation de modèles, offrant chacune une manière bien précise pour l'élaboration des règles de transformations. La Figure 3.4 ci-dessous présente la hiérarchie des approches existantes permettant d'élaborer les transformations de type M2M et M2T [36].

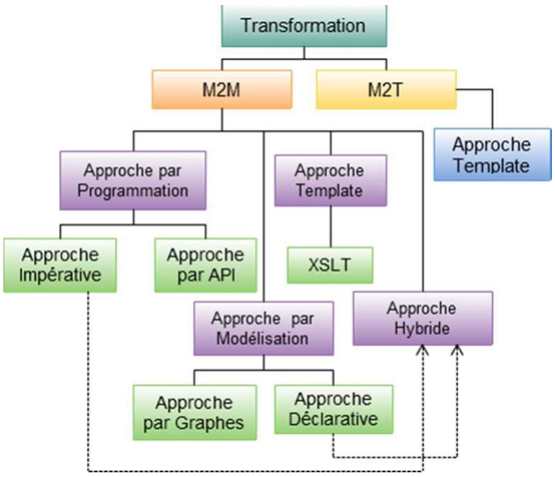


FIGURE 3.4 – Classification des différentes approches de transformations de modèles .

3.2.6.1 L'approche par programmation

Cette approche se base sur l'utilisation des langages de programmation pour décrire les règles de transformation sous forme de programmes informatiques, cette approche est considérée comme une approche de manipulation directe. L'approche par programmation se présente sous deux différentes visions :

— **L'approche impérative :** Similaire à la programmation impérative, cette approche s'appuie sur les fondements de la programmation structurée et définit les transformations sous forme de séquences d'instructions contenues dans des fonctions et des procédures pour changer l'état du modèle à l'exécution. L'intérêt d'une telle approche réside dans le fait qu'elle reste relativement familière aux développeurs, la syntaxe et la sémantique qu'elle emploie est proche du langage OCL (Object Constraint Language) [18].
— **L'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 [24].

3.2.6.2 L'approche par Template

Elle consiste à définir des patrons paramétrés pour les modèles cibles. Les paramètres sont fournis par les valeurs contenus dans les modèles sources pour pouvoir monter ainsi le modèle de sortie. L'implémentation de réference représentant cette approche est l'approche XSLT qui met en œuvre le langage XSLT (eXtensible Stylesheet Language Transformations). Ce langage a été conçu à la base pour la transformation des documents XML (eXtensible Markup Language) vers d'autres formats. Dans un cadre MDA, les modèles sont sérialisés sous format XMI (XML Metadata Interchange) qui est une variante du langage XML, cela explique alors pourquoi une telle approche s'adapte parfaitement aux transformations de modèles dans un tel contexte. Néanmoins, cette approche présente des inconvénients de performances et d'efficacité une fois que les modèles à transformer atteignent un certain niveau de complexité [24][25].

3.2.6.3 L'approche par modélisation

Aussi appelée l'approche de transformation par règles, cette approche consiste à modéliser les transformations eux mêmes afin de les rendre pérennes et productives en leur appliquant l'ingénierie des modèles. Elle peut être concrétisée par l'une des deux approches suivantes :

- **L'approche par graphes :** Il s'agit d'un formalisme mathématique qui applique la théorie des graphes pour la transformation de modèles, en considérant ces derniers comme des graphes. La stratégie de transformation dans cette approche consiste en un remplacement et une mise en correspondance entre le modèle source et cible, qui s'appuie sur une syntaxe de règles de graphes pour prendre un LHSG (Left Hand Side Graph) et le transformer en un RHSG (Right Hand Side Graph). La complexité de cette approche réside dans l'aspect non-déterministe de la stratégie d'application des règles de transformations , ce qui implique que les solutions basées sur ce paradigme sont très peu utilisées dans le concret [5].
- **L'approche déclarative :** Dans cette approche, Une règle fait la correspondance entre un ensemble de concept 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 [25].
- **L'approche hybride :** Cette approche est la plus récente parmi les autres approches de transformation, elle combine l'approche déclarative et impérative, L'approche déclarative est généralement utilisé pour la définition et la sélection des transformations qui peuvent être appliquées, tandis que l'approche impérative est bien adapté à la description de la stratégie de transformation [29].

3.3 Transformation de modèles par l'exemple

L'approche de transformation de modèles par l'exemple a pour objectif d'apprendre des programmes de transformation de modèles à partir d'un ensemble de paires de modèles sources et cibles fournis en guise d'exemple . Un mécanisme d'apprentissage est ensuite utilisé pour dériver des règles de transformations qui peuvent être sujettes à un affinement manuel avant d'être exécutées . Il existe deux axes de recherche portant sur l'apprentissage de transformation de modèles par l'exemple : les approches par l'exemple proprement dites (TMPE) ainsi que celles par démonstration (TMPD) [4] .

3.3.1 Transformation de modèles par l'exemple (TMPE)

Les travaux de TMPE œuvrent 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 cible et d'un modèle source. 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 En 2006, Varró propose une première approche pour l'apprentissage de transformations à partir de paires d'exemples en utilisant un algorithme basé sur les graphes. L'algorithme prend en entrée des paires de modèles inter reliés (accompagnés de traces) et dérive un ensemble de règles de transformation de type 11. Le processus de dérivation est itératif et interactif, à chaque itération, les règles produites sont raffinées par l'utilisateur. La contribution est ensuite automatisée davantage par Balogh e t alen 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 nm). Durant la même période Wimmer propose une approche similaire pour dériver des transformations sous la forme de règles de type 11 exprimées en ATL. La contribution en question est également itérative et incrémentale, elle diffère cependant des deux contributions précédentes sur deux aspects. D'abord, les traces de transformation exploitées sont spécifiées dans une syntaxe concrète plutôt qu'abstraite. Ensuite, le processus de dérivation se fait selon une approche orientée objet [2, 17, 34, 31]. La contribution de Wimmer est également améliorée dans des contributions subséquentes, par Strommer, où un langage pour la définition de correspondances de type nm est présenté, jumelé à un algorithme de raisonnement pour la dérivation de règles (nm également) à partir des correspondances exprimées. Un autre travail qui s'inscrit dans le contexte des transformations de modèles par l'exemple est celui de Garcia-Magarino qui propose un algorithme permettant de générer des règles de transformation de type nm, vraisemblablement dans plusieurs langages de transformation, à partir d'un ensemble de paires de modèles sources et cibles interconnectés (agrémentés par des traces). Kessentini 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 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). Une autre contribution de TMPE qui ne produisait pas de règles de transformation initialement est celle de Dolques. Ce travail se base sur l'analyse relationnelle de concepts (ARC), une variante de l'analyse formelle de concepts, pour classifier les éléments sources et cibles ainsi que les correspondances fournies en entrée. Les patrons identifiés sont organisés dans des treillis partiellement ordonnés et sont ensuite analysés pour filtrer les plus pertinents. L'approche est également étendue par Saada où des règles exécutables sont produites à partir des patrons filtrés. Dans cette dernière contribution, les règles sont exprimées en Jess [12].

Les travaux les plus récents dans le contexte de la TMPE sont ceux de Faunes dans lesquels 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) et produit en sortie des règles exécutables de typenm. L'approche est ensuite améliorée par la contribution de Baki où le programme génétique tente d'apprendre simultanément les règles de transformation ainsi que le contrôle d'exécution qui doit être exercé sur celles-ci 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 définition lors de la construction du modèle cible [27, 36, 35]. .

3.3.2 Transformation de modèles par démonstration (TMPD)

simplifie la mise en œuvre des transformations de modèle en déduisant des modèles de transformation à partir des opérations démontrées d'un utilisateur pour transformer une instance de modèle concrète [15, 16, 1].. Etat de l'art Ce paragraphe a pour but de présenter les travaux QUI ONR réduire les fonctions d'un administrateur de nombreux travaux de recherche se sont intéressés à l'automatisation des TM. L'apprentissage de transformations de modèles par l'exemple (MTBE) constitue, à cet égard une approche prometteuse.

3.4 Modèle de transformation

La transformation d'un modèle source à un modèle cible est motivée par le besoin de stockage, de simulation, de validation, d'interopérabilité. En règle générale, cette transformation d'un modèle à un autre peut se fonder sur des règles de transition bien définies et en l'absence de telles règles, il est nécessaire de s'appuyer sur les exemples.

3.5 Scenario de transformation modèle

Dans l'exemple que nous avons choisi (modèle de classe vers modèle relationnel) est basé sur un certain nombre d'actions pour N objets avec un nombre k de sous-domaines de ces attributs le nombre des solutions du modèle cible est égal à . Le concepteur doit être assisté par un algorithme

Approche	Algorithme	entrée	Sortie	n-m règles?	Con- trole?	Contexte?	état?	Complexe dérivations?
Approches MTBE								
Varro [2006]	ad-hoc heuristic	Exemples & traces	Règles	1-1	Non	Non	Non	Non
Wimmer et al. [2007]	ad-hoc heuristic	Exemples & traces	Règles	1-1	Non	Non	Non	Non
Strommer et al. [2007; 2008]	Pattern matching	Exemples & traces	Règles	n-m	Non	Non	Non	String operator
Kassentini et al. [2008; 2012]	PSO/PSO- SA	Exemples & traces	Target model	–	Non	Non	Non	Non
Balogh and Varro [2009]	ILP	Exemples & traces	Règles	n-m	Non	Non	Non	Non
Garcıa-Magarin~ o et al. [2009]	ad-hoc algorithme	Exemples & traces	Règles	n-m	Non	Non	Non	Non
Kassentini et al. [2010]	PSO-SA	Exemples seulement	Règles	1-m	Non	Non	Non	Non
Deloques et al. [2010]	RCA	Exemples & traces	Paternes	–	Non	Non	Non	Non
Saada et al. [2012a; 2012b]	RCA	Exemples & traces	Règles	1-m	Non	Non	Non	Non
Faunes et al. [2012; 2013]	GP	Exemples seulement	Règles	n-m	Non	Non	Non	Non
Baki et al. [2014]	GP	Exemples & traces	Règles	n-m	Oui	Oui	Non	Non
Approches MTBD								
Brosch et al. [2009a; 2009b]	Pattern matching	User actions	Règles	–	–	Non	Non	Ajouter manuellement
Sun et al. [2009; 2011]	Pattern matching	User actions	Règles	–	–	Non	Non	Demon- strated by the user
Langer et al. [2010]	Pattern Matching	User Actions	Règles	n-m	Oui	Non	Non	Ajouter manuellement

Tableau 3.1 – Travaux connexes .

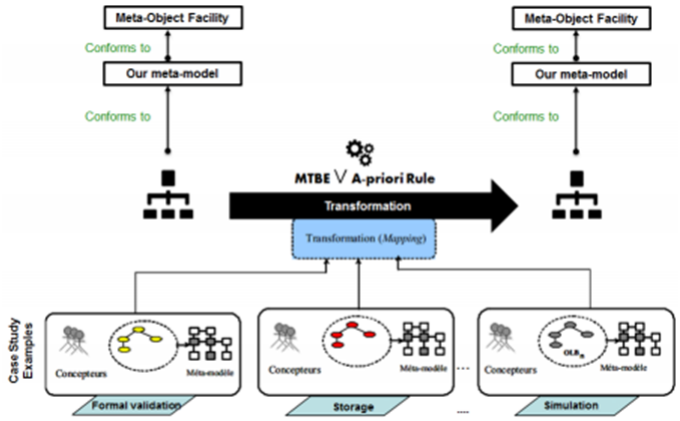


FIGURE 3.5 – présentation du modèle de transformation .

d'optimisation pour explorer la solution spatiale et évalué le module qui quantifie le coût de chacune des solutions par rapport à l'objectif de transformation.

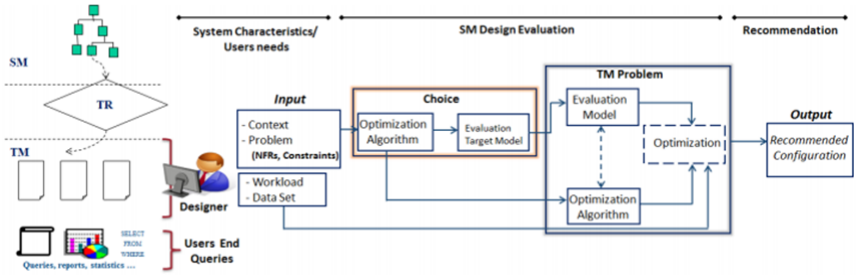


FIGURE 3.6 – scenario de transformation modèle.

3.6 Conclusion

Le développement logiciel dirigé par les modèles a pour principal objectif de concevoir des applications en séparant les préoccupations et en plaçant les notions de modèles, métamodèles et transformations de modèles au centre du processus de développement .cette dernière a fait l'objet de nombreux travaux de recherche et qui s'est récemment focalisé sur l'approche de transformation de modèles par l'exemple qui ne consiste pas à générer le code d'une transformation de modèles à partir d'exemples mais de faire apprendre à un système à transformer des modèles.

Nous devrons savoir que la transformation d'un modèle à un autre est un ensemble d'actions à réaliser telles que (diviser, supprimer, créer...) sur la base de la transformation les actions réalisant une instance d'un modèle source peuvent générer un espace de solutions considérable avec complexité liée aux paramètres des objets SM.

Troisième partie

Notre proposition

Chapitre

4

Notre approche proposée

Sommaire

4.1 Introduction

L'écriture de transformations de modèles demeure cependant une tâche complexe, qui requiert à la fois beaucoup de connaissances et d'efforts. Le nouveau défi est de savoir comment transformer des modèles sans spécifier de règles, détecter les défauts de conception sans les spécifier et tester les transformations sans définir les modèles cibles attendus et spécifier les contraintes. L'idée est d'utiliser le modèle transformation par l'exemple (MTBE). Dans ce travail, nous intégrons entre l'algorithme évolutif et systèmes multi-agents, essayant de tirer le meilleur des deux. Les algorithmes évolutionnaires (AE est un ensemble d'algorithmes destinés à l'optimisation et à l'apprentissage,qui imitent l'évolution naturelle des espèces qui s'adaptent au l des générations à leur environnement) fonctionnent avec une population de structures qui codent des solutions possibles au au problème. Cette population évolue itérativement au moyen d'opérateurs heuristiques inspirés ou motivés par des concepts des systèmes naturels et des principes darwiniens. L'intégration entre les systèmes multi-agents et les algorithmes évolutifs conduit à ce que l'on appelle les systèmes multi-agents évolutifs (EMAS) .Dans de tels systèmes les agents ont aussi la capacité d'évoluer, de se reproduire et de s'adapter à l'environnement, rivaliser pour les ressources, de communiquer avec d'autres agents et de prendre des décisions autonomes. L'EMAS utilise des opérateurs génétiques et le système multi-agents. L'EMAS présente des avantages potentiels dans la résolution de problèmes liés aux systèmes complexes. En effet, il résout le problème par décomposition modulaire et fonctionnelle. Le processus de transformation est une optimisation distribuée des structures coopératives. Une nouvelle approche de transformation des modèles par l'exemple grâce à un système multi agents évolutif (EMAS) pour donner un modèle cible.

Considérons un scénario réel, où un concepteur veut transformer un modèle source de donnée (diagramme de classe UML) vers un modèle cible (diagrammes ER). Cette transformation doit s'appuyer sur plusieurs opérations (supprimer, créer, diviser et fusionner. . . etc).

4.2 Approche proposée

Nous Présentons maintenant notre approche qui est destinée à apporter une transformation de modèle par l'exemple basé sur un système multi-agents évolutif. EMAS peut étendre le comportement des primitives génétiques conventionnels et système multi-agents. Les agents peuvent agir de manière proactive et réactive, ils peuvent prendre des décisions en appliquant des mécanismes d'interaction différenciés avec les autres agents et explorer l'espace des solutions de manière intelligente en utilisant des primitives génétique. L'environnement est composé d'un ensemble d'agents, où chaque agent essaye à trouver la meilleure solution, c'est-à-dire le meilleur modèle cible pour le modèle source. Le système peut être décrit comme : EMAS = <E ; ; w>, où E est l'environnement, est l'ensemble des ressources et W contient les types d'informations disponibles pour les agents du système. L'environnement est accessible ce qui signifie que les agents peuvent obtenir des informations précises et des informations complètes sur l'état de l'environnement. Il est également non déterministe car il n'y a aucune certitude sur l'état résultant d'une action donnée, puisque les agents peuvent modifier l'environnement pendant l'exécution. Nous formalisons d'abord les fondements pertinents de notre approche proposée. Ensuite on donne un aperçu du travail ainsi que des notions qui l'accompagnent.

4.2.1 Architecture système de l'approche proposée

Dans ce paragraphe, nous présentons l'ensemble des étapes du processus présenté dans la [figure 4.1] pour transformer un modèle SM source donné en modèle cible appropriée TM dans lequel les méta-modèles source et cible sont distincts, par exemple : transformant des diagrammes de classes UML aux diagrammes ER.

Notre solution est basée sur une architecture d'agents BDI, chaque agent possède :

- **Croyances :** les connaissances liées au domaine de la transformation de modèle,
- **Intentions :** les actions de transformation qui peuvent être appliquées sur SM pour obtenir un TM.
- les désirs ou le but :les buts de transformation TG1,. . . ,TGM .

4.2.2 Fondement théorique de l'approche proposée

Dans ce qui suit nous donnons une formalisation de notre approche proposée afin de clarifier ses principaux concepts. Un agent de transformation (le modèle Croyance-Désir-Intention) est un composant logiciel BDI qui se caractérise par quatre éléments : sa Croyance, ses désirs, son Intention, ses objectifs et son contexte.

Définition 3 (*Belief*)

Soit $\mathbb{B}$=$\{\mathcal{B}_1,..,\mathcal{B}_n\}$ *un ensemble de croyances représentant les informations que l'agent a obtenu sur son*

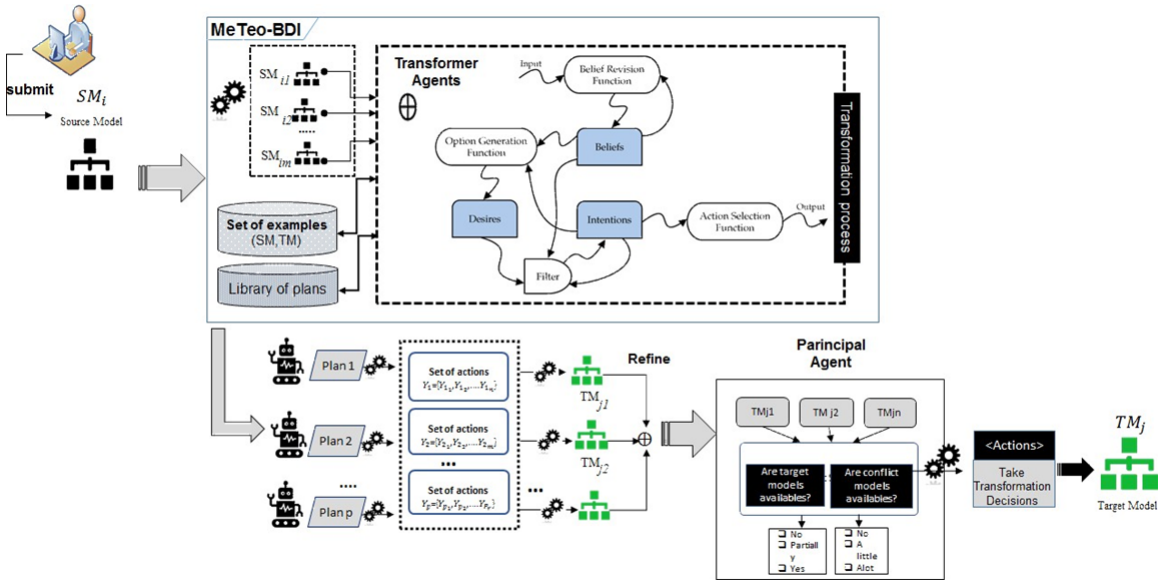


FIGURE 4.1 – Présentation du cadre l'approche proposée .

état interne et sur l'environnement externe. Une transformation de croyances $\mathcal{B}_i =< CT_i, cxt_i, Op_i, b_i >$ est associée à un contexte spécifique cxt_i, et fournit un ensemble d'opérations Op_i sous forme d'actions (Supprimer, Créer, Ajouter, déplacer) sur le modèle, ce qui permet de transformer le schéma du modèle SMi SM en un modèle cible $SM_i \in SM$, la condition de transformation CT_i représente la condition considérée par l'Agent BDI.

Définition 4 (*Context*)

Un contexte est un ensemble de paramètres $cxt = \{p_1, p_2, .., p_n\}$, chaque $p_i \in P = P_{Entity} \cup P_{Association} \cup P_{Attribute} \cup P_{Method} = \bigcup cxt_i$, où chaque P_j est un ensemble de paramètres appartenant à l'une des quatre catégories. De plus, $P_{Entity} \times P_{Association} \times P_{Attribute} \times P_{Method} = \{cxt_1, .., cxt_m\} = CXT$ est un ensemble de tous les contextes de transformation possibles liés au modèle source et cible.

Définition 5 (*Intention*)

L'intention INT de l'agent utilisée pendant le processus de transformation et le raisonnement (means-ends). L'intention INT_i, d'un contexte donné $\mathcal{B}_i$, permet de transformer le schéma du modèle source SM_i basé sur un sous-ensemble des paramètres de contexte cxt_i. L'Intention est un tuple $INT_i =< Param_i, Op_i >$.$Param_i \subseteq cxt_i$ (sous-ensemble du contexte cible/source), et Op_i est un ensemble d'opérations qui sont le moyen pour atteindre ses intentions, où $Op_i : P^{SM} \rightarrow P_{TM}$ et $k \leq cardinal(Param_i)$.. L'agent de transformation BDI sélectionne une opération à partir des opérations énumérées, en fonction d'un contexte approprié. Pour atteindre l'intention, l'agent sélectionne l'un des plans alternatifs applicables dans les mêmes moyens prévus et l'exécute.

Définition 6 (*Goal*)

Soit G l'ensemble des objectifs (Goals) que l'agent pourrait poursuivre, la représentation des objectifs au cours de leur processus de transformation et du raisonnement (means-ends) ainsi que lors l'exécution

de leurs activités. Les objectifs peuvent être utilisés comme une abstraction pour modéliser les fonctions autour lesquelles les systèmes peuvent sélectionner de manière autonome le comportement approprié. La transformation est motivée par l'objectif pour changer le modèle cible.

Définition 7 (*source model Language*)

L'agent peut extraire le modèle SM (c'est-à-dire conforme au langage de conception SM_{MM}). Puisque le fichier téléchargé est un langage de conception SMMM d'instance, il peut être modifié et visualisé.

Définition 8 (*Target model language*)

Chaque instance du modèle cible doit être conforme au langage TM_{MM} du méta-modèle du modèle cible. La vérification de conformité est également prise en charge par l'interface. Une fois la conformité de l'instance vérifiée, l'instance sera visualisée aux utilisateurs.

Définition 9 (*Heuristic Rule*)

L'agent BDI principal a des connaissances liées au domaine de transformation de modèle représentées comme de règles de cartographie (Mapping rules) HRR pour le raffinement d'un modèle cible donné. La $\mathcal{HR}$ contient des règles utilisées par l'agent BDI principal pour choisir le TM approprié à l'instance SM. Ces règles sont déduites de l'ensemble d'exemples de notre ensemble de données (dataset) Nous définissons une règle heuristique comme ECA type (Event - Condition - Action).

Les règles de mappage sont construites sur le modèle suivant : une liste d'au moins une condition (liée par un opérateur logique si plusieurs) si elle est satisfaite, implique ce qui défini dans l'action. Les deux conditions et l'action sont des opérations qui sont appliquées sur les éléments Contraintes, c'est-à-dire une caractéristique ou un attribut du Modèle Cible : Une règle HR_x^R est définie par (N^{Rx}, S^{Rx} , E^{Rx}, C^{Rx}, A^{Rx}) avec :

- N^{Rx} est le nom identifiant la règle,
- S^{Rx} désigne le type de composant de la règle. Les règles peuvent être associées soit à une valeur d'attribut (AVi), soit une caractéristique (Fi), soit une hiérarchie (attribut, attribution de valeurs) (FHj) de TM.
- E^{Rx} est l'opération qui déclenche la règle (Croyances qui déclenchent) et qui est appliquée à une instance donnée en entrée SMi.
- $E^{Rx} =< exclude|require|recommend|recommend - not|CardinalAttribute>$. Cette opération est contrôlée par ligne directrice, ce qui signifie que tous les enfants obligatoires doivent être sélectionnés chaque fois que ce parent est sélectionné, au moins une caractéristique doit être sélectionnée, exactement un de ces enfants doit être sélectionné.
- CRx est une condition définissant si la règle est déclenchée par un ERx courant. $condition_1\ \delta\ ...\ \delta\ condition_n \longrightarrow action$, avec $\delta \in \{\wedge, \vee\}$. Notez que l'opération abstraite est uniquement pour but de modélisation, car une condition est une opération de valeur ou une opération simple. Chaque $condition_i$ est défini comme suit : $condition_i = Element.attribute\ \theta\ Value$, avec $\theta \in \{<;\leq;>;\geq\}$ et l'élément de $SM_{MM} <$ et ses fragments.
- A^{Rx} est la séquence d'actions déclenchées. Les actions s'appliquent aux composants (feature, child-feature, attribute…) du modèle cible. Les actions doivent être incluses ou exclues d'un modèle cible.

Dans ce qui suit, nous décrivons cette transformation :

- **R1 :** chaque *concept* ou classe $c \in SM_{MM}$ est mappé à une caractéristique (Fi), soit à une hiérarchie (variantes, enfants) (FHj) de TM.
- **R2 :** chaque attribut $a \in c.attributes$ est mappé à une définition de propriété (une valeur d'attribut (AVi)) de TM.
- **R3 :** chaque association entre les classes c_1, c_2,...,c_n de SM_{MM} est mappée soit à une valeur d'attribut (AVi), soit à une caractéristique (Fi), soit à une hiérarchie (variantes, enfants) (FHj) de TM.

Par exemple, nous définissons des règles de transformation, qui représentent les règles de base de la base de données de déploiement. Le premier vérifie que le nombre AssociationClasses d'un SM_i est toujours supérieur à 2, le second vérifie que les Types d'attribut sont de haute cardinalité, et assure une cardinalité de type *many-many* (*).

Dans ce cas, il est recommandé de créer une nouvelle relation.

Définition 10 (*Knowledge Base*)

> *Soit $D = f(X_i; Y_i)1^N$ représente un ensemble de taille N, où $Xi = (x_{i1},..., x_{in})$ est un vecteur de caractéristiques avec n caractéristiques et Yi est le paramètre cible. Dans notre approche , Xi est constitué d'attributs (ou caractéristique) associés à une instance particulière du modèle source SM_i dont il est transformé. Soit Y_i l'ensemble des actions de transformation correspondant à $Xi = (x_{i1},..., x_{in})$. La sortie, Yi, représente label $class_k$ correspondant à l'instance Xi qui est ('createClass', 'deleteAttribute', 'addKey', 'Split', 'Merge', ...).*
> *Chacun x_i décrit le composant de SM et TM de l'instance de l'ensemble de données (dataset).*

En résumé, le problème de transformation de modèle (SM) pour produire un modèle cible optimal TM est un modèle d'entrée-sortie tel que décrit ci-dessous : Optimisation de transformation du modèle :

Entrée (Input) :

- Un Modèle Source (SM).
- Un ensemble de contraintes (exigences des utilisateurs ou contraintes d'applications).
- But (Goal) : Le but est de sélectionner un TM optimal en se basant sur un ensemble d'actions de transformation.

Sortie (Output) :

$\mathcal{TM}^{optimal}$: un modèle cible optimal.

la taille d'espace de recherche du problème de sélection du $\mathcal{TM}^{optimal}$, augmente exponentiellement par rapport au nombre d'attributs des modèles cible candidats. Le problème de sélection d'un ensemble de TM est dit *NP-complet* dans la littérature.

4.2.3 Organisation conceptuelle de l'approche proposée

L'approche proposée est basée sur un agent BDI comme illustré par la Figure 4.2. Chaque intention (instance d'intention) d'un modèle cible donné est décrite par un ensemble d'actions de transformation. Ces actions sont liées à différentes catégories. La méta-modélisation de ces catégories d'actions et de leurs attributs conduit à de nombreuses classes et énumérations. La figure 9 nous donne une vue abstraite de nos classes de méta-modèle qui correspondent aux

actions de transformation. La vue est exprimée sous la forme d'une arborescence de fonctionnalité [17].

- **Action sur la relation :** les éléments de cette catégorie sont liés à la relation du schéma (par exemple tables/colonnes, table de partitionnement).
- **Action sur l'Attribut :** les éléments de cette catégorie sont liés à la relation des d'attribut (par exemple : ajouter, supprimer....etc).
- **Action sur l'association :** l'association signifie les concepts utilisés pour lier l'objet du modèle (relation, table, classe...). L'association peut être unaire, binaire, n-aire, récursive, relations ternaires dans un schéma de modèle. Le fonctionnement de l'association peut être restreint par un ensemble d'actions (par exemple supprimer, créer).
- **Action sur la méthode :** les éléments de cette catégorie sont liés à la relation d'attribut (par exemple : ajouter, supprimer, ...).

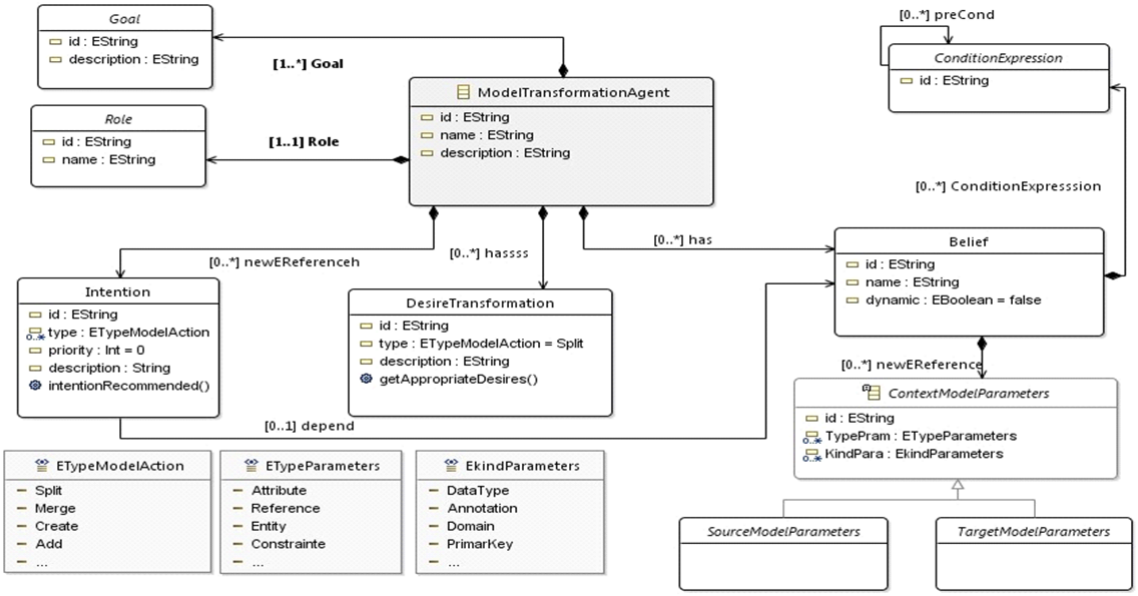


Figure4.2 – Approche basée sur le modèle BDI.

4.2.4 Éléments fondamentaux de l'approche

Notre problème est modélisé par la méthodologie VOWEL. Cette méthode consiste à déterminer les cinq composantes de l'EMAS : les Utilisateurs (U), l'Environnement (E), Les agents (A), les Interactions (I) et l'Organisation (O) .

4.2.4.1 Utilisateurs (U)

C'est le designer et le concepteur du modèle. L'intervention du concepteur est importante . Il valide certaines recommandations fournies par le système et initialise les paramètres suivants tels que : un ensemble de contraintes comme :

- S : espace de stockage requis d'un modèle donné.

— W : nombre maximum de fragments.

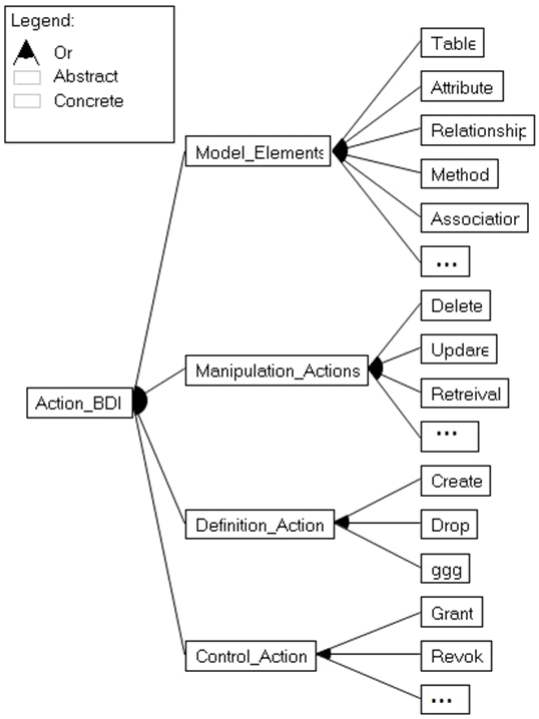


FIGURE 4.3 – Intention de l'Agent BDI.

4.2.4.2 Environnement (E)

Consistent a SM par exemple : le SGBD(système de gestion de base de données), API ...

4.2.4.3 Agent (A)

Le système se compose de deux types d'agents :

— Agent principal (Main Agent) : cet agent effectue les tâches suivantes :
  — **Extraction , fragmentation et diffusion :** l'agent accède au SM pour extraire automatiquement les métadonnées (c'est-à-dire les propriétés du modèle) grâce à la méthode « getParametersSM », puis il le stocke dans une donnée bien structurée pour être utilisé par transformers agents. Enfin, il envoie la transformation demandé à tous les agents transformateurs.
  — **Analyse :** Analyser les différents modèles cibles proposés par les agents transformer.
  — Raffinement : après réception de toutes les propositions de modèles cibles des agents transformateur BDI, l'agent principal utilise la matrice binaire Refine (HRj , TM) où les lignes sont les propositions de modèles cibles et les colonnes décrivent les règles heuristiques. Chaque proposition sera vérifiée par rapport à toutes les heuristiques existantes, l'agent principal renvoie le modèle cible qui satisfait les règles heuristiques.

— **Fusionnement :** Heuristique : Pour la bonne marche de notre approche proposée un ensemble de règles heuristiques, nous proposer un langage de conception permet au concepteur d'ajouter ces heuristiques via notre DSL (c'est-à-dire Conception de langage spécifique au domaine), voir la figure 12 pour avoir des connaissances de base initiales sur la transformation de modèle. Une instance d'une règle heuristique (classe d'instance HeuristicRules) est rendue explicite utilisant en entrée les concepts liés au méta-modèle du modèle cible (instance classe TargetModelMetaModel). Cette formulation nécessite une combinaison de conditions arithmétiques et définies. Une condition doit être décrite pour qu'une valeur soit affectée à la règle de mappage. La condition est une expression booléenne ou un ensemble de « BooleanExpressions » connecté avec LogicalOperators. Une expression booléenne est satisfait si la relation entre l'attribut de quantité QuantifiableElement, l'attribut de valeur BooleanExpression et le comparateur défini sont vrais. Pour un nouveau type de transformation, nous pouvons personnaliser et étendre un ensemble de règles heuristiques pour accompagner ce type de transformation.

toutes les heuristiques existantes, l'agent principal renvoie le modèle cible qui satisfait les règles heuristiques.

— **Agent transformateur (Agent Transformer) :** Le rôle de cet agent est de transformer un SM reçu sur la base de ses convictions. Dans notre cas, le nombre d'agents transformateurs dépend du nombre d'experts de transformation. La transformation d'un SM donné vers un TM élu est obtenue en utilisant des primitives génétiques, c'est-à-dire un croisement et une mutation qui représentent le choix au cours du processus de transformation pour atteindre une séquence de TM. En d'autres termes, les combinaisons de ces primitives ont fourni le TM élu qui satisfait l'objectif de transformation. La sélection du meilleur TM est calculée en utilisant la similarité pour les données binaires.

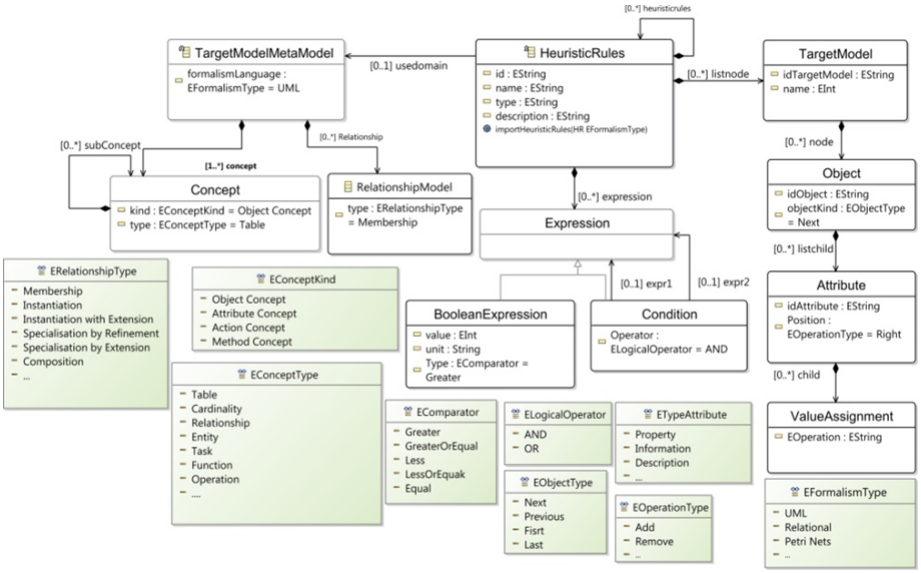


FIGURE 4.4 – Extrait de notre langage de conception de règle heuristique utilisé par l'agent principal.

Agents	Goals	Resources	Abilities	Type
Intra Transformer BDI	Compatible	Insufficient	Insufficient	Simple collaboration coordinated
Agent Transformer BDI	Incompatible	Insufficient	Insufficient	Collective conflict on resources
Principal BDI Transformer BDI	Incompatible	Insufficient	Insufficient	Coordinated collaboration

Tableau 4.1 – Situation des interactions entre les agents.

4.2.4.4 Interactions (I)

L'interaction dans ce cas est divisée en deux types :

- Interaction dans le cas où les agents travaillent localement pour générer un ensemble de recommandations pour un modèle source donné isolément et les ressources utilisées par ce dernier dans ce cas la communication est directe (envoyer un message).
- Inter action dans le cas d'une sélection multiple du modèle cible, l'agent transformer prend en compte l'impact de chaque décision de l'agent transformer sur l'autre. Les agents transformer pense en termes locaux, l'agent principal raisonne globalement

La classification des situations d'interaction entre les différents agents, en fonction des objectifs, ressources et les compétences des agents sont représentées dans le tableau suivant :

4.2.5 Processus de transformation

Notre processus de transformation est le suivant :

- Transformer des fragments du SM selon les croyances des agents transformers, ce processus de transformation (Plan) qui abouti a un TM est basé sur des opérateurs génétiques.
- Raffiner les transformations (TM) des agents transformers par un main agent.

4.2.5.1 Étape 1 : Trouver la meilleure solution de modèle de transformation

La sélection d'un modèle cible élu est d'ordre NP-Complet, ce qui nous a poussé à recourir à l'utilisation des primitives génétique. Dans ce qui suit, nous allons détailler l'étape 1. Principe de codage : Dans la phase de préparation du codage de la solution, les propriétés des objets TM sont divisées en sous-propriétés. Un schéma de TM redéfinit la division de valeurs d'affectation de ces propriétés lorsque certaines valeurs d'affectation sont regroupées en partitions. Une propriété composée de ni sous-propriétés peut être décomposé au moins en une partition et au plus en ni partitions. Le codage est fait comme suit : $CODE[i]\ [j] = k$ si les sous-propriétés sd_j de l'attribut A_i appartiennent à la partition P_k de cet attribut où $1 \leq i\ \ n$, $1 \leq j\ \ n_i$, $1 \leq k \leq n_i$.

Deux sous-propriétés sdj, sdl appartiennent à la même partition si et seulement si sd_j, sd_l. Nous illustrons ce codage sur la Figure 4.7, où la partie gauche représente la division d'attributs en sous-domaines et la partie droite représente le SM correspondant codage du schéma. Comme

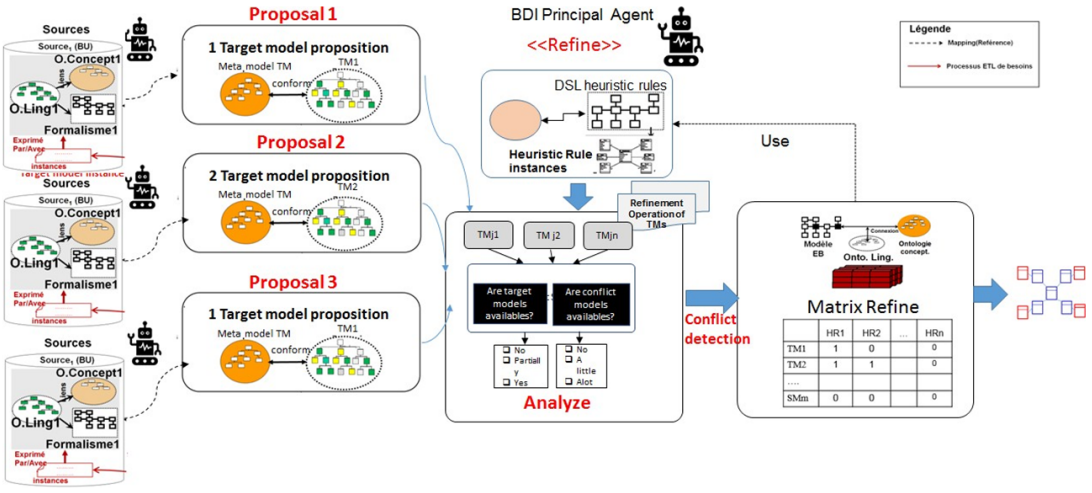


FIGURE 4.5 – Processus de transformation global.

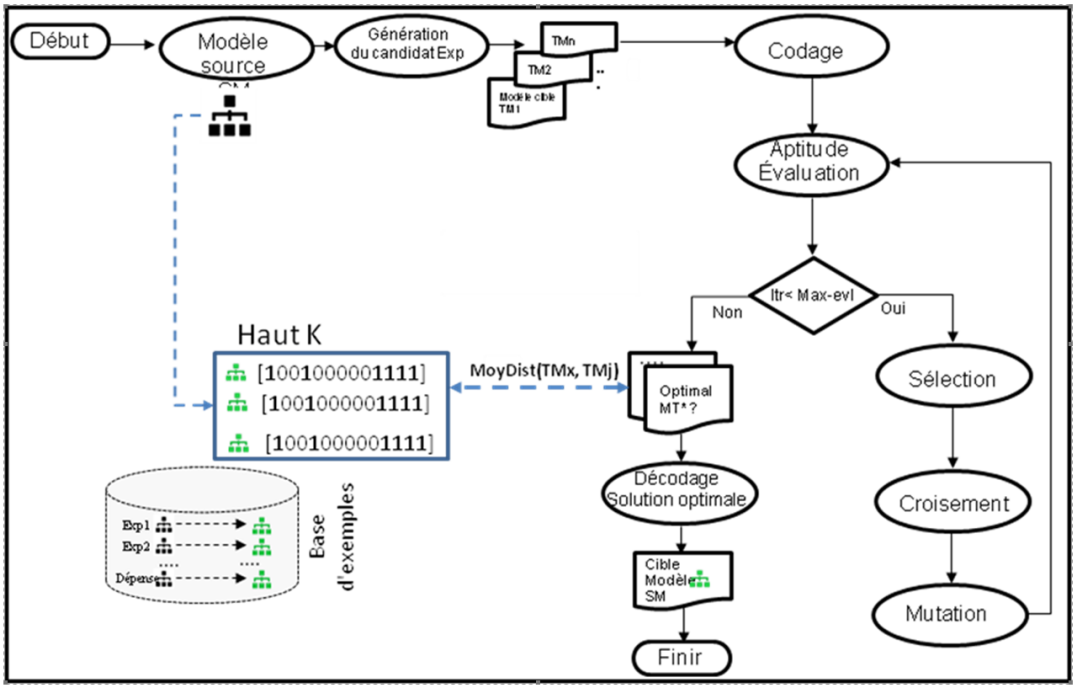


FIGURE 4.6 – Processus de transformation (Agent transformer)..

décrit dans la Figure 4.7, les domaines des variables représentent maintenant les instances qui peuvent être codées, par exemple, les instances de : Générer un schéma de transformation à partir d'un encodage : Notre algorithme exploiter le mécanisme de codage afin de générer le schéma de transformation correspondant à un code donné. Avant de générer les fragments de chaque relation de modèle cible TM, nous commençons par une étape d'identification du partitionnement des domaines attributaires de chaque relation du modèle source SM. Rappelons que la i éme partition de l'attribut AK noté (Pik est constitué d'un ensemble de sous-domaines Sik défini par : Sik =

fSik/CODE[k][j]= ig o sdj représente le j ème sous-domaine d'un attribut Ak.

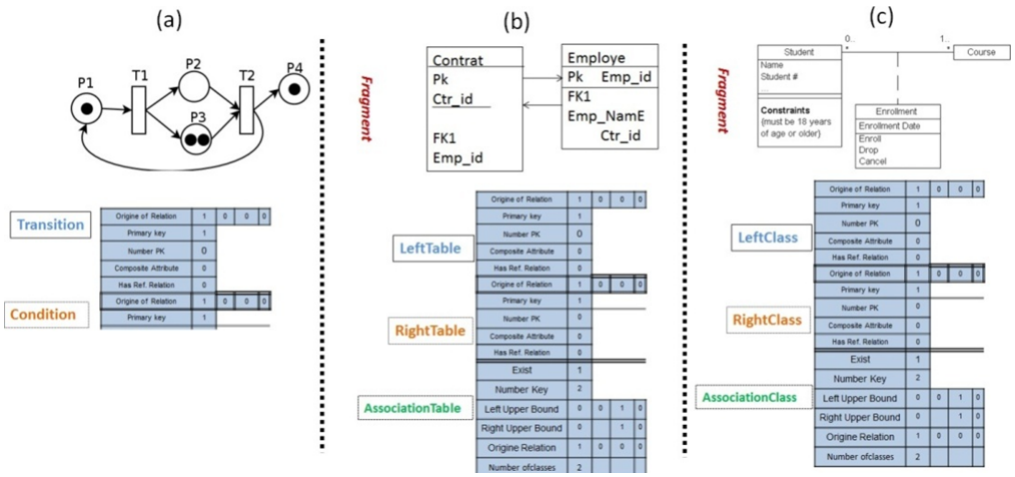


FIGURE 4.7 – Exemple de codage de solution : (a) Modèle de réseau de petri, (b) Modèle Relationnel, (c) Modèle UML.

La génération des fragments de TM est obtenue en utilisant la règle suivante : Chaque relation de fragment Rji de la relation Ri est défini par une clause ClRji représentant une conjonction ,16 auteurs supprimés en raison d'une longueur excessive de prédicats. Chaque prédicat dans cette conjonction définit une partition d'une fragmentation d'attribut.

$$Cl_{Rji} : pp^{l}{}_{1} \wedge pp^{r}{}_{2} \wedge \cdots \wedge pp^{s}{}_{k} \tag{1}$$

La génération des fragments de la relation Ft contient des clés étrangères est définie par une conjonction de clauses où chaque clause définit un fragment d'une table, comme suit :

$$F_t : Cl_{Rj1} \wedge Cl_{Rk2} \wedge \cdots Cl_{Rlg} \tag{2}$$

$(1 \leq j \leq m_1,\ 1 \leq k \leq m_2, \cdots, 1 \leq l =\leq m_g)$ ou g, m_i et Cl_Rji représentent : le nombre de relations fragmentées, le nombre de fragments de la relation R_i et la clause définissant le fragment de relation $R^j{}_i$. Opérateurs génétiques Les GA utilisent les opérateurs génétiques : croisement, mutation et sélection. L'opérateur de sélection (sélection à la roulette) permet de sélectionner les meilleurs individus d'une population à participer à la prochaine génération de la population. La traversée s'effectue avec un taux de croisement de Tc, qui est généralement important afin de pouvoir combiner des solutions et former générations de meilleure qualité. L'opérateur de mutation permet la modification occasionnelle des gènes d'un individu pour permettre l'exploration de certains domaines dans la codification des individus où le croissement ne peut pas explorer. La mutation a lieu avec une petite Tm. La Figure 4.9 illustre un exemple d'opérateurs de mutation croisé.

Fonction objective pour l'algorithme génétique : Nous avons formalisé le problème de fragmentation horizontale comme un problème de minimisation. Pour traiter un problème de minimisation avec les algorithmes génétiques, une transformation de la fonction à minimiser est souvent[12]

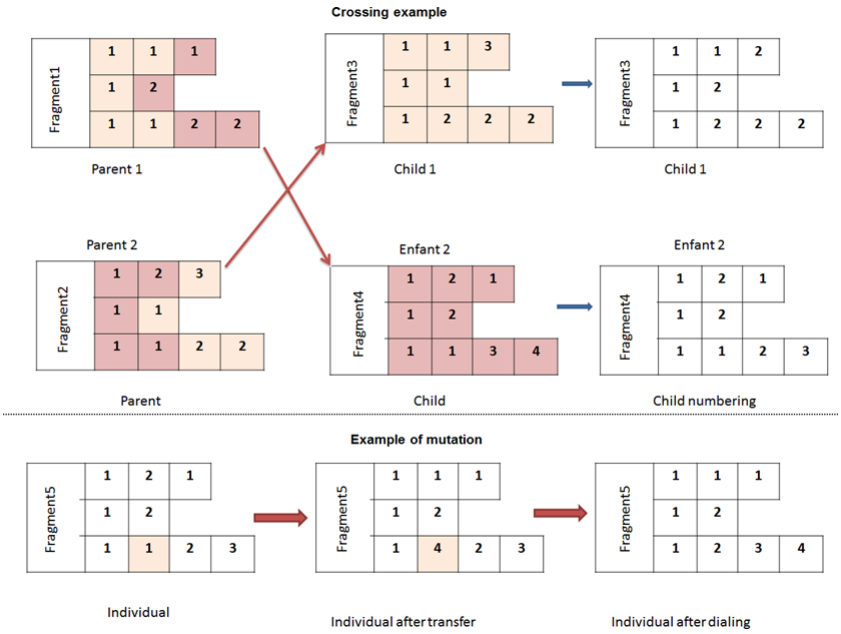


FIGURE 4.8 – Exemple des opérations génétiques.

effectuée pour le ramener à un problème de minimisation. Par conséquent, nous proposons la minimisation de la distance moyenne entre le modèle cible et l'existant de modèles cibles similaires de l'ensemble de données. La forme physique peut être calculée comme suit :

$$Fitness = Min\ DistAVG \sum_{i=1}^{n} (TM_x, TM_i) \tag{3}$$

Comme nous avons dit précédemment, les TM sont représentés par des données binaires. Nous comparons deux cibles modèles basés sur les coefficients de similarité pour les données binaires. Ce dernier est basé sur coefficient de Russel et Rao, qui est simplement la proportion de correspondances positives. (Voir l'équation 4).

$$Dist(O_i, O_j) = \frac{a}{a+b+c+d} \tag{4}$$

D'autres méthodes peuvent être utilisées pour calculer les coefficients de similarité pour les données binaires tels que le coefficient de Jaccard et le coefficient de Dice (voir les équations 5 et 6).

$$Dist(O_i, O_j) = \frac{a}{a+b+c} \tag{5}$$

$$Dist(O_i, O_j) = \frac{2}{a} 2a+b+c \tag{6}$$

Les valeurs a,b,c et d sont calculées à l'aide du tableau de contingence2, avec a (resp. b,c,d)=

nombre de positions où i égale 1 (resp. 1,0,0) et j égale 1 (resp. 0, 1, 0), que signifie le nombre de fois que les deux objets ont les mêmes occurrences positives. Exemple 4 : Considérons deux modèles cibles TMa et TMb, nous avons besoin pour calculer le tableau de contingence pour : TMa = (1, 0, 1, 1, 1) et TMb = (1, 0, 1, 0, 0). Le tableau de contingence est calculé comme illustré à la figure 10. En utilisant le Russel et la formule du coefficient Rao (voir équation 4) pour calculer la distance entre les deux modèles cibles : a=2, b=2, c=0, d=1, Dist(TMa ; TMb) = 0 ; 4.

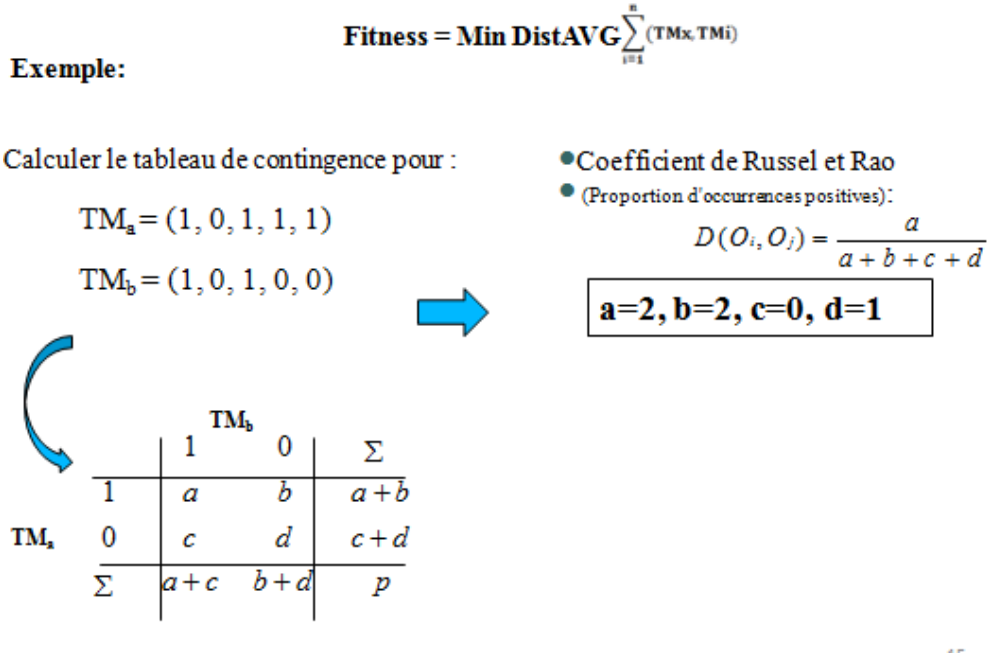


FIGURE 4.9 – exemple de calcule de la fonction objective.

4.2.5.2 Étape 2 : Raffinement des modèles cibles proposés

Afin de raffiner les solutions proposées par les agents transformateurs, nous définissons des règles heuristiques au niveau de l'agent principal pour recommander le modèle cible optimal. L'algorithme 1 montre le processus de raffinement de TM, nous commençons par recevoir les modèles cibles proposés par les agents transformateurs. Nous nous sommes basés sur une analyse automatique utilisant la dépendance d'analyseur comparant la valeur existante de TM dans une séquence de données binaires. Pour chaque séquence de données binaires, nous pouvons suggérer des fonctionnalités alternatives comme Obligatoire ou Optionnel. Pour chaque modèle cible TM, nous évaluons s'il est réalisable en vérifiant les contraintes pour renvoyer le modèle cible final qui satisfait au maximum les règles l'heuristiques. Pour chaque TMi cible dans notre TM, nous analysons l'intégration des contraintes (IC). L'identification de la contraintes C pet être vraies, fauss ou indéfinies (C^{undef}, C^{true}, C^{false}) le resultat est utilisé pour générer le modèle cible optimal.

Input

— Un ensemble de modèles cibles TM=TM1,TM2,…..,TMn ;// vecteur générer par agent

```
begin
      C=Cundef =Ctrue=Cfalse=f?g;
    foreach target model TM 2 T M do
        foreach Heuristic rules HRi 2 HR do
            foreach constraint ICj of HRi do
                if ICi is feasible and V alue(TMi) 2 HRi then
                    Ctrue Ctrue [ ICi;
                end
                    else if ICi is not feasible and V alue(TMi) 2= HRi then
                        Cfalse Cfalse [ ICi ;
                        end
                else
                    Cundef Cundef [ ICi;
                end
            end
        end
                C Cundef [ Ctrue [ Cfalse ;
   end
end
Return the TM according to the maximum of identified constraints (IC)
return TM;
```

FIGURE 4.10 – Algorithme de Raffinement TM.

transformer

- Un ensemble des heuristiques HR=HR1,HR2,. . . ,HRp ;
- Un ensemble des contraintes liées aux HR, IC=IC1,IC2,. . . .,ICm ;

Output

- Le modèle cible : TMr= TMi TM ; // le modèle cible satisfait les conditions des heuristique.

On peut distinguer quatre scénarios possibles pour le raffinement :

- (a) Même décision,
- (b)Sans conflit,
- (c) Transformation partielle,
- (d) Avec conflit.

Dans le scénario « même décision », l'agent principal de recommande la même décision sans utiliser de règles heuristiques. Dans le scénario de transformation partielle », l'agent principal du BDI recommande un modèle cible en combinant les différentes propositions d'agent BDI transformateur.

Dans le scénario « avec conflit » l'agent principal BDI utilise l'algorithme 1 pour raffiner la solution finale.

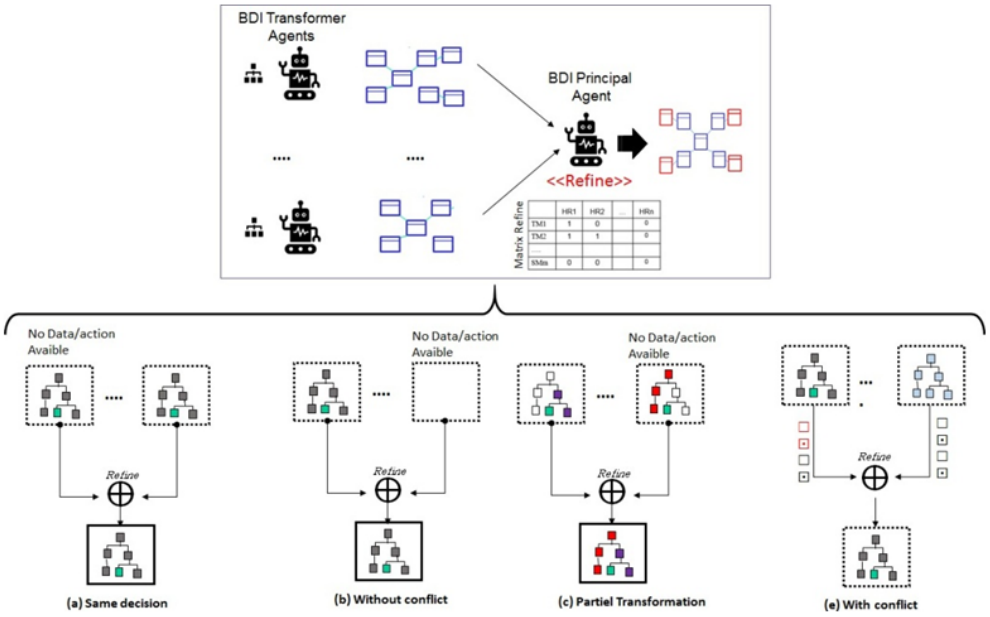


FIGURE 4.11 – Scenarios de Raffinement.

4.3 Conclusion

Dans ce chapitre nous avons présenté la transformation de modèles par l'exemple basé sur l'architecture d'agents BDI pour obtenir une solution optimal à laid d'utilisation des primitifs génétique.

Chapitre

5

Etude de Cas

Sommaire

5.1 Introduction

Afin de motiver nos travaux de recherche d'un point de vue pratique, nous avons développé une application que nous appelons MeteoBDI pour assister le concepteur dans ses tâches de la transformation des modèles. Nous présentons dans ce chapitre l'environnement d'implémentation, l'outil implémentant notre approche SMA ainsi qu'une série d'expériences en utilisant ce dernier.

5.2 Environnement d'implémentation

Nous présentons ici brièvement l'environnement d'implémentation et outils utilisés durant la phase de réalisation.

5.2.1 Langage de programmation

Nous avons implémenté notre plateforme avec le langage de programmation JAVA vu que ce dernier est un langage simple, intuitif, orienté objet, performant et dynamique. C'est aussi un langage multiplateforme disposant d'une JVM (Java Virtual Machine) lui permettant de s'exécuter dans des environnements hétérogènes en permettant une indépendance envers les réseaux et les systèmes d'exploitation.

5.2.2 Environnement de Développement Intégré (IDE)

Le choix de l'IDE est souvent dicté par des préférences personnelles. Néanmoins, il est conseillé d'utiliser Eclipse ou Netbeans dans notre cas. Ces deux IDEs facilitent le développement en Java SE et ME et permettent l'utilisation de plateforme JADE. Les développements de ce projet ont été effectués en utilisant Netbeans.

5.2.3 Plateforme d'agents

Dans le choix de plateforme, on a eu le choix entre différentes plateformes d'agent Jade, Aglet, Voyager, … mais on a choisi JADE. Ce choix est motivé par le fait que JADE est open source et facile à manipuler vu qu'il est développé par JAVA. Nous pouvons ajouter le fait qu'il est compatible FIPA t doté d'une API puissante.

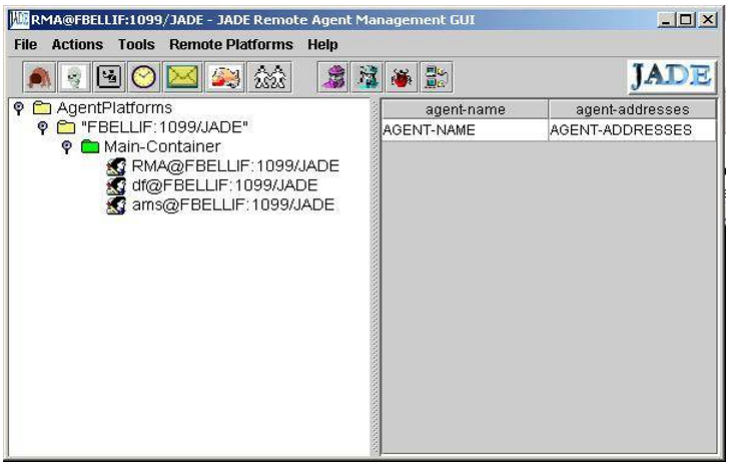


FIGURE 5.1 – L'environnement d'exécution des agents.

Nous présentons dans ce qui suit la présentation de notre outil développé AdminSMA et un scénario d'utilisation.

5.2.4 Présentation de l'outil réalisé

L'application fonctionne sur des machines différentes ou sur une seule machine. La console d'administration JADE pourra être gérée par les agents sur une seule machine. Ce chapitre explique également la manière dont les différents agents du système vont communiquer. Notre outil permet de charger un schéma en étoile d'un entrepôt de donnée et de spécifier une charge de requêtes à optimiser. MeteoBDI permet de réaliser les opérations suivantes :

- L'extraction des metadonnées du modèle source.
- la fragmentation de modèle source.
- la transformation de modèle vers un modèle cible conforme à un métamodèle.
- la génération de script de création du modèle cible.

5.2.5 Scénario d'utilisation de notre Prototype

Nous présentons dans cette section les principales fonctionnalités de l'outil à travers des captures écran illustrant les interfaces de ce dernier. Pour bien décrire ces interfaces, nous considérons un scénario d'exécution et de manipulation de l'outil.

5.2.5.1 Extraction de modèle source

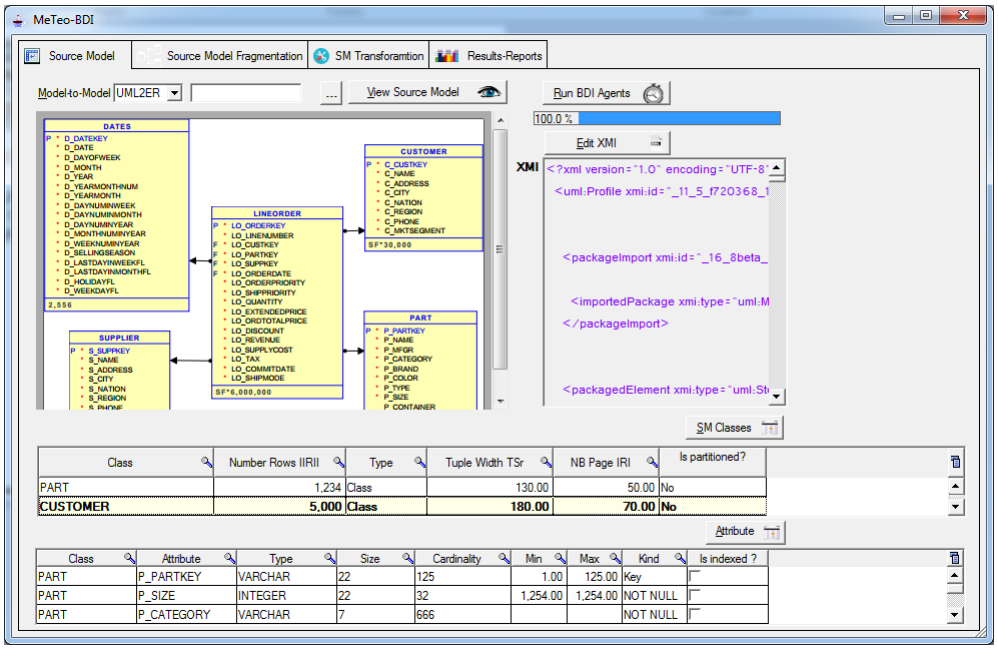


FIGURE 5.2 – L'environnement d'exécution des agents.

5.3 Conclusion

Tout au long du présent chapitre, nous avons présenté notre outil et discuté les mesures de performances que nous avons mené pour valider l'approche proposée. Le prototype realisé montrent la faisabilité de notre approche. Néanmoins, dans le but d'avoir de meilleures performances, des améliorations peuvent être apportées.

5.2.5.2 Fragmentation de modèle

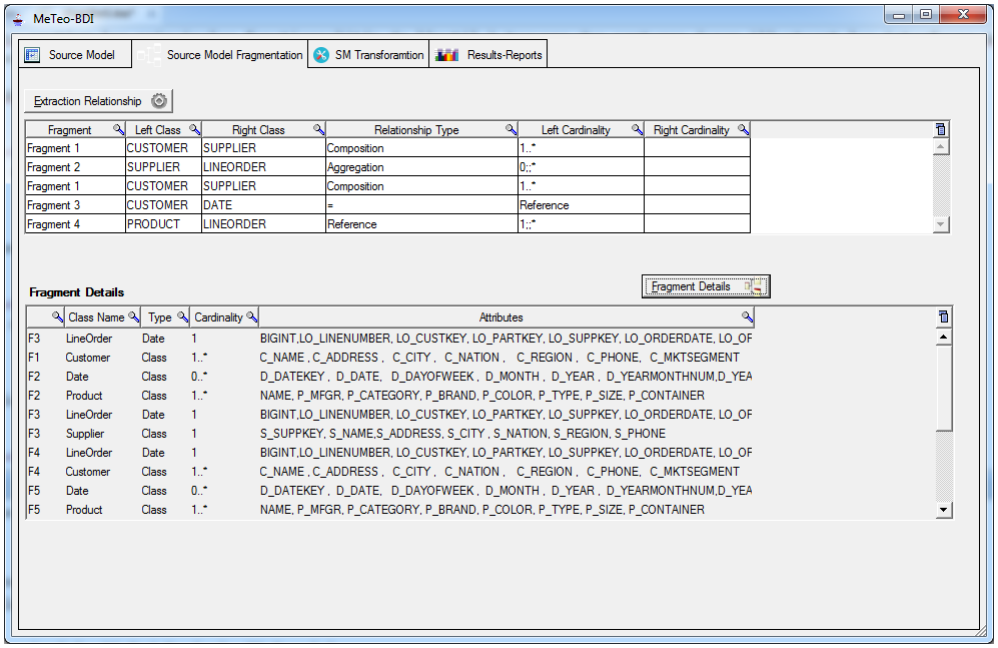


FIGURE 5.3 – L'environnement d'exécution des agents.

5.2.5.3 Transformation de modèle

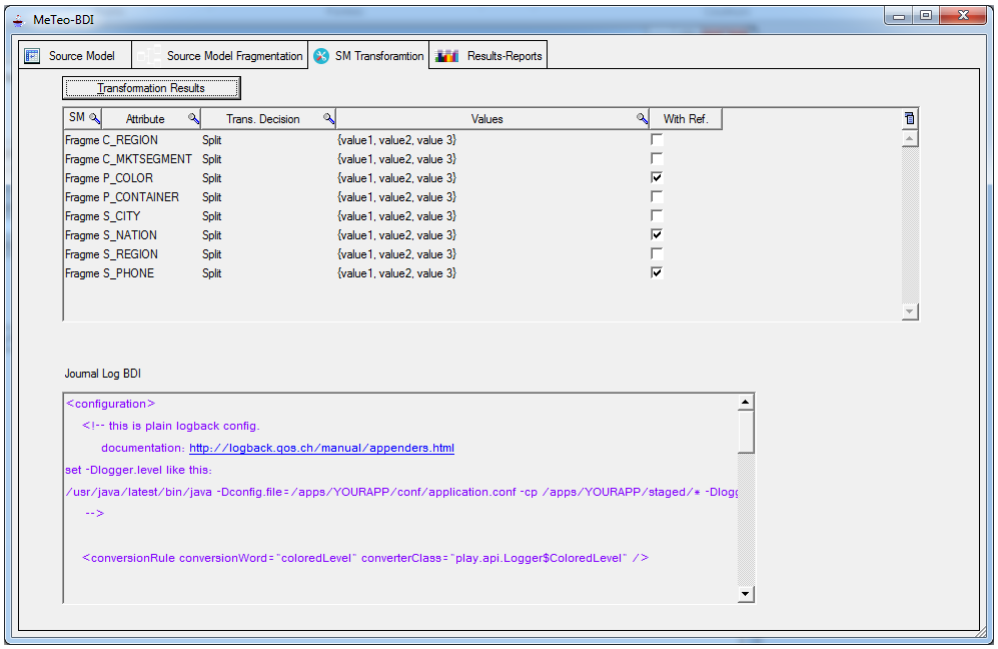
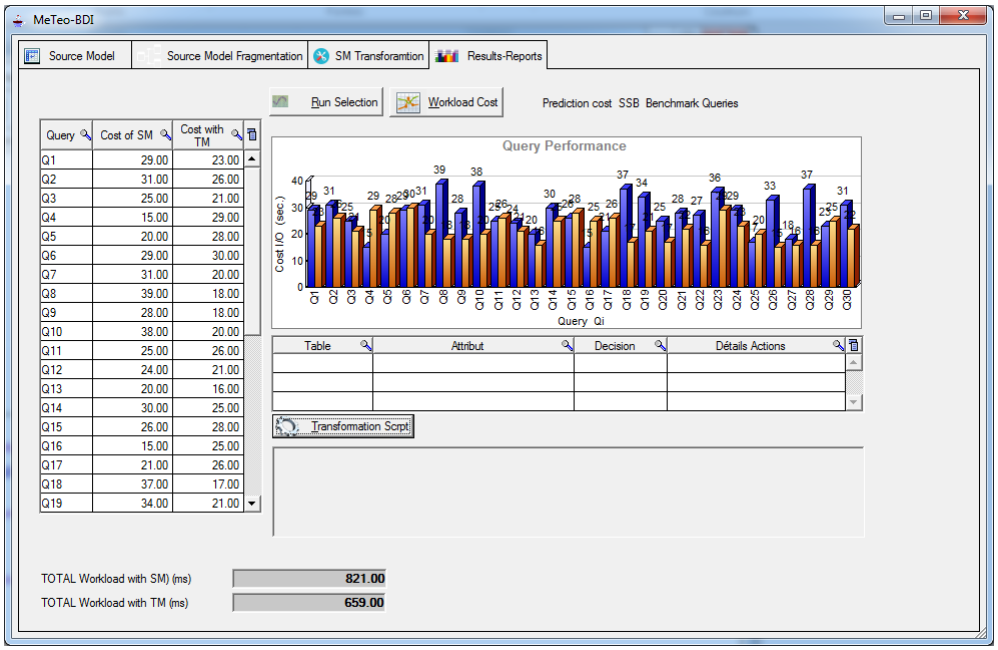


FIGURE 5.4 – L'environnement d'exécution des agents.

5.2.5.4 Estimation de qualité de la solution



Query	Cost of SM	Cost with TM
Q1	29.00	23.00
Q2	31.00	26.00
Q3	25.00	21.00
Q4	15.00	29.00
Q5	20.00	28.00
Q6	29.00	30.00
Q7	31.00	20.00
Q8	39.00	18.00
Q9	28.00	18.00
Q10	38.00	20.00
Q11	25.00	26.00
Q12	24.00	21.00
Q13	20.00	16.00
Q14	30.00	25.00
Q15	26.00	28.00
Q16	15.00	25.00
Q17	21.00	26.00
Q18	37.00	17.00
Q19	34.00	21.00

FIGURE 5.5 – L'environnement d'exécution des agents.

Quatrième partie

Conclusion et perspectives

Chapitre

6

Conclusion and Perspectives

« Research is to see what everybody else has seen, and to think what nobody else has thought. »

— Albert Szent-Gyorgyi

Sommaire

6.1 Conclusion

Le MDE est un paradigme qui promet de réduire la complexité du logiciel par l'utilisation intensive de modèles et des transformations automatiques entre modèles source vers un modèle cible. l'automatisation implique la récolte des connaissances sur la façon dont la TM doit être effectuée, et ces connaissances sont souvent non disponibles ou difficiles à obtenir. Les approches de TM par l'exemple viennent pallier cette situation, en particulier, le MTBE qui vise à apprendre automatiquement les TMs à partir d'un ensemble de paires de modèles sources et cibles fournis en guise d'exemples. Nous avons proposé dans le cadre de ce projet un processus d'apprentissage d'une transformation exogène (un model source vers un model cible). L'approche proposée est basée sur l'absence des règles, il est nécessaire de s'appuyer sur les exemples. Nous devrons savoir que la transformation d'un modèle à un autre est un ensemble d'actions à réaliser telles que (diviser, supprimer, créer...) sur la base de la transformation les actions réalisant une instance d'un modèle source peuvent générer un espace de solutions considérable avec complexité liée aux paramètres des objets SM. Nous Présentons notre approche qui est destinée à apporter une transformation de modèle par l'exemple basé sur un système multi-agents évolutif. EMAS peut étendre le comportement des primitives génétiques conventionnels et système multi-agents. Les agents peuvent agir de manière proactive et réactive, ils peuvent prendre des décisions en appliquant des mécanismes d'interaction différenciés avec les autres agents et explorer l'espace des solutions de manière intelligente en utilisant des primitives génétique. Enfin nous avons validé et implémenté ce travail par le développement d'un outil prototype qui montre la faisabilité de l'approche proposée.

Abréviations

MDE Model Driven Architecture

OMG Object Management Group

CIM Computation Independent Model

PIM Platform Independent Mode

MTBD Model Transformation By Demonstration

MTBD **ModelTransformationByExemples**

IDM(MDE) IngénierieDirigéeparlesmodeles

TMPE(MTBE) Transformation de Modeles par l'Exemple

AE Algorithme Evolutionnaires

TM Transformation de Modeles

TM Modéle cible

SM Modéle Source

UML Unifed Modeling Language

AG Algorithme Génitique

Bibliographie

[1] B. Athamena et Z. Houhamdi. « A petri net based multi-agent system behavioral testing ». In : *Modern Applied Science* 6.3 (2012), p. 46 (cf. p. 26).

[2] I. Baki et H. Sahraoui. « Multi-step learning and adaptive search for learning complex model transformations from examples ». In : *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25.3 (2016), p. 1–37 (cf. p. 25).

[3] I. Baki et al. « Learning implicit and explicit control in model transformations by example ». In : *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2014, p. 636–652 (cf. p. 21).

[4] Z. Balogh et D. Varró. « Model transformation by example using inductive logic programming ». In : *Software & Systems Modeling* 8.3 (2009), p. 347–364 (cf. p. 25).

[5] G. Bergmann et al. « Advances in model transformations by graph transformation : Specification, execution and analysis ». In : *Rigorous Software Engineering for Service-Oriented Systems*. Springer, 2011, p. 561–584 (cf. p. 24).

[6] G. Bergmann et al. « Incremental pattern matching in the VIATRA model transformation system ». In : *Proceedings of the third international workshop on Graph and model transformations*. 2008, p. 25–32 (cf. p. 17).

[7] K. Berramla et al. « Model Transformation by Example with Statistical Machine Translation ». In : *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2020, Valletta, Malta, February 25-27, 2020*. Sous la dir. de S. Hammoudi, L. F. Pires et B. Selic. SCITEPRESS, 2020, p. 76–83. doi : 10.5220/0009168200760083. url : https://doi.org/10.5220/0009168200760083 (cf. p. 8).

[8] E. Bousse et al. « Execution framework of the GEMOC studio (tool demo) ». In : *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*. 2016, p. 84–89 (cf. p. 9).

[9] K. Czarnecki et S. Helsen. « Feature-based survey of model transformation approaches ». In : *IBM systems journal* 45.3 (2006), p. 621–645 (cf. p. 9, 18).

[10] X. Dolques et al. « Learning transformation rules from transformation examples : An approach based on relational concept analysis ». In : *2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops*. IEEE. 2010, p. 27–32 (cf. p. 21).

[11] M. Eysholdt et H. Behrens. « Xtext : implement your language faster than the quick and dirty way ». In : *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. 2010, p. 307–309 (cf. p. 10).

[12] S. Faridmoayer, M. Sharbaf et S. Kolahdouz-Rahimi. « Optimization of model transformation output using genetic algorithm ». In : *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*. IEEE. 2017, p. 0203–0209 (cf. p. 26).

[13] 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. 22).

[14] M. Faunes, H. Sahraoui et M. Boukadoum. « Genetic-programming approach to learn model transformation rules from examples ». In : *International Conference on Theory and Practice of Model Transformations*. Springer. 2013, p. 17–32 (cf. p. 21).

[15] I. García-Magariño. « A set of method fragments for developing multi-agent systems assisted with model transformations ». In : *2011 11th International Conference on Intelligent Systems Design and Applications*. IEEE. 2011, p. 30–35 (cf. p. 26).

[16] I. García-Magariño, J. Gómez-Sanz et R. Fuentes-Fernández. « Ingenias development assisted with model transformation by-example : A practical case ». In : *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*. Springer. 2009, p. 40–49 (cf. p. 26).

[17] 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. 25).

[18] A. Ghannem, G. El Boussaidi et M. Kessentini. « Model refactoring using interactive genetic algorithm ». In : *International Symposium on Search Based Software Engineering*. Springer. 2013, p. 96–110 (cf. p. 23).

[19] S. Jean, L. Bellatreche et al. « OntoDBench : Interactively Benchmarking Ontology Storage in a Database ». In : *ER*. 2013, p. 499–503 (cf. p. 8).

[20] J.-M. Jézéquel, O. Barais et F. Fleurey. « Model driven language engineering with kermeta ». In : *International Summer School on Generative and Transformational Techniques in Software Engineering*. Springer. 2009, p. 201–221 (cf. p. 13).

[21] L. Jin, L. Liu et D. Guo. « Applying model-transformation techniques to ontology evolution ». In : *NODe 2005–GSEM 2005* (2005) (cf. p. 19).

[22] N. Kahani et al. « Survey and classification of model transformation tools ». In : *Software & Systems Modeling* 18.4 (2019), p. 2361–2397 (cf. p. 19).

[23] G. Kappel et al. « Model transformation by-example : a survey of the first wave ». In : *Conceptual modelling and its theoretical foundations*. Springer, 2012, p. 197–215 (cf. p. 18, 19).

[24] T. Kehrer, A. Alshanqiti et R. Heckel. « Automatic inference of rule-based specifications of complex in-place model transformations ». In : *International Conference on Theory and Practice of Model Transformations*. Springer. 2017, p. 92–107 (cf. p. 24).

[25] M. Kessentini, H. Sahraoui et M. Boukadoum. « Model transformation as an optimization problem ». In : *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2008, p. 159–173 (cf. p. 24).

[26] M. Kessentini et al. « Example-based sequence diagrams to colored petri nets transformation using heuristic search ». In : *European Conference on Modelling Foundations and Applications*. Springer. 2010, p. 156–172 (cf. p. 18).

[27] M. Kessentini et al. « Search-based model transformation by example ». In : *Software & Systems Modeling* 11.2 (2012), p. 209–226 (cf. p. 26).

[28] D. S. Kolovos, R. F. Paige et F. A. Polack. « The epsilon transformation language ». In : *International Conference on Theory and Practice of Model Transformations*. Springer. 2008, p. 46–60 (cf. p. 12).

[29] P. Langer, M. Wimmer et G. Kappel. « Model-to-model transformations by demonstration ». In : *International Conference on Theory and Practice of Model Transformations*. Springer. 2010, p. 153–167 (cf. p. 24).

[30] K. Lano et al. « Enhancing model transformation synthesis using natural language processing ». In : *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems : Companion Proceedings*. 2020, p. 1–10 (cf. p. 20).

[31] A. Likas, N. Vlassis et J. J. Verbeek. « The global k-means clustering algorithm ». In : *Pattern recognition* 36.2 (2003), p. 451–461 (cf. p. 25).

[32] M. W. Mkaouer et M. Kessentini. « Model transformation using multiobjective optimization ». In : *Advances in Computers*. T. 92. Elsevier, 2014, p. 161–202 (cf. p. 22).

[33] C. Pahl. « Ontology transformation and reasoning for model-driven architecture ». In : *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2005, p. 1170–1187 (cf. p. 19).

[34] J. Pavón et J. Gómez-Sanz. « Agent oriented software engineering with INGENIAS ». In : *International Central and Eastern European Conference on Multi-Agent Systems*. Springer. 2003, p. 394–403 (cf. p. 25).

[35] J. Pavón, J. Gómez-Sanz et R. Fuentes. « Model driven development of multi-agent systems ». In : *European Conference on Model Driven Architecture-Foundations and Applications*. Springer. 2006, p. 284–298 (cf. p. 26).

[36] A. Perini et A. Susi. « Automating model transformations in agent-oriented modelling ». In : *International Workshop on Agent-Oriented Software Engineering*. Springer. 2005, p. 167–178 (cf. p. 26).

[37] S. Roser et B. Bauer. « Ontology-based model transformation ». In : *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2005, p. 355–356 (cf. p. 19).

[38] Y. Sun, J. White et J. Gray. « Model transformation by demonstration ». In : *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2009, p. 712–726 (cf. p. 10).

[39] G. Taentzer. « AGG : A graph transformation environment for modeling and validation of software ». In : *International Workshop on Applications of Graph Transformations with Industrial Relevance*. Springer. 2003, p. 446–453 (cf. p. 16).