République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Larbi Tébessi -Tébessa-
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département : Mathématiques et Informatique

MEMOIRE DE MASTER
Domaine : Mathématiques et Informatique
Filière : Informatique
Option : Systèmes d'information

Thème

Co-évolution modèles / méta-modèles

Présenté par :

Lynda BEZAZ

Rebea HABEL

Devant le jury:

Mohammed Ridda LAOUAR	Professeur	Université Larbi Tébessi	***Président***
Samir KHEDIRI	MAA	Université Larbi Tébessi	***Examinateur***
Mohammed Yassine HAOUAM	MCB	Université Larbi Tébessi	***Encadreur***

Promotion : 2017

RESUME

Le développement de logiciels témoigne du besoin croissant de techniques de gestion pour soutenir l'évolution des artefacts basés sur des modèles. À cet égard, les méta-modèles peuvent être considérés comme l'un des concepts de base de l'ingénierie dirigée par les modèles et devraient évoluer au cours de leur cycle de vie. En conséquence, les modèles non conformes aux méta-modèles modifiés doivent être mis à jour pour préserver leur conformité.

Dans ce mémoire, nous introduisons un problème de co-évolution, une multitude d'approches pour la co-évolution du modèle / méta-modèle ont été proposées au cours des dernières années. Cependant, le nombre élevé de solutions rend difficile pour les praticiens de choisir une approche appropriée. Pour remédier à ce risque nous donnons un aperçu des différentes approches de co-évolution en mettant l'accent sur la co-évolution des modèle / méta-modèle et les comparons à base de certain critères.

Mots clés : Modèles, méta-modèles, ingénierie dirigée par les modèles, évolution, co-évolution, approches co-évolution.

ABSTRACT

Software development demonstrates the growing need for management techniques to support the evolution of model based artefacts. In this respect, meta-models can be considered as one of the basic concepts of model driven engineering and should evolve over their life cycle. As a result, models no conformant to modified meta-models must be updated to preserve their conformity.

In this master's memory, we introduce a problem of co-evolution, a variety of approaches for the model / meta-model co-evolution has been proposed in recent years. However, the high number of solutions makes it difficult for practitioners to choose an appropriate approach. To remedy this risk we give an overview of the different approaches of co-evolution by focusing on the model / meta-model co-evolution and comparing them based on certain criteria.

Keywords : Models, meta-models, model driven engineering, evolution, co-evolution, co-evolution approaches.

ملخص

يوضح تطور البرمجيات الحاجة المتزايدة لتقنيات التسيير لدعم تطور الحوادث المصطنعة القائمة على النماذج. وفي هذا الصدد، يمكن اعتبار النماذج الفوقية (الميتامودل) من بين المفاهيم الأساسية للهندسة الموجهة بواسطة النماذج والتي يجب أن تتطور على مدى دورة حياتها. و بالتالي، يجب تحديث النماذج لتكون مطابقة للنماذج الفوقية المعدلة.

و في هذه المذكرة، نعرض مشكلة التطور المشترك (كوإفوليسيون)، حيث اقترحت في السنوات الأخيرة مجموعة متنوعة من منهجيات التطور المشترك للنماذج / النماذج الفوقية. غير أن هذا العدد الكبير من الحلول يجعل من الصعب على مستعمليها اختيار المنهجية المناسبة. ومن أجل معالجة هذه المخاطر، نقدم لمحة عامة عن المنهجيات المختلفة للتطور المشترك من خلال التركيز على التطور المشترك للنماذج / النماذج الفوقية ومقارنتها استنادا إلى معايير محددة.

الكلمات المفاتيح : النماذج، النماذج الفوقية، الهندسة الموجهة بواسطة النماذج، التطور، التطور المشترك، منهجيات التطور المشترك.

Dédicace

Je dédie ce mémoire

A mes chers parent ma **mère** et mon **père**,

Pour leur patience, leur amour, leur soutien et leurs encouragements,

Aucun hommage ne pourrait être à la hauteur de l'amour Dont ils ne cessent de me combler. Que dieu leur procure bonne santé et longue vie,

A la mémoire de mon cher frère « **Djamel** », qui est décédé le **17/05/2017**.que Dieu vous pardonne et vous fait des gens du Paradis,

A mes frères et leurs épouses,

A mes sœurs et leurs époux,

A mes nièces et neveux,

A mon binôme et chère amie ma belle **Rabi3a**,

A mes chers amies **AHlem H, Nadjette D, Mouchira B** et **Assia Ch**, et mes amis et mes camarades,

A mes collègues de promotion de Master **2** Système d'Information **2017** de l'université Larbi Tébessi –Tébessa,

A notre encadreur **Mr Mohammed Yassine Haouam** pour sa patience, sa rigueur et sa disponibilité durant notre préparation de ce mémoire,

A tous ceux qui me sont chers.

LyNDa

DEDICACE

Je dédie ce modeste travail

À

Mes parents

Tous mes frères, et leurs enfants

Toutes mes sœurs, et leurs enfants

Mon fiancé et sa famille

Tous mes amis

Tous mes collègues de travail

Tous ceux qui m'ont aidée à réaliser ce mémoire de prés ou de loin.

Rebea

REMERCIEMENTS

On ne sait pas s'il existe des statistiques sur le sujet, mais il nous semble que la page la plus lue dans un mémoire est celle des remerciements. La mission est donc difficile. En espérant qu'il ne soit pas trop rébarbatif.

Nous tenons tout d'abord à remercier **ALLAH** le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce Modeste travail.

La construction de ce mémoire n'aurait été possible sans l'intervention de certaines personnes. Qu'elles trouvent ici l'expression de nos plus sincères remerciements pour leurs précieux conseils.

Nous tenons à exprimer toutes nos gratitudes à ***M. Mohammed Yassine HAOUAM*** qui nous a permis de bénéficier de son encadrement, nous le remercie pour l'orientation, la confiance, l'aide, les conseils, la patience qui ont constitué un apport considérable sans lequel ce travail n'aurait pas pu être mené au bon port. Qu'il trouve dans ce travail un hommage vivant à sa haute personnalité.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail et de l'enrichir par leurs propositions

Nous adressons nos sincères remerciements à l'administration et à l'ensemble du corps enseignant de l'**Université Larbi Tébessi *Tébessa*** pour leurs efforts à nous garantir la continuité et l'aboutissement de ce programme de Master.

Nous remercions nos **très cher parents**, qui ont toujours été là pour nous, *« Vous avez tout sacrifié pour vos enfants n'épargnant ni santé ni efforts. Vous nous avez donné un magnifique modèle de labeur et de persévérance. Nous sommes redevable d'une éducation dont nous sommes fier ».*

A nos familles et nos amis qui par leurs prières et leurs encouragements, on a pu surmonter tous les obstacles.

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

TABLE DES MATIERES

RESUME…….... 01
DEDICACE ……..……..... 04
REMERCIEMENT …… 06
LISTE DES FIGURES…… 11
LISTE DES TABLEAUX……... 12
INTRODUCTION GENERALE…… 13
CHAPITRE I : L'ingénierie dirigée par les modèles……... 15
INTRODUCTION……..…... 15
1. L'ingénierie dirigée par les modèles (IDM)…… 16
1.1. Modèle……... 16
1.2. Méta-modèle……. 17
1.3. Les artefacts de l'IDM…….. 18
1.4. Situations de modélisation…… 19
1.4.1. La concrétisation…… 20
1.4.2. L'abstraction…….... 21
1.4.3. La multi-abstraction…….. 21
1.4.4. La co-abstraction……... 21
1.4.5. La multi-concrétisation…….. 21
1.4.6. La co-concrétisation…….. 22
1.4.7. L'interprétation…….... 22
1.4.8. La co-interprétation……... 22
1.4.9. L'exemplification/L'extension……..…… 23
1.4.10. L'évolution/L'extension……..…… 23
1.4.11. La co-évolution…… 23
1.5. Avantages et inconvénients de l'IDM…….. 24
1.5.1. Avantages de l'IDM……. 24
1.5.2. Inconvénients de l'IDM…… 24
1.6. Défis de l'évolution de l'IDM…….. 25
1.6.1. L'évolutivité……..... 25
1.6.2. La gestion de l'automatisation……... 26
1.6.3. L'hétérogénéité des dépendances……..... 26
1.6.4. L'études empiriques……..... 26
1.6.5. L'utilité des outils…….. 27
1.6.6. La syntaxe concrète……... 27
1.6.7. Les temps d'exécution…..…….... 27
1.6.8. Les interfaces externes……..……. 28
1.6.9. La sémantique linguistique…… 28
1.6.10. Les méta-modèles……..…….. 29

1.7. Principales caractéristiques des solutions d'évolution dans l'IDM…….. 29
1.7.1. Le champ d'application…….. 29
1.7.2. L'automatisation…… 30
1.7.3. L'environnement…… 30
1.7.4. La conformité…… 30
2. L'architecture dirigée par les modèles (MDA)…… 30
2.1. Principes du MDA…… 31
2.2. Système et environnement du MDA…… 31
2.3. Normalisation……... 31
2.4. Niveaux d'abstraction du MDA…… 33
2.4.1. Le Modèle indépendant du calcul «The Computation-Independent Model» (CIM)…… 34
2.4.2. Le modèle indépendant de la plateforme «The Platform-Independent Model» (PIM)..... 35
2.4.3. Le modèle spécifique à la plateforme «The Platform-Specific Model» (PSM)……... 35
2.5. Les transformations……..... 35
2.5.1. Définition de la transformation des modèles……..... 36
2.5.2. Classification des langages de transformations des modèles…….…….. 36
2.6. Contraintes…… 37
CONCLUSION…….…….. 38
CHAPITRE II : Co-évolution modèles/métamodèles…….…… 39
INTRODUCTION…… 39
1. Types de modifications effectuées sur les méta-modèles…… 40
1.1. Les modifications atomiques……. 40
1.2. Les modifications complexes…… 40
2. Types d'impacts de modifications effectuées sur les méta-modèles…….. 40
2.1. Les modifications sans effet de bord…… 40
2.1.1. Généraliser une méta-propriété « Generalize meta-property »……. 40
2.1.2. Ajouter des méta-classes (non obligatoires) « Add (non-obligatory) meta-class »……. 40
2.1.3. Ajouter des méta-propriétés (non obligatoire) «Add (non-obligatory) meta-property»… 41
2.2. Les modifications avec effet de bord et soluble…… 41
2.2.1. Extraire une super-classe (abstraite) «Extract (abstract) super-class»……. 41
2.2.2. Supprimer des méta-classes «Eliminate meta-class»……. 41
2.2.3. Suppression des méta-propriétés « Eliminate meta-property »……. 41
2.2.4. Push méta-propriété «Push meta-property »……..... 41
2.2.5. Flatten hierarchy……. 41
2.2.6. Renommer des méta-éléments « Rename meta-element »……. 41
2.2.7. Déplacer une méta-propriété « Move meta-property »……..... 42
2.2.8. Extract/Inline meta-class…… 42
2.3. Les modifications avec effet de bord et non soluble …… 42
2.3.1. Ajouter des méta-classes (obligatoire) « Add (obligatory) meta-class »…… 42
2.3.2. Ajouter des méta-propriétés (obligatoire) « Add (obligatory) meta-property »……. 42
2.3.3. Tirer une méta-propriété « Pull meta-property »……... 42
2.3.4. Restriction d'une méta-propriété « Restrict meta-property »……. 42
2.3.5. Extrait (non-abstraite) super-classe « Extract (non-abstract) super-class »……... 43
3. Collecte des modifications du méta-modèle…… 43
3.1. Collecte des modifications hors ligne……..... 43
3.2. Collecte des modifications en ligne……..... 43
3.2.1. Les approches de préservation de l'interface utilisateur (UI-preserving approaches)…… 44
3.2.2. Les approches intrusives de l'interface utilisateur (UI-intrusive approaches)…….. 44
4. Co-évolution des méta-modèles……... 44

4.1. Co-évolution modèle de transformation (opération) /méta-modèle……..... 45
4.1.1. Étape de détection……..... 45
4.1.2. Étape de co-évolution……. 45
4.1.3. Aperçu des approches de co-évolution opérations / méta-modèles……... 45
4.2. Co-évolution contraintes/méta-modèles…… 46
4.3. Aperçu des autres approches de co-évolution…….. 47
4.3.1. L'ingénierie Round-trip……. 47
4.3.2. L'évolution du modèle en tant que problème de transformation…… 47
4.3.3. Evolution du schéma……..... 48
4.3.4. Versionnement du modèle (Model Versioning)…… 48
4.4. Identification des changements de méta-modèle……...…….. 49
4.4.1. Identification des changements atomiques……. 49
4.4.2. Identification de changement complexe……. 50
4.4.3. Degré d'automatisation……... 51
4.5. Résolution de modèles……..... 51
4.5.1. Ordre de résolution des modèles ……...…… 52
4.5.2. Catégories de spécification des stratégies de résolution des modèles…….……. 52
4.5.3. Solutions prises en charge……..... 53
5. Co-évolution des modèle / méta-modèle……. 54
5.1. Définitions……. 54
5.2. Théorie de la co-évolution modèle / méta-modèle……..... 54
5.3. Modèles de co-évolution modèle/méta-modèle…… 55
5.4. Caractéristiques du cadre de modélisation…… 55
5.4.1. Séparation modèle / méta-modèle……..... 55
5.4.2. Conformité implicite……..... 56
5.5. Catégorisation des approches de co-évolution modèles / méta-modèles……..... 57
5.5.1. Les approches de spécification manuelle…….. 57
5.5.2. Les approches d'opérateurs…… 59
5.5.3. Les approches d'inférence (Meta-Model Matching)…….. 62
5.5.4. Les approches hybrides……..... 66
5.6. Exemple de co-évolution modèles/méta-modèles……. 66
CONCLUSION…… 68
CHAPITRE III : Étude comparative des approches de Co-évolution modèles/méta-modèles… 69
INTRODUCTION…… 69
1. Approches identifiées…….. 70
2. Critères de comparaison……... 70
2.1. Collection des modifications……. 71
2.2. Identification des modifications de méta-modèles…… 71
2.3. Résolution de modèles……..... 71
2.4. Correction des résultats requis……..... 71
2.5. Automatisation……. 71
2.6. Outils et mis en œuvre des approches de co-évolution modèles / méta-modèles……. 71
2.7. Contrôle du processus de co-évolution……. 72
2.8. Compétences de la personne qui change le méta-modèle…… 72
2.9. Compétences de la personne qui co-évolue les modèles……. 72
3. Processus de comparaison…… 72
3.1. Collection des modifications……. 73
3.2. Identification des modifications de méta-modèles…… 74
3.2.1. Identification des modifications atomiques…… 74

3.2.2. Identification des modifications complexes avec la collection hors ligne……. 74
3.2.3 Identification des modifications complexes avec la collecte en ligne…… 75
3.3. Résolution de modèles…….... 76
3.3.1. Ordre de résolution des modèles…..…….. 76
3.3.2. La catégorie de spécification des stratégies de résolution des modèles 0: n…… 77
3.3.3. La catégorie de spécification des stratégies résolution des modèles 1: n……. 79
3.3.4. La catégorie de spécification des stratégies résolution des modèles 1: 1……. 80
3.4. Correction des résultats requis……..... 80
3.4.1. Exactitude des résultats requis…….. 81
3.4.2. Opportunités de généralisation de la résolution…… 81
3.5. Automatisation……..... 82
3.5.1. Soutien d'automatisation des tâches manuelles…… 82
3.6. Outils et mise en œuvre des approches de co-évolution modèles / méta-modèles…… 83
3.6.1. Intégration des outils des approches de co-évolutions modèles / méta-modèles…….. 84
3.6.2. Prise en charge des outils et risque de première mise en œuvre…… 85
3.7. Contrôle du processus de co-évolution……. 85
3.8. Compétences de la personne qui change le méta-modèle…… 85
3.9. Compétences de la personne qui co-évolue les modèles……. 87
CONCLUSION……..... 88
CONCLUSION GENERALE…….. 89
LISTE DES ACRONYMES ……..……..... 91
REFERENCES…… 93

LISTE DES FIGURES

Figure I.1: Méta-modèle et modèle……. 17

Figure I.2 : Des situations de modélisation (haut : niveau d'instance, et bas : niveau de méta)..... 19

Figure I.3 : Les trois niveaux d'abstraction définie par MDA……. 34

Figure I.4 : Transformation du modèle Homogène……..…….…... 37

Figure I.5 : Transformation du modèle hétérogène…….…..……... 37

Figue II.1 : Mode de collection des modifications de méta-modèle…… 43

Figure II.2 : Types et caractéristiques pour identifier les changements de méta-modèle…… 50

Figure II.3 : Fonctionnalités pour la résolution de modèles……...……. 50

Figure II.4 : Les activités de co-évolution……... 56

Figure II.5 : Processus des approches d'inférence……... 64

Figure. II.6 : Exemple d'évolution d'un méta-modèle simplifié …….. 67

Figure. II.7 : Exemple de co-évolution du modèle…….. 67

Figure III.1 : Exactitude des résultats requis…… 81

Figure III.2 : Automatisation et assistance pour les tâches manuelles……. 83

Figure III.3 : Support d'outils……... 84

Figure III.4 : Contrôle du processus de co-évolution des modèles/méta-modèles…….. 86

Figure III.5 : Compétences de la personne qui change le méta-modèle…….. 86

Figure III.6 : Compétences de la personne qui co-évolue les modèles…… 87

LISTE DES TABLEAUX

Tableau III.1 : Les approches de co-évolution modèles/méta-modèles identifiées, et indication de leurs (année de proposition, catégorie) ……..…….….... 70

Tableau III.2 : La collection des modifications de méta-modèle (hors ligne/en ligne) dans les approches de co-évolution modèles/méta-modèles identifiées .....…. 73

Tableau III.3 : Identification des modifications de méta-modèle (atomique et complexe) dans les approches de co-évolution modèles/méta-modèles identifiées ……...……..... 75

Tableau III.4 : Ordre de résolution des modèles dans les approches de co-évolution modèles / méta-modèles identifiées ……....……. 76

Tableau III.5 : Résolution des modèles dans la catégorie 0:n pour chaque approches de co-évolution modèles / méta-modèles identifiées ……..... 78

Tableau III.6 : Résolution des modèles dans la catégorie 1:n pour chaque approches de co-évolution modèles / méta-modèles identifiées ……....… 79

Tableau III.7 : Résolution des modèles dans la catégorie 1:1 pour chaque approches de co-évolution modèles / méta-modèles identifiées …… 80

INTRODUCTION GENERALE

La modélisation est essentielle à l'activité humaine car chaque action est précédée par la construction (implicite ou explicite) d'un modèle. Il ya beaucoup d'usages pratiques des modèles. En particulier en informatique où les modèles logiciels sont construits, vise à améliorer la productivité, la qualité et la rentabilité du développement de logiciels en transférant le paradigme des activités centrées sur le code vers le modèle. La méta-modélisation est devenue la technologie clé pour définir des langages de modélisation de domaines spécifiques pour l'ingénierie dirigée par modèles IDM [31].

Au cours des dernières années, l'IDM a joué un rôle de premier plan dans l'avancement d'un nouveau changement de paradigme dans le développement de logiciels. L'IDM est de plus en plus émerger comme une discipline qui prescrit strictement les concepteurs de développer des logiciels en termes de modèles plutôt que des programmes. Selon cette perspective, les modèles sont exploités à un statut de première classe. L'évolution est inévitable et affecte tout le cycle de vie du logiciel. De manière analogue au logiciel, les méta-modèles sont également soumis à une pression évolutive. Cependant, le changement d'un méta-modèle pourrait compromettre les artefacts apparentés, dont la validité doit être rétablie [30].

Les principaux artefacts de l'IDM sont: *les modèles*, *les méta-modèles* et *les transformations*. Alors que le modèle et l'évolution de la transformation peuvent être confrontés dans l'isolement, l'évolution du méta-modèle influe sur les modèles et les transformations. Les changements de méta-modèle peuvent avoir des conséquences inquiétantes sur leurs modèles d'instance et briser les transformations associées [53].

Les modèles et les méta-modèles subissent des évolutions complexes pendant leurs cycles de vie. En conséquence, lorsqu'un méta-modèle est modifié, les modèles conformes à ce méta-modèle doivent être migrés de manière à ce qu'ils soient conformes à la version modifiée. On dit qu'un modèle est conforme à un méta-modèle, ou en d'autres termes, il est exprimé par les concepts codés dans le méta-modèle, les contraintes sont exprimées au niveau de la méta-modèle et la transformation du modèle se produit lorsqu'un modèle source est modifié pour produire un modèle cible. Dans la littérature, ce problème est appelé ***évolution du méta-modèle*** ou ***co-évolution de modèle*** [87].

La co-évolution des méta-modèles a déjà été identifiée comme un problème principal dans la littérature. L'évolution entre deux versions du même méta-modèle peut consister en des centaines d'ajouts, de suppressions et de changements d'éléments. Par exemple, pendant l'évolution des diagrammes de classe UML de UML version 1.5 à 2.0, un total de 238 éléments du langage ajoutés, supprimés ou modifiés. Pour l'évolution de GMF version 1.0 à 2.0, 136 changements ont été identifiés par Herrmannsdoerfer et Al [31].

Récemment, plusieurs approches ont été proposées pour résoudre le problème de la co-évolution. Principalement, en se concentrant sur la co-évolution du modèle et méta-modèle (c'est-à-dire la migration du modèle). La migration du modèle est une activité cruciale et elle est intrinsèquement complexe et se traduit par un processus fastidieux qui prend beaucoup de temps et d'erreurs si aucun soutien adéquat n'est fourni. Dans cette étude, nous discutons de l'état de l'art dans les approches de la co-évolution du modèle et méta-modèle mettant en évidence leurs forces et leurs faiblesses, puis nous comparons un ensemble sélectionné des approches décrites en utilisant des critères généraux que nous jugeons importants pour l’opération de co-évolution [53].

Le reste de ce mémoire est organisé comme suit, dans le premier chapitre on parle sur les notions de base de l’IDM ainsi leurs défi, le deuxième chapitre met l’accent sur le concept de la co-évolution dans l’IDM on se base sur les concepts de base liés à la co-évolution du modèle/méta-modèle, aussi dans ce chapitre, nous présentons un aperçu des approches de co-évolution modèle/ méta-modèle avec leur catégorisation. Dans le dernier chapitre, une comparaison des approches selon des critères généraux a été faite. Enfin, on présente notre conclusion.

CHAPITRE I

L’ingénierie dirigée par les modèles

INTRODUCTION

Au cours des dernières années, les ingénieurs logiciels et informaticiens ont intégré l'utilisation de modèles et de langages de modélisation dans les processus de développement. Alors que les modèles sont largement utilisés en génie logiciel pour décrire les exigences, l'architecture et les détails d'implémentation du projet logiciel. Ils sont utilisés pour permettre la manipulation et le raisonnement sur les systèmes à un niveau plus abstrait et donc gérer la complexité. La même chose se fait dans d'autres disciplines techniques pour fournir des moyens de planification, de validation ou d'analyse de systèmes. Les modèles sont utilisés par exemple pour la génération de code, pour dériver d'autres artefacts ou comme documentation. Par exemple : les diagrammes d’***UML*** (***U****nified* ***M****odeling* ***L****anguage*) peuvent être utilisés pour la documentation d'un système [72].

Mais aujourd'hui les modèles sont également utilisés comme entrée pour les générateurs de code. Cette discipline dans l’informatique est appelée ***IDM*** (***I****ngénierie* ***D****irigée par les* ***M****odèles*), en anglais ***MDE*** (***M****odel* ***D****riven* ***E****ngineering*) ou ***MDD*** (***M****odel* ***D****riven* ***D****evelopment*) ou bien ***MDSD*** (***M****odel* ***D****riven* ***S****oftware* ***D****evelopment*) [71]. Dans ce mémoire, nous adoptons "l’IDM" comme terme pour désigner cette discipline.

1. L'ingénierie dirigée par les modèles (IDM)

L'IDM est une approche de développement de logiciels basée sur des modèles de domaine. Ces modèles sont créés sur le plan conceptuel, donc ils sont indépendants de la mise en œuvre finale. Après leur validation, ils peuvent être transformés en niveaux plus concrets.

Cette validation très précoce vise à trouver des erreurs dans la conception de la solution dès que possible, réduisant ainsi les coûts (les défauts coûtent moins chers s'ils sont détectés tôt).

L'IDM est une méthodologie abstraite, de sorte que des initiatives ont été créées à partir d'elle pour essayer de formaliser les méthodes et les principes de cette nouvelle méthodologie. L'architecture dirigée par les modèles, en anglais ***MDA*** (***M****odel* ***D****riven* ***A****rchitecture*) est l'une des plus connues, dont nous parlerons dans ce mémoire [81].

1.1. Modèle

Les modèles sont fondamentaux pour l'IDM. **Kurtev** identifie nombreuses définitions du terme modèle (Kurtev en 2004), y compris ce qui suit:

« Any subject using a system A that is neither directly nor indirectly interacting with a system B to obtain information about the system B, is using A as a model for B » (Apostel en 1960), signifie que : "*Tout sujet utilisant un système* ***A*** *qui n'est ni directement ni indirectement interagissant avec un système* ***B*** *pour obtenir des informations sur le système* ***B****, utilise* ***A*** *comme modèle pour* ***B*** ".

« ***A model is a representation of a concept. The representation is purposeful and used to abstract from reality the irrelevant details*** » (Starfield et Al en 1990), signifie que : "*Un modèle est une représentation d'un concept. La représentation est résolue et utilisée pour abstraire de la réalité les détails non pertinents*".

« ***A model is a simplification of a system written in a well-defined language*** » (Bezivin et Gerbe en 2001), signifie que : "*Un modèle est une simplification d'un système écrit dans une langue bien définie* " [75].

Les modèles représentent un aspect spécifique du système d'une manière abstraite et souvent formalisée, ils peuvent être utilisés pour analyser les propriétés du système avant une mise en œuvre complète, ce qui permet aux développeurs de prendre des décisions éclairées au lieu de s'appuyer sur l'expérience ou l'intuition.

En outre, les modèles peuvent être utilisés pour dériver d'autres artefacts qui sont utilisés dans le processus de développement. Par exemple : des représentations graphiques de

systèmes logiciels peuvent être utilisées pour générer de la documentation, ou des codes peuvent être générés à partir d'un schéma architectural d'un système. Les programmes qui mettent en œuvre ces générations ont des modèles d'entrée et de sortie et sont appelés transformations de modèles. Ils sont écrits dans des langages de transformation de modèle.

Dans l'ingénierie dirigée par les modèles, chaque artefact qui décrit un système est considéré comme un modèle, y compris des documents de code et de langue naturelle. En principe, tous les modèles peuvent être modifiés pendant le processus de conception et de développement, pour arriver finalement à un produit fini [84].

1.2. Méta-modèle

Dans l'IDM, la méta-modélisation est une technique pour définir un langage de modélisation de modèle. Par conséquent, un méta-modèle est un modèle du langage de modélisation.

La définition précise du terme méta-modèle est fréquemment débattue dans la littérature. En particulier, il n'y a pas un accord commun quant aux artefacts appartiennent à un méta-modèle. Dans ce mémoire, un méta-modèle est un modèle à base de graphes définissant une syntaxe de langage de modélisation, ce qui limite l'ensemble des modèles valides (voir Figure I.1) [2].

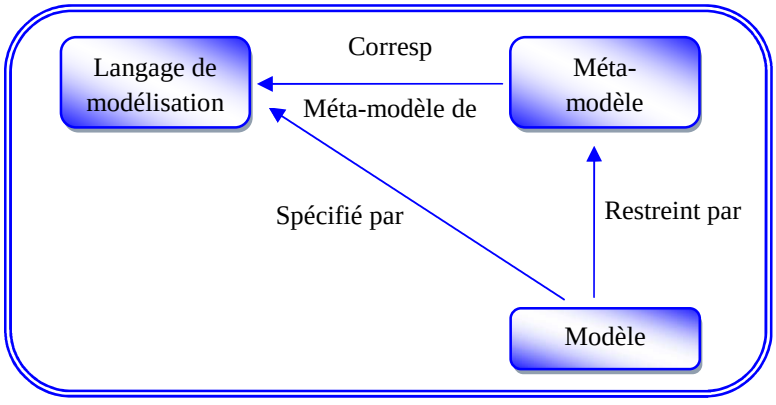


Figure I.1: *Méta-modèle et modèle [2]*

Un méta-modèle se compose des éléments suivants :

- ✓ Une syntaxe abstraite décrivant les éléments dont un modèle valide peut se composer indépendamment de la représentation.
- ✓ Au moins une syntaxe concrète qui spécifie comment exprimer un modèle, par exemple dans un langage textuel ou sous forme de diagramme graphique.

- ✓ La sémantique statique qui décrit davantage la forme des instances valides et ne contient pas d'informations sur l'utilisation ou la transformation des instances du modèle. La sémantique statique peut par exemple être exprimée sous forme de contraintes.

Un modèle peut se conformer à un méta-modèle. Nous disons également que le modèle est une instance d'un méta-modèle de façon interchangeable [84].

1.3. Les artefacts de l'IDM

Dans l'IDM, trois principaux artefacts sont utilisés : les ***méta-modèles***, les ***modèles*** et les ***opérations***. Elle inclut la construction de modèles, par exemple : des exigences, des architectures, des conceptions, du code, des tests, etc. Et l'application d'opérations, par exemple : transformation, comparaison, fusion, etc., sur ces modèles afin d'automatiser des parties du processus d'ingénierie.

Il y a plusieurs observations critiques à ce sujet:

- ✓ Tous les artefacts sont interdépendants : les modèles sont conformes aux méta-modèles, les méta-modèles eux-mêmes sont conformes aux méta-méta-modèles, les opérations sont définies en termes de méta-modèles et s'appliquent aux modèles et ces interrelations sont formelles, en ce sens il existe des spécifications précises, par exemple : la conformité du modèle.
- ✓ Les processus IDM sont formels, dans le sens où des opérations formellement définies peuvent être utilisées pour implémenter des parties importantes entre ces processus.
- ✓ L'hétérogénéité est inhérente à ces artefacts. L'IDM utilise généralement différents langages "méta-modèles", différents modèles, différents types de fonctionnement, par exemple : les transformations de modèle à modèle, les transformations de modèle à texte, et même différentes technologies de persistance de modèle.
- ✓ Certains changements de méta-modèle qui pourraient se produire et ne pouvaient pas être traités automatiquement. Cela suggère que pour la gestion des changements dans l'IDM, le jugement sera toujours nécessaire, soit pour choisir l'approche la plus appropriée pour le problème en question, soit pour choisir parmi un ensemble de solutions potentielles de gestion des changements.

Bien sûr, tous ces artefacts peuvent avoir besoin de changer, et certains changements peuvent être plus difficiles à gérer que d'autres. Les modifications apportées aux modèles peuvent de manière inhérente faire partie du processus d'ingénierie et peuvent être effectuées par un ingénieur. Les modifications apportées aux opérations peuvent être appuyées par un expert du l'IDM chargé d'écrire et de maintenir l'opération. Les modifications apportées aux méta-modèles peuvent être analysées et exécutées par un ingénieur linguistique. Toute modification d'un artefact peut entraîner des modifications d'autres artefacts. C'est notamment le cas des modifications apportées aux méta-modèles, qui peuvent nécessiter des modifications aux opérations qui utilisent ces méta-modèles et des modifications à tous les modèles conformes à la version précédente du méta-modèle [78].

1.4. Situations de modélisation

Pour décrire les situations de modélisation, nous avons deux types d'artefacts à considérer : les modèles et les méta-modèles. Les situations varient selon l'ordre dans lequel ces artefacts apparaissent ou sont reliés dans la démarche. On simplifie la description en ne considérant qu'un seul acteur dans chaque situation. Une analyse plus fine de ces situations pourraient être intéressante en prenant en compte différents acteurs et donc différentes intentions dans le processus. La figure I.2 présente les 11 situations considérées.

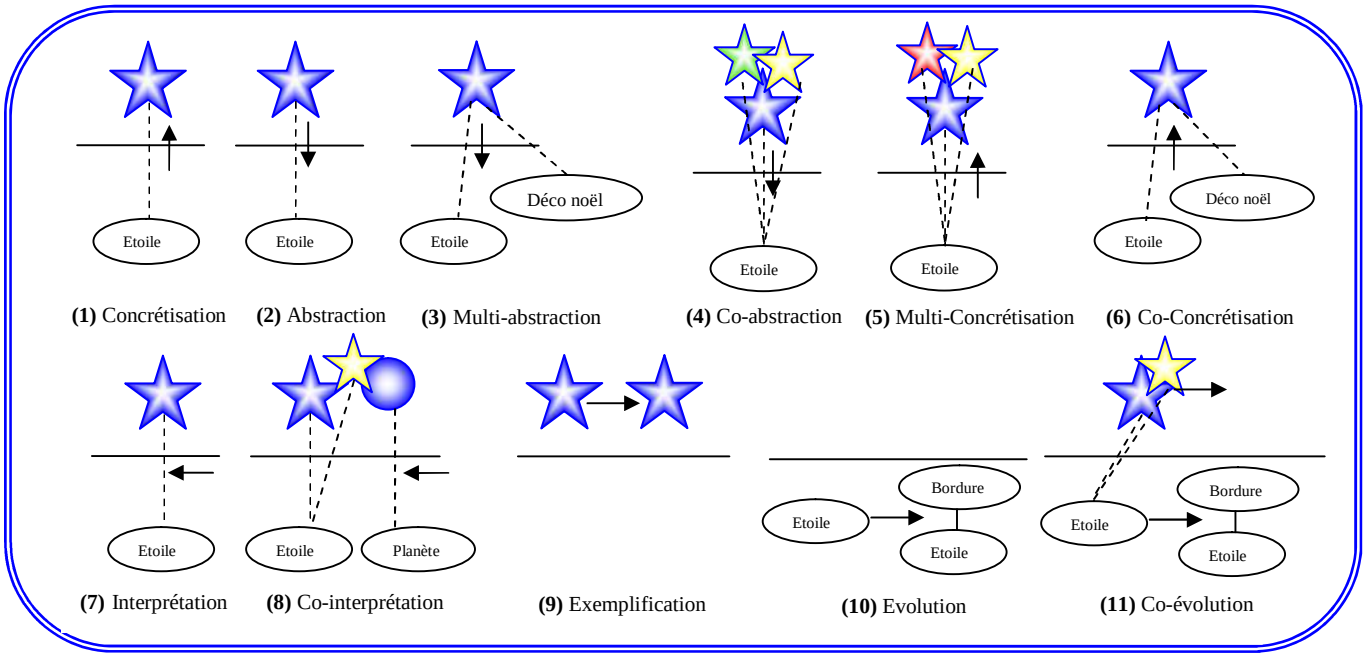


Figure I.2 : *Des situations de modélisation (haut : niveau d'instance, et bas : niveau de méta) [7]*

- ✓ (1) « *concrétisation* » : Un méta-modèle existe, on cherche à produire un modèle.
- ✓ (2) « *abstraction* » : Un modèle existe, le travail consiste à trouver un méta-modèle.
- ✓ (3) « *multi-abstraction* » : Un modèle existe, il faut trouver plusieurs méta-modèles.
- ✓ (4) *« co-abstraction » :* Des modèles existent, il faut élaborer un méta-modèle.
- ✓ (5) *« multi-concrétisation » :* Un méta-modèle existe, le travail consiste à construire plusieurs modèles.
- ✓ (6) *« co-concrétisation » :* Des méta-modèles existent, le travail consiste à construire un modèle.
- ✓ (7) *« interprétation » :* Un modèle et un méta-modèle existent, il faut les relier.
- ✓ (8) *« co-interprétation » :* Des modèles existent, des méta-modèles existent, le travail consiste à les relier.
- ✓ (9) « *exemplification*/*extension* » : Un modèle existe, le travail consiste à construire un autre modèle (sans aucun méta-modèle).
- ✓ (10) « *évolution*/*extension* » : Un méta-modèle existe, le travail consiste à construire un autre méta-modèle (sans aucun modèle).
- ✓ (11) « *co-évolution* » : Un méta-modèle existe avec plusieurs de ses modèles conformes, le travail consiste à faire évoluer le méta-modèle (cas précédent) en adaptant (ou non) ses modèles.

Les deux cas (9) et (10) sont probablement équivalents dans le cadre d'une interprétation multi-niveaux de la modélisation. En effet, à un niveau donné d'abstraction, l'absence de référence à un autre niveau, plus concret ou plus abstrait, rend le travail équivalent. On les différencie tout de même car la plupart des outils proposent des moyens de manipulations de ces deux niveaux très différents, liés à leur mode de représentation [7].

1.4.1. La concrétisation

La concrétisation est la forme d'utilisation la plus classique d'un outil de modélisation. Les concepteurs de l'outil ont préparé des éditeurs adaptés à un ensemble de concepts (méta-modèle). Les utilisateurs élaborent des modèles en respectant les règles prévues. Cette approche est efficace lorsque le modèle que l'on souhaite construire est adapté au méta-modèle utilisé, c'est-à-dire les concepts nécessaires sont présents. Dans le cas où les concepts ne s'alignent pas exactement, les utilisateurs sont parfois amenés à tordre les interprétations ou imaginer des usages des concepts non prévus [82].

1.4.2. L'abstraction

Ce cas, très classique, également dans une démarche de modélisation consiste à partir d'un exemple à identifier les concepts et les règles pour construire un méta-modèle auquel le modèle initial est conforme. Le résultat est exploité par des développeurs d'outil pour construire des éditeurs de modèles [42].

1.4.3. La multi-abstraction

Ce cas, moins classique, généralise la situation d'abstraction, où le but est à la fois d'abstraire, et également d'organiser l'abstraction en faisant apparaitre des points de vue complémentaires sur le modèle. On ne différencie pas les méta-modèles, l'un peut préexister ou ils peuvent être conçus tous en même temps. L'important est qu'il existe plusieurs méta-modèles [43].

1.4.4. La co-abstraction

Cette pratique est une généralisation du deuxième cas (l'abstraction). Plusieurs modèles servent d'exemples pour construire un méta-modèle auxquels tous les exemples seront conformes. Les démarches taxonomiques ou de classification sont des situations de co-abstraction. L'intérêt d'utiliser plusieurs modèles exemples permet de confronter les concepts et de faire apparaitre des consensus ou au contraire des différences d'interprétation et des conflits conceptuels.

Les situations de co-abstraction et de multi-abstraction sont combinables en *une multi-co-abstraction* [21].

1.4.5. La multi-concrétisation

Ce cas est une généralisation du premier cas (concrétisation). Le même méta-modèle est utilisé à plusieurs reprises pour produire différents modèles. Le risque s'existe lorsque les interprétations ne sont pas les mêmes pour produire les modèles, des incohérences peuvent apparaitre sans être détectables. Le risque de cette approche est d'autant plus important que les interprétations du méta-modèle sont nombreuses comme c'est le cas avec UML et ses multiples points de variation sémantique. Par exemple : la sémantique des diagrammes d'états n'est pas définie précisément en UML et peut donner lieu à diverses interprétations [32].

1.4.6. La co-concrétisation

Cette situation se rencontre lorsque plusieurs experts, chacun disposant de son propre méta-modèle, se réunissent pour décrire un système dans lequel leurs points de vue doivent être représentés. Dans cette situation on souhaite laisser les abstractions séparées et ne pas construire un méta-modèle commun [48].

1.4.7. L'interprétation

Ce cas est moins habituel, participe sans aucun doute à la démarche de modélisation, au moins dans la tête des modélisateurs. Un modèle exemple existe ainsi qu'un méta-modèle. Cette situation est courante et s'appuie sur la connaissance des concepts et de leur organisation par un expert. Le travail consiste à relier les éléments du modèle exemple à un concept présent dans le méta-modèle dans le but :

- ✓ soit de s'assurer que l'exemple respecte les règles du méta-modèle.
- ✓ soit de trouver un contre-exemple "le modèle exemple" au méta-modèle afin de le faire évoluer.

Il faut noter que la relation d'interprétation n'est pas la relation d'instanciation. ***L'instanciation*** repose sur un mécanisme de construction d'une représentation particulière.

Un même concept peut être instancié de multiples manières. L'interprétation décrit une intention, celle de mettre en relation une représentation et son sens, défini et décrit dans un méta-modèle [7].

1.4.8. La co-interprétation

Ce cas est une généralisation du cas précédent. Plusieurs modèles exemples existent ainsi que plusieurs méta-modèles. Cette situation est encore plus courante que la précédente et s'appuie sur la connaissance des concepts et de leurs organisations par un expert. Le travail consiste à relier les éléments des modèles à un (ou plusieurs) concept(s) présent(s) dans les méta-modèles dans le but :

- ✓ soit de s'assurer que les exemples respectent les règles des méta-modèles.
- ✓ soit de trouver un contre-exemple aux méta-modèles afin de les faire évoluer [81].

1.4.9. L'exemplification/L'extension

Cette situation est souvent la première rencontrée. Il s'agit de produire des exemples de représentation qui servent de base à la réflexion. Pour des dessins ou des diagrammes, on choisira des couleurs, des formes, des positions relatives des formes pour représenter le problème. L'excellent article de **D. Moody** en 2009 passe en revue de nombreux exemples de représentations graphiques. Une histoire de représentation est présentée par **M. Friendly** en 2005, aussi par **Friendly et Denis** en 2001. Pour les langues, une variabilité extraordinaire des langues et grammaires est observable dans le dictionnaire des langues de **Peyraube et Al** en 2010 [7].

1.4.10. L'évolution/L'extension

Cette situation, parallèle à la précédente, ne peut être mise en place qu'une fois la conceptualisation est réalisée, c'est-à-dire lorsqu'un méta-modèle est disponible. Il s'agit de faire évoluer les concepts ou les règles les gouvernant. Ce travail se réalise souvent :

- ✓ explicitement ou non.
- ✓ en référence à des évolutions des modèles [7].

1.4.11. La co-évolution

C'est une situation liée à la précédente, où le méta-modèle évolué est était relié à des modèles existants, qu'advient-il des modèles existants ? L'article de ***Sprinkle et Karsai*** en 2004 est l'un des premiers à identifier le problème avec le vocabulaire de l'ingénierie dirigée par les modèles.

Ce cas est très complexe car il peut prendre de nombreuses formes. Par exemple : si un nouveau concept est introduit dans le méta-modèle sans qu'aucune instance de ce concept n'ait été précédemment créée, la solution est triviale. Par contre, si la structure d'un concept est remise en cause et que des instances avaient été créées, la solution pourrait ne pas avoir de solution automatisable et demander l'intervention d'un humain pour guider l'évolution [72].

1.5. Avantages et inconvénients de l'IDM

Certains des avantages et des inconvénients proclamés de l'utilisation de l'IDM.

1.5.1. Avantages d'IDM

Parmi les avantages de l'IDM, on trouve [40] :

- ***La portabilité et l'obsolescence de la technologie :*** le noyau du système est décrit d'une manière indépendante de la plate-forme cible, donc tous les changements de la plate-forme (que ce soit pour des raisons commerciales ou technologiques) est beaucoup plus facile.
- ***La productivité :*** à l'aide de transformations automatisées la nécessité d'une certaine programmation est éliminée et permet ainsi aux développeurs de mettre leurs efforts à des parties plus cruciales du système.
- ***La qualité :*** les artefacts produits à partir de modèles formellement définis sont fiables et contribuent à l'amélioration de la qualité.
- ***La diversité :*** l'IDM est embrassé par diverses organisations et sociétés, Il existe également une grande variété d'outils disponibles, tels que le cadre ***Eclipse*** ***G****enerative* ***M****odel* ***T****ransformer* (**GMT**), ***G****eneric* ***M****odeling* ***E****nvironment* (**GME**) (l'environnement de modélisation générique) et beaucoup d'autres. Donc cette diversité favorise la concurrence et assure la durabilité du paradigme émergent.

1.5.2. Inconvénients de l'IDM

Les points faibles de l'IDM sont [45] :

- ✓ Cependant, l'IDM n'est pas une approche de développement adapté à tous les domaines du génie logiciel. Pour obtenir des avantages de l'IDM, les modèles doivent être à un haut niveau d'abstraction. Cela est généralement possible que si les modèles spécifient des concepts similaires et récurrents.
- ✓ Pour spécifier des modèles sur un haut niveau d'abstraction, des langages de modélisation spécifiques au domaine sont souvent construites, ce qui implique des efforts notables pour construire et maintenir la chaîne d'outils nécessaires autour des modèles.
- ✓ L'IDM est une jeune technologie de génie logiciel, et de nombreux outils qui existent pour les langages de programmation comme par exemple pour Versionnement

(***Versioning***) ou Refactorisation (***Refactoring***) sont pour les modèles que partiellement disponibles et soumis à des recherches en cours.

- ✓ La diversité entre les différentes approches IDM rend l'enseignement du paradigme assez difficile, ce qui empêche en fin l'adoption industrielle de nouvelles techniques.
- ✓ Bien que la diversité des organisations et des sociétés assure la pérennité du nouveau paradigme, mais il existe un besoin croissant de normes d'IDM. Sans de telles normes, un temps précieux est gaspillé sur la mauvaise communication entre les praticiens de différentes sous-communautés de l'IDM et la construction, la maintenance et l'intégration de fragments d'outils qui seraient mieux partagés.

1.6. Défis de l'évolution de l'IDM

Il y a des recherches importantes sur les modèles et l'évolution au cours des dix dernières années, mais de nombreux défis demeurent, Certains sont techniques et théoriques, d'autres sont plus axés sur la pratique et le processus d'ingénierie. On identifie certains des principaux défis d'IDM dans cette section.

1.6.1. L'évolutivité

Un défi fondamental pour l'IDM et pas seulement l'évolution dans l'IDM est l'évolutivité. Les théories et les outils de l'IDM doivent prendre en charge de grands méta-modèles (comme ***UML 2.x***, ***AUTOSAR***, ***EAST-ADL***), de petits méta-modèles (comme les langages artisanaux créés en un seul) et tous les éléments intermédiaires. Ils doivent soutenir la construction de grands et petits modèles, ainsi que des opérations qui s'exécutent sur de gros modèles. Le soutien à l'évolution de modèles très volumineux est requis. Cela peut signifier que fournir des installations pour ignorer certaines parties du modèle qui sont hors de portée pour résoudre les problèmes de migration et d'évolution, pour une meilleure décomposition des méta-modèles, par exemple : dans des éléments qui ne devraient pas évoluer et des éléments susceptibles d'évoluer, pour des mécanismes de persistance plus évolutifs pour les modèles et les nouvelles architectures pour les opérations de gestion de modèles, par exemple : les transformations qui réagissent aux événements déclenchés, telles que les modifications apportées aux parties d'un modèle. Beaucoup de ces concepts sont bien connus dans d'autres domaines qu'IDM, mais des recherches seront nécessaire pour les exploiter [79].

1.6.2. La gestion de l'automatisation

L'un des importants principes de l'IDM est d'automatiser les tâches répétitives et sujettes aux erreurs, les opérations comme les transformations, les comparaisons, les validations et la fusion visent à encapsuler certains des modèles d'automatisation qui ont été identifiés pour l'IDM. Il ya toujours un compromis entre l'automatisation et le contrôle : une fois qu'une tâche a été automatisée "partiellement ou totalement", le degré de contrôle dépend du fonctionnement de la tâche et ce qu'elle produit est probablement réduit. Dans la gestion de modèle, chaque opération, méta-modèle et modèle peuvent évoluer, et cette évolution doit être gérée systématiquement et précisément. Un défi particulier est automatiser idéalement un processus d'ingénierie, mais l'automatisation dépend des méta-modèles, qui peuvent être sujets aux changements [86].

1.6.3. L'hétérogénéité des dépendances

Les trois principaux artefacts d'IDM modèles, méta modèles et opérations sont tous interdépendants, et une grande partie de l'évolution d'IDM nécessite la gestion et le contrôle de ces interdépendances. Un défi majeur consiste à gérer l'hétérogénéité des interdépendances, il existe de nombreux types de relations entre les artefacts d'IDM. Par exemple : entre le modèle et le méta-modèle, il existe une relation de conformité. Entre le modèle et l'opération, il y a au moins un paramètre et des relations générés. Une théorie sonore et précise des dépendances hétérogènes entre les artefacts d'IDM est nécessaire, ainsi qu'un soutien d'outil conforme et pragmatique [3].

1.6.4. L'études empiriques

Certaines recherches ont identifié différents types de changement de méta-modèle et ont classé ces changements selon différents critères. Cependant, une classification plus nuancée et empirique est encore à produire. En particulier, on a besoin de la question : quels types de changements de méta-modèle sont les plus fréquents dans les méta-modèles déployées? Grâce à ces informations, on peut cibler des solutions de migration de modèles et de co-évolution vers les problèmes d'évolution les plus fréquents et obtenir une confiance accrue que ces solutions répondent effectivement aux problèmes d'évolution d'IDM les plus importants [40].

1.6.5. L'utilité des outils

Une grande partie de la recherche sur les modèles et l'évolution s'est focalisée jusqu'ici sur les théories et les outils de l'utilisateur "les langages pour spécifier les règles de migration / co-évolution de modèle, les générateurs de stratégies de migration / co-évolution de modèle, les algorithmes de différenciation, les cadres mathématiques pour exprimer les opérateurs de changement, etc.". L'utilisation de ces outils pour résoudre les problèmes des utilisateurs finaux n'a pas été explorée de manière approfondie et systématique. La plupart des outils existants sont basés sur ***Eclipse*** à cause de son facilité d'utilisation. De plus, la compatibilité syntaxique et sémantique avec d'autres opérations de gestion de modèles a également contribué au développement de solutions de migration de modèles et de co-évolution, les constructeurs d'outils de migration / co-évolution veulent que leurs solutions soient familières et faciles à apprendre pour les utilisateurs des autres applications "autres outils de gestion de modèle aussi bien qu'***Eclipse***". Une étude approfondie des utilisateurs, ainsi que la prise en compte des exigences des utilisateurs qui ne sont pas directement prises ou influencées par ***Eclipse***, serait une orientation utile pour la recherche future [3].

1.6.6. La syntaxe concrète

L'évolution du méta-modèle est souvent la cause des problèmes de migration / co-évolution d'IDM "la migration / co-évolution de modèles ou d'opérations vers le nouveau méta-modèle". Un sujet qui n'a pas reçu autant d'attention que d'autres aspects de l'évolution d'IDM traite de la syntaxe concrète, comment évoluer la syntaxe concrète à la suite de l'évolution du méta-modèle ou de la migration / co-évolution de modèle? De nombreuses approches à la syntaxe concrète dans l'IDM sont génératives (par exemple ***EuGENia***) et basées sur le méta-modèle, donc une solution plausible pour l'évolution de la syntaxe concrète serait de mettre à jour les générateurs et de régénérer tous les éditeurs. Cependant, pour les grands méta-modèles, ou des langages complexes, cela peut être inefficace et il peut être également difficile. Ainsi, il existe des recherches plus importantes pour mieux étayer la syntaxe concrète, par exemple au moyen d'approches basées sur les visiteurs [34].

1.6.7. Les temps d'exécution

L'évolution du langage peut se produire à tout moment, par exemple : lors de la publication d'une nouvelle version d'une norme, à la suite de modifications apportées à l'infrastructure de l'outil, par exemple : ***EMF***. Une hypothèse pour la plupart des recherches sur les modèles et l'évolution est que « la migration / co-évolution des modèles et des

opérations se produit \ hors ligne », c'est-à-dire lorsque les éditeurs de modèles ne sont pas utilisés, ou bien les transformations ne sont pas exécutées, etc.

Cette hypothèse n'est pas claire pour les architectures IDM à grande échelle qui peuvent être nécessaires pour supporter des langages très importants et des modèles très volumineux, par exemple : pour gérer les transformations des modèles continus.

Il se peut que les processus évolutifs et les migrations / co-évolutions soient effectués tandis qu'un langage est utilisée par les ingénieurs, par exemple : la forme d'un éditeur de modèle, un outil de transformation de modèle. Pour soutenir une telle adaptation d'exécution, on devra probablement poursuivre des recherches sur de nouvelles façons de décomposer les définitions de langage et de découpler les opérations à partir des définitions de langage, dans le sens des approches génériques utilisées dans le développement de logiciels générateurs [62].

1.6.8. Les interfaces externes

Les processus et les outils d'IDM ne doivent pas fonctionner dans le vide, ils doivent interagir avec d'autres processus non IDM pour assurer la qualité, la certification, l'élicitation précoce des exigences et ils doivent interagir aussi avec des outils, par exemple : compilateurs, débogueurs, serveurs d'intégration continue. Sans parler des politiques et des processus organisationnels. L'évolution du méta-modèle, la migration / co-évolution du modèle et l'évolution de l'exploitation peuvent avoir un impact sur l'interopérabilité et l'intégration avec des processus et des outils non IDM, il peut exister des applications qui interagissent avec des méta-modèles, par exemple : via une ***API***, sur des nouvelles versions de méta-modèles [59].

1.6.9. La sémantique linguistique

Les recherches sur l'IDM se sont concentrées sur la syntaxe abstraite (méta-modèles) et la syntaxe concrète (éditeurs, cadres graphiques tels que ***GMF*** et ***Graphiti***), la sémantique étant relativement sous-traitée par comparaison. Il y a des questions ouvertes liées à la maintenance de la sémantique via la migration / co-évolution de modèle. Si un modèle est migré / co-évolué du langage X vers le langage Y, est-il possible (via le processus de migration / co-évolution) de fournir des garanties de la préservation de la sémantique des entités du langage X quand codé dans Y ou de la préservation d'autres propriétés souhaitables? Une certaine considération de la préservation sémantique a été réalisée, par

exemple : dans le travail sur ***COPE***, bien que la théorie et la pratique de la préservation sémantique sous la migration / co-évolution de modèle est encore sous-développée [70].

1.6.10. Les méta-modèles

Les modèles dépendent des méta-modèles. Les opérations IDM dépendent des méta-modèles. La gestion de l'évolution des modèles exige la gestion de l'évolution des méta-modèles. La plupart des solutions pour modéliser l'évolution et la co-évolution se sont concentrées sur les méta-modèles.

Une autre approche consisterait à rejeter entièrement les méta-modèles. Dans quelle mesure peut-on prendre en charge l'IDM sans méta-modèles ? Par exemple : en utilisant des schémas ***XML*** ou des outils de dessin tels que ***yEd***.

Peut-on utiliser des outils de modélisation flexibles, tels que ***GenMyModel7*** ou ***EuGENia Live8***, pour supporter les processus IDM rigoureux? Les méta-modèles apportent des avantages importants, ils fournissent effectivement un système qui permet de vérifier et de valider les propriétés des modèles. Mais l'évolution et la migration / co-évolution impliquant des méta-modèles peuvent être très coûteuses.

Une autre voie d'investigation intéressante pourrait être explorée lorsque des méta-modèles peuvent être imposés plus efficacement dans un processus IDM. Par exemple : un processus IDM peut commencer par utiliser des outils de dessin flexibles comme ***yEd***, puis un méta-modèle est imposé une fois que l'analyse de domaine appropriée a été effectuée, et le méta-modèle implicite s'est stabilisé. IDM sans méta-modèle (phase précoce) peut avoir les mêmes compromis que les magasins de données sans schéma, popularisés par le mouvement ***NoSQL***. Les méta-modèles sont importants et utiles, mais on ne doit pas être dogmatiques en termes qu'ils devraient être appliqués dans l'IDM [26].

1.7. Principales caractéristiques des solutions d'évolution dans l'IDM

En essayant de gérer l'évolution et le changement dans l'IDM, il existe un certain nombre de caractéristiques importantes associées aux solutions existantes.

1.7.1. Le champ d'application

La solution est-elle applicable à la gestion du changement pour un type d'artefact d'IDM à la fois, ou plus d'un artefact à la fois? L'évolution aux méta-modèles déclenche des changements à d'autres artefacts d'IDM [57].

1.7.2. L'automatisation

Dans quelle mesure la solution est-elle automatisée? Les solutions actuelles vont des approches manuelles (qui fournissent une syntaxe spécifique à la tâche et des outils pour créer des solutions personnalisées et évolutives) à celles entièrement automatisées [57].

1.7.3. L'environnement

Dans quelle mesure un environnement spécialisé est-il nécessaire pour soutenir l'évolution d'IDM? En particulier, les outils de modélisation standard, par exemple : ***Eclipse EMF***, peuvent-ils être utilisés pour construire des modèles ? ou faut-il utiliser des éditeurs de modèles spécialisés ou des enregistreurs des opérations? [57].

1.7.4. La conformité

Lorsque le changement de méta-modèle est impliqué, les solutions d'évolution doivent fournir des moyens pour établir et rétablir la conformité à un méta-modèle. Par exemple : un méta-modèle peut changer et les modèles par la suite doivent être mis à jour pour se conformer à la nouvelle version du méta-modèle. Quand la conformité de modèle a leur méta-modèle doit-elle appliquée? [57].

2. L'architecture dirigée par les modèles (MDA)

A l'industrie, l'idée de l'IDM a été adoptée et normalisé dans le ***MDA*** (***A****rchitecture* ***D****irigée par les* ***M****odèles*) en anglais (***M****odel* ***D****riven* ***A****rchitecture*), par *l'****O****bject* ***M****anagement* ***G****roup* (***OMG***) à la fin de 2000. Les idées de base de MDA sont étroitement liées à la programmation générative, les usines de logiciels et les langages spécifiques au domaine. MDA est basé en normes multiple, y compris l'installation de ***M****eta-****O****bject* ***F****acility* (***MOF***), l'***U****nified* ***M****odeling* ***L****anguage* (***UML***), l'***O****bject* ***C****onstraint* ***L****anguage* (***OCL***) et le ***XML*** ***M****etadata* ***I****nterchange* (***XMI***). Une implémentation de référence de MOF est essentiel par exemple ***E****clipse* ***M****odeling* ***F****ramework* (***EMF***).

En d'autres termes, cette méthodologie met l'accent sur les modèles formels d'un système comme la source ultime de toutes les activités du cycle de vie du développement logiciel, y compris la conception, la mise en œuvre, le déploiement et la documentation. Il exploite également divers types de transformations de ces modèles [47].

2.1. Principes du MDA

MDA se fonde sur plusieurs principes qui sont sous l'avis de l'***OMG*** de l'***IDM*** [66] :

- ✓ Les modèles doivent être exprimées dans une notation bien définie. Ceci est de parvenir à une compréhension commune du modèle, et ce faisant améliorer la communication et la compréhension du système
- ✓ Les spécifications des systèmes doivent être organisées autour d'un ensemble de modèles et de transformations associées et les relations entre les modèles.
- ✓ Les modèles doivent être construits en conformité avec un ensemble de méta-modèles, facilitant ainsi une véritable intégration et de transformation entre les modèles et l'automatisation grâce à des outils.

2.2. Système et environnement du MDA

Comme MDA a été créé principalement pour le développement de logiciels, on pourrait supposer que le système dans ce contexte signifie système informatique ou logiciel. Toutefois, ce terme est défini beaucoup plus largement comme «*une collection de parties et de relations organisées pour atteindre un but*». Avec cette définition, le système peut contenir non seulement des ordinateurs et des logiciels, mais aussi des personnes, des entreprises et même des groupes d'entreprises.

L'environnement est tout extérieur au système qui interagit avec lui. La pensée intéressante ici est qu'un système peut jouer un rôle d'environnement pour un autre système, ce qui nous permettant d'appliquer les mêmes principes récursivement dans le cas échéant [46].

2.3. Normalisation

L'IDM est embrassé par diverses organisations et sociétés, y compris ***OMG***, ***IBM*** et ***Microsoft***. L'IDM englobe une grande variété de techniques différentes, mais conceptuellement il contient des très semblables approches, y compris les usines de logiciels de Microsoft, ***M****odel-****I****ntegrated* ***C****omputing* de Vanderbilt (***MIC***), ***C****omputer* ***A****utomated* ***M****ulti-****P****aradigm* ***M****odeling* de l'Université McGill (***CAMPaM***) et d'autres approches individuelles. Il y a aussi une grande variété d'outils disponibles, tels que le framework ***E****clipse* ***G****enerative* ***M****odel* ***T****ransformer* (***GMT***), ***G****eneric* ***M****odeling* ***E****nvironment* (***GME***) et beaucoup plus [45].

En *2001*, *l'**O**bject **M**anagement **G**roup* a présenté le ***M**odel-**D**riven **A**rchitecture* (***MDA***) comme un ensemble de normes pour l'IDM. UML a été encore proposé d'être le langage de modélisation de base. Cependant, au lieu de se concentrer sur une sémantique fixe, les modèles UML étaient censés être enrichi avec des informations spécifiques à un domaine sémantique.

Des contraintes sémantiques entre les différents types de diagrammes pourraient être appliquées en évaluant un domaine spécifique ***W**ell-**F**ormedness **R**ule**s*** (***WFRs***) exprimées en ***OCL***. La structure de l'UML et d'autres langages pourrait être définie dans une variante de diagrammes de classes UML, appelée ***M**eta **O**bject **F**acility* (***MOF***).

En se fondant sur un langage de liaison tels que ***J**ava **M**etadata **I**nterface* (***JMI***), les modèles pourraient alors être interrogés et transformés de manière standard. En outre, les requêtes et les transformations de modèles sérialisés selon le ***XML I**nterface **M**étadonnées* (***XMI***) pourrait être mis en œuvre avec la technologie XML à usage général tels que **XSLT** [49].

En *2002*, l'***OMG*** a émis une demande de propositions visant à normaliser ce soi-disant domaine des requêtes, vues et transformations (***QVT***). À la fin de *2005*, l'***OMG*** a proposé une norme consistant en trois langages pour le modèle de transformation. Bien que ces langages étaient basées sur pas mal de langages de recherche populaires, la norme ***QVT*** encore émergente souffre de problèmes qui sont semblables à ceux qui ont ralenti l'adoption de la norme ***UML***.

Par conséquent, on peut attendre à ce que les enseignements tirés de l'effort de normalisation ***UML*** sont pris en compte pour faciliter l'adoption de la norme ***QVT***. Franchir une étape supplémentaire, on peut même vouloir se spécialiser ***UML*** pour les transformations de modèles, puisque les langages de modélisation sont déjà modélisés d'une manière orientée objet en utilisant ***MOF***. Étonnamment, la relation entre la norme ***UML*** et le standard ***QVT*** est minime.

Alors que les trois langages ***QVT*** utilisent un fragment de l'***OCL*** pour exprimer des chemins à travers les éléments du modèle, les constructions de conception par contrat (invariants, pré et post-conditions) qui dominent la plupart des spécifications ***OCL*** pour les modèles d'affaires sont censés être sans importance pour ***QVT***. En outre, la norme ***QVT*** ne repose pas sur les types de diagrammes ***UML*** visuels pour le comportement de modélisation.

Au lieu de cela, des nouveaux concepts de langage tels que des expressions de modèle, les zones, les domaines et les règles sont introduites. Cela donne l'impression que la transformation du modèle est fondamentalement différente de la programmation orienté objet

(méta). Dans ce cas, la connaissance mature orientée objet (l'analyse, la conception, les essais et la restructuration) devrait être revu pour correspondre au nouveau paradigme introduit par les langages de transformation de modèle standard [44].

2.4. Niveaux d'abstraction du MDA

- ✓ Le cadre MDA présente trois niveaux d'abstraction d'un modèle peut être représenté.
- ✓ Les niveaux d'abstraction sont organisés dans une hiérarchie avec le niveau le plus abstrait au sommet, et chaque niveau descendant étant plus détaillée.
- ✓ Les niveaux d'abstraction se limiter à trois niveaux, mais normalement tous les niveaux dans la hiérarchie auront les caractéristiques de l'un des niveaux décrits par MDA.
- ✓ Entre chaque niveau, il y a une application de l'abstraction plus élevé à l'abstraction inférieure. Ces applications peuvent être considérées comme des transformations de modèles entre deux modèles adjacents dans la hiérarchie.
- ✓ Le niveau d'abstraction est déterminé sur la base de ce qui est en cours de modélisation, ce qui signifie que se produisent un certain écart par rapport aux descriptions de niveau ci-dessous.

La figure I.3 montre les trois niveaux d'abstraction et de la façon dont ils se rapportent les uns aux autres [77].

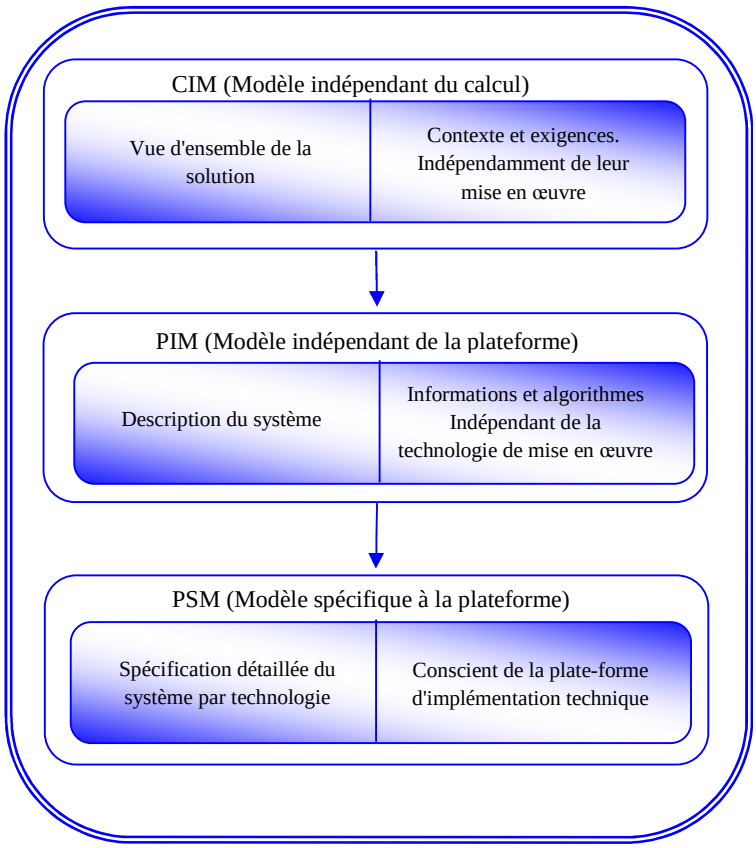


Figure I.3 : *Les trois niveaux d'abstraction définie par MDA [77]*

2.4.1. Le Modèle indépendant du calcul «*The Computation-Independent Model*» (CIM)

Est le modèle le plus abstrait et placé au sommet de la hiérarchie. Il représente normalement le contexte, les conditions et le but du système, mais ne comprend pas de détails sur la façon dont le système doit être mis en œuvre. Cela permet de garder le modèle indépendant de toute plate-forme ou de la technologie. Les modèles à ce niveau sont communément appelés *modèles d'affaires* ou des *modèles de domaine* car ils sont constitués de termes et des notations appartenant à un domaine spécifique. Notez que le CIM n'est pas nécessairement limité à des implémentations logicielles, certaines parties peuvent être destinées à rester à ce niveau d'abstraction, comme la documentation composée dans une langue naturelle [77].

2.4.2. Le modèle indépendant de la plateforme «*The Platform-Independent Model*» (PIM)

Est un modèle qui décrit le comportement et la structure du système, mais reste indépendant de la plateforme. A ce niveau, le modèle comprend des informations et des algorithmes, mais rien qui va l'attacher à une technologie spécifique. Ces modèles sont souvent appelés *modèles de système* comme logiques car ils se concentrent généralement sur la façon dont les composantes du système interagissent les uns avec les autres, sans inclure les données spécifiques de la plateforme. De cette façon, le modèle peut être associé à plusieurs technologies telles que ***Java***, ***C#*** et ***JavaScript***. Il est seulement la partie logicielle du CIM qui est mappé à ce niveau. Les documentations et d'autres objets qui ne sont pas basée sur les logiciels ne seront pas représentés à ce niveau d'abstraction [41].

2.4.3. Le modèle spécifique à la plateforme «*The Platform-Specific Model*» (PSM)

Est en bas de la hiérarchie et comme son nom l'indique le modèle comprend des informations sur le comportement de la structure du système spécifié de telle sorte qu'il est spécifique à une plateforme unique. Ces modèles sont appelés *modèles de mise en œuvre*, car ils modélisent comment le système doit être mis en œuvre sur une plateforme spécifique afin de mener à bien sa fonction.

L'architecture à trois niveaux est définit afin d'améliorer les systèmes. Le processus architectural comprend: la compréhension de la portée des systèmes, les intérêts, les besoins, et arriver à une conception pour satisfaire ces exigences. En utilisant les niveaux de modélisation de MDA, il est possible de comprendre et de spécifier un système tout en ignorant des détails tels que la façon dont il devrait être mis en œuvre dans une certaines plateformes ou technologies. Le CIM et le PIM peuvent être mises en correspondance à des multiples modèles, ce qui améliore la réutilisabilité du modèle [4].

2.5. Les transformations

Les méta-modèles utilisés dans un processus d'IDM peuvent être modifiés. Donc les transformations de modèle, et aussi les conservateurs de cohérence de modèle, doivent également être synchronisés avec les modèles changeants. Ceci est nécessaire à la fois au niveau syntaxique et à un niveau sémantique.

Certains langages de transformation de modèle permettent également au développeur de traiter les transformations comme modèles. Ensuite, des techniques d'IDM peuvent être

utilisées pour analyser et transformer les transformations par des transformations dites d'ordre supérieur.

En particulier, des approches de cohérence peuvent également être utilisées pour synchroniser des transformations de modèle avec des méta-modèles changeants. Par exemple : les transformations de modèle qui se rapportent à un élément de méta-modèle modifié peuvent être marquées, ou des modifications plus simples, telles que la modification du nom d'une méta-classe, peuvent être propagées directement à la spécification de cohérence [84].

2.5.1. Définition de la transformation des modèles

«*Une transformation est la modélisation de génération automatique de modèle cible à partir de modèle source, selon une définition de transformation. Une définition de transformation est un ensemble de règles de transformation qui décrivent ensemble comment un modèle dans le langage source peut être transformé en un modèle dans le langage cible. Une règle de transformation est une description de la façon dont une ou plusieurs constructions dans le langage source peuvent être transformés en un ou plusieurs construit dans le langage cible*». Pour le dire en un mot, une transformation du modèle transforme un modèle de source dans un modèle cible. Bien qu'il est fortement souhaité que les modèles soient conformes à leurs méta-modèles [13].

2.5.2. Classification des langages de transformations des modèles

Il existe de nombreuses approches et outils disponibles de transformation de modèles existants, l'***OMG*** fournit par exemple un ensemble standard de langages de transformation de modèle appelé ***Q**uery **V**iew **T**ransformation* (***QVT***)

QVT se compose de différents langages de transformation de modèles. Les langages de transformations des modèles sont souvent classés en *transformations homogènes* et *transformations hétérogènes* [2].

- ***Les transformations de modèles homogènes :*** transforment les modèles spécifiés dans un seul langage dans les modèles du même langage (voir Figure I.4). Des exemples de transformations de modèles homogènes sont modèles de refactorisation ou étapes d'évolution du méta-modèle qui sont mises en œuvre par des règles de transformation.
- ***Les transformations de modèles hétérogènes :*** sont des transformations du modèles qui transforment les modèles de langage de modélisation de la source dans les

modèles d'un langage de modélisation de la cible qui ne sont pas identiques (voir la figure I.5). En outre, la migration / co-évolution de modèle après le changement du méta-modèle peut être considérée comme transformation de modèle hétérogène même si la majeure partie du langage de modélisation peut rester inchangé [10].

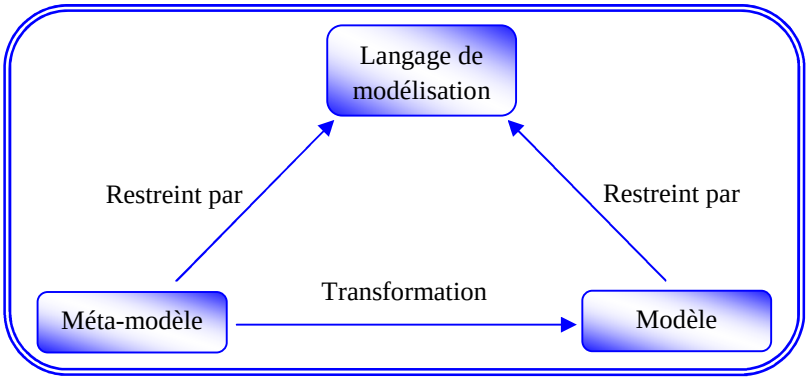


Figure I.4: *transformation du modèle Homogène [2]*

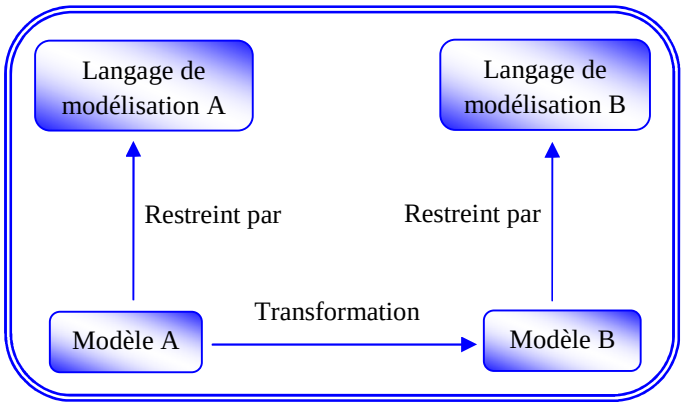


Figure I.5: *transformation du modèle hétérogène [10]*

2.6. Contraintes

Un méta-modèle restreint l'ensemble des instances de modèles possibles. En outre, les méta-modèles peuvent être annotées par des contraintes. Dans ***MOF***, quelques annotations de contraintes telles que les contraintes de multiplicité sont intégrés dans le langage de méta-modélisation, à savoir, codé en dur dans le méta-méta-modèle. Leurs sémantiques sont textuellement décrites dans la spécification ***MOF*** et vérifiées par les éditeurs de modèles. Ces annotations que nous appelons des *contraintes intégrées*. En outre, ***MOF*** prend également en charge les contraintes attachées dans un langage de contrainte textuelle appelé ***O****bject* ***C****onstraint* ***L****anguage* (***OCL***). Avec ***OCL***, le méta-modèle ou modèle est annoté par un script et les modèles sont vérifiés par un vérificateur ***OCL*** [2].

CONCLUSION

Nous avons introduit dans ce premier chapitre les principes généraux de l'IDM, tel que modèle, méta-modèle, les transformations et autres notions. Par la suite on était cité les avantages et les inconvénients d'IDM et le dernier point parle sur les défis d'évolutions d'IDM. De plus, qu'on était parlé sur MDA tel que sa normalisation et ses principaux niveaux d'abstractions.

CHAPITRE II

Co-évolution modèles / méta-modèles

INTRODUCTION

Les méta-modèles peuvent être considérés comme l'un des concepts constitutifs de l'IDM, ils constituent les langages par lesquelles une réalité donnée (modèles) peut être décrite dans un sens abstrait. Les méta-modèles devraient évoluer au cours de leur cycle de vie, comme tout autre artefact logiciel.

L'évolution est un aspect inévitable qui affecte les méta-modèles. Lorsque les méta-modèles évoluent, la conformité du modèle peut être interrompue. La co-évolution du modèle est essentielle dans l'ingénierie dirigée par les modèles pour adapter automatiquement les modèles aux versions les plus récentes de leurs méta-modèles [1].

Les solutions de co-évolution modèle/méta-modèle s'appliquent à deux artefacts IDM à la fois (l'intention est de mettre à jour les ***modèles*** afin qu'ils soient conformes à un ***méta-modèle*** évolué). Une variété de solutions a été proposée pour la co-évolution modèle/méta-modèle qui varie en fonction de leur degré d'automatisation, de l'environnement nécessaire à la co-évolution et de la tenue de la relation de conformité entre le modèle et le méta-modèle.

Toutes ces solutions sont partiellement automatisées. Certaines modifications apportées au méta-modèle (modifications avec effet de bord et non soluble) ne peuvent être traitées que manuellement, car elles nécessitent une expertise de domaine à les résoudre. Les approches varient en fonction de la façon dont elles traitent les changements. La plupart des approches permettent aux ingénieurs d'intervenir dans un processus évolutif, alors que d'autres les excluent [23].

Malheureusement, la co-évolution des modèles n'est pas toujours simple et présente des difficultés intrinsèques qui sont liées au type d'évolution auquel le méta-modèle a été soumis [20]. Ce chapitre explore les différents problèmes liés à l'évolution des méta-modèles en mettant l'accent sur la co-évolution modèles/méta-modèles.

1. Types de modifications effectuées sur les méta-modèles

Lors de l'évolution du méta-modèle, deux types d'évolutions sont distingués : *Les modifications atomiques* et *Les modifications complexes* [68].

1.1. Les modifications atomiques

Les modifications atomiques sont des modifications simples d'un élément du méta-modèle qui peuvent être des additions, des suppressions et des mises à jour [68].

1.2. Les modifications complexes

Qui consistent en une séquence de modifications atomiques combinés ensemble. Par exemple : la modification « ***move meta-property »*** est un changement complexe dans lequel une propriété est déplacée d'une classe à une autre via une référence. Ceci est fait par deux modifications atomiques: supprimer la propriété de la classe et ajouter la propriété dans l'autre classe. Dans la littérature, plus de ***soixante*** modifications complexes proposés [68].

2. Types d'impacts de modifications effectuées sur les méta-modèles

Les modifications survenues sur un méta-modèle peuvent avoir des effets différents sur les modèles correspondants. À cet égard, les modifications de méta-modèle sont classées en trois types : les modifications *sans effet de bord*, les modifications *avec effet de bord* et soluble et les modifications *avec effet de bord et non soluble* [12].

2.1. Les modifications sans effet de bord

En anglais **(*No breaking changes*),** les modifications survenant dans le méta-modèle ne brisent pas la conformité des modèles au méta-modèle. En d'autres termes, les changements sans effet de bord consistent en des additions de nouveaux éléments dans un méta-modèle MM conduisant à MM_0 sans compromettre des modèles qui se conforment à MM et donc, à leur tour, se conformer à MM_0 [12].

Nous citons dans ce qui suit quelques exemples de ce type de modifications [15].

2.1.1. Généraliser une méta-propriété « Generalize meta-property »

Une méta-propriété est généralisée lorsque sa multiplicité ou son type est modifié. Par exemple : si la multiplicité *3..n* de méta-classe *MC* est modifiée à *0*..n, aucune action de co-évolution n'est requise sur les modèles correspondants puisque les instances existantes de *MC* sont toujours conformes à la nouvelle version de la méta-classe.

2.1.2. Ajouter des méta-classes (non obligatoires) « Add (non-obligatory) meta-class »

L'introduction de nouvelles méta-classes est une pratique courante dans l'évolution du méta-modèle qui donne lieu à des extensions de méta-modèle. L'ajout de nouvelles méta-classes non obligatoires (multiplicité non spécifiée) ne brisent pas la conformité des modèles aux méta-modèles.

2.1.3. Ajouter des méta-propriétés (non obligatoire) «Add (non-obligatory) meta-property»

Ceci est similaire au cas précédent car une nouvelle méta-propriété est non obligatoire en ce qui concerne la cardinalité spécifiée. Les modèles existants maintiennent la conformité au méta-modèle considéré si l'addition se produit dans des méta-classes abstraites sans sous-classes.

2.2. Les modifications avec effet de bord et soluble En anglais (*Breaking and resolvable change*),

Des changements qui brisent la conformité des modèles au méta-modèle, même s'ils peuvent être automatiquement co-évolués [12].Quelques exemples de modifications de ce type sont cités au dessous.

2.2.1. Extraire une super-classe (abstraite) « Extract (abstract) super-class »

Une super-classe est extraite dans une hiérarchie et un ensemble de propriétés est activé. Si la super-classe est abstraite, les instances du modèle sont préservées [15].

2.2.2. Supprimer des méta-classes « Eliminate meta-class »

La suppression des méta-classes est une pratique courante dans l'évolution du méta-modèle qui donne lieu à des sous méta-modèle. En général, les modèles correspondant sont changés et toutes les instances de la méta-classe sont supprimées. En outre, si la méta-classe supprimée comporte des sous-classes ou elle est référée par d'autres méta-classes, la suppression provoque également des effets secondaires sur les entités liées [15].

2.2.3. Suppression des méta-propriétés « Eliminate meta-property »

Si une propriété est supprimée d'une méta-classe, elle a le même effet de la modification précédente [15].

2.2.4. Push méta-propriété «Push meta-property »

En basculant une propriété dans les sous-classes signifie qu'elle est supprimée d'une super-classe initiale *A* et ensuite l'ajouter dans toutes ses sous-classes. Une telle modification de méta-modèle ne nécessite aucune co-évolution de modèle, si la super-classe initiale est abstraite, sinon toutes les instances de la super-classe initiale ainsi que ses sous-classes doivent être modifiées en conséquence [15].

2.2.5. « Flatten hierarchy »

Signifie que supprimer une super-classe et introduire toutes ses propriétés dans les sous-classes. Ce scénario peut être renvoyé à la méta-propriété [15].

2.2.6. Renommer des méta-éléments « Rename meta-element »

Renommer des méta-éléments est un cas simple dans lequel le changement doit être propagé aux instances existantes et peut être exécuté d'une manière automatique [15].

2.2.7. Déplacer une méta-propriété « Move meta-property »

Il consiste à déplacer une propriété d'une méta-classe vers une autre méta-classe. Ceci est un changement résoluble et les modèles existants peuvent être facilement évolués en déplaçant la propriété de toutes les instances de la méta-classe aux instances de l'autre méta-classe [15].

2.2.8. Extract/Inline meta-class

L'extraction d'une méta-classe signifie créer une nouvelle classe et déplacer les champs pertinents de l'ancienne classe vers la nouvelle. Par contre, *Inline meta-class* signifie déplacer toutes les fonctionnalités d'une classe dans une autre classe et supprimer la première classe. Les deux modifications de méta-modèle induisent des co-évolutions de modèles automatisés [15].

2.3. Les modifications avec effet de bord et non soluble En anglais (***Breaking and non resolvable change***).

Dans cette catégorie les modifications survenant dans le méta-modèle brisent la conformité des modèles au méta-modèle, et ces modifications ne peuvent pas être résolues automatiquement. Une intervention humaine est donc nécessaire afin d'apporter des informations supplémentaires pour assurer l'opération d'adaptation. Par exemple : la suppression d'un élément du méta-modèle signifie que l'instance concernée doit être supprimée alors que cette dernière est reliée à d'autres éléments [12].

Nous présentons ci-dessous quelques exemples de ce type de modification :

2.3.1. Ajouter des méta-classes (obligatoire) « Add (obligatory) meta-class »

L'ajout de nouvelles méta-classes augmente les problèmes de co-évolution puisque les nouveaux éléments sont obligatoires par rapport à la multiplicité spécifiée. Dans ce cas, des nouvelles instances de la méta-classe ajoutée doivent être introduites dans les modèles existants [15].

2.3.2. Ajouter des méta-propriétés (obligatoire) « Add (obligatory) meta-property »

Dans ce cas, l'intervention humaine est nécessaire pour spécifier la valeur de la propriété ajoutée dans tous les éléments du modèle concernés [15].

2.3.3. Tirer une méta-propriété « Pull meta-property »

Une méta-propriété *P* est tirée dans une super-classe *A* et elle est retirée d'une sous-classe *B*. En conséquence, les instances de la méta-classe *A* doivent être modifiées en héritant de la valeur de *P* des instances de la méta-classe *B* [15].

2.3.4. Restriction d'une méta-propriété « Restrict meta-property »

Une méta-propriété est restreinte lorsque sa multiplicité ou son type est appliqué. C'est un cas complexe où les instances doivent être co-évolué ou restreintes. Par exemple : la

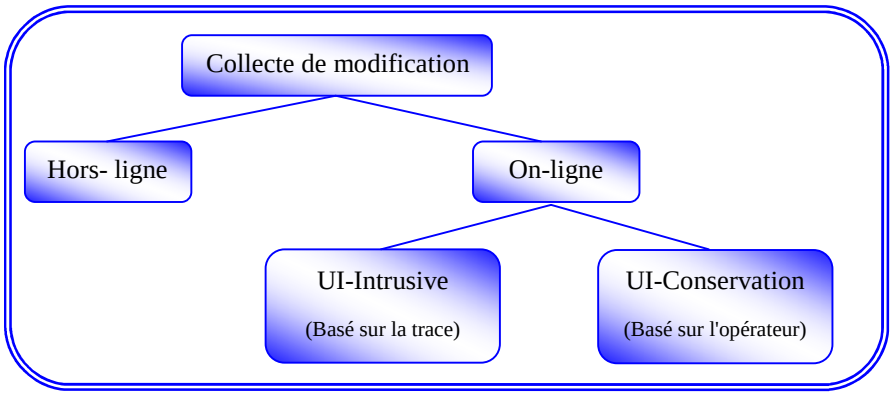
restriction de la limite supérieure de la multiplicité nécessite une sélection de certaines valeurs à supprimer [15].

2.3.5. Extrait (non-abstraite) super-classe « Extract (non-abstract) super-class »

Une super-classe est extraite dans une hiérarchie et un ensemble de propriétés est tiré. Dans ce cas, les effets se réfèrent au cas du "*Pull meta-property*" [15].

3. Collecte des modifications du méta-modèle

La collecte des modifications (changements) de méta-modèle peut se produire en mode hors ligne ou en ligne (voir la figure II.1) [65].



Figue II.1 : *Mode de collection des modifications de méta-modèle [65]*

3.1. Collecte des modifications hors ligne

La collecte des modifications hors ligne se produit après la modification du méta-modèle. Les informations de changement collectées se composent uniquement des deux versions de méta-modèle. La collecte des modifications hors ligne peut se faire indépendamment de l'outil de modélisation utilisé pour modifier le méta-modèle. Cependant, les changements ne sont pas directement collectés, mais doivent être identifiés à l'étape suivante. En référence au fait que cette identification est souvent faite sur la base d'une comparaison de l'état avant et après d'un méta-modèle, les approches basées sur des changements hors ligne sont parfois appelées ***approches basées sur l'état*** [65].

3.2. Collecte des modifications en ligne

La collecte des modifications en ligne survient quand les modifications sur le méta-modèle sont effectuées. Ici, les informations collectées sont une liste des changements survenus. En général, la collection de modifications en ligne présente l'avantage que les changements capturés sont ordonnés chronologiquement. Cependant, la collecte de changements en ligne n'est possible que si on applique des outils qui permettent la collecte des changements en ligne et fournit le résultat de la collecte des modifications aux responsables de la co-évolution des modèles.

En outre, on a différencié les approches ***UI-conserving*** et ***UI-intrusive*** [65].

3.2.1. Les approches de préservation de l'interface utilisateur (UI-preserving approaches)

Ces approches n'influent pas sur l'éditeur utilisé pour changer le méta-modèle. Ainsi, ils n'influent pas sur la façon dont le modélisateur change le méta-modèle. La collecte des changements se fait en observant / retraçant les actions réalisées dans l'éditeur, ce qui conduit à une liste des changements atomiques commandés [65].

3.2.2. Les approches intrusives de l'interface utilisateur (UI-intrusive approaches)

Par contre ces approches influent sur les éditeurs. Dans toutes les approches correspondantes qui peuvent être trouvées dans la littérature jusqu'à présent, cette modification de l'éditeur se produit en ajoutant de nouvelles actions d'édition sous la forme d'opérateurs, qui sont définis par un changement complexe ou atomique de méta-modèle. C'est pourquoi ces approches sont souvent appelées «***approches basées sur l'opérateur***». En conséquence, les approches de collecte de modifications intrusives de l'interface utilisateur peuvent également enregistrer une liste ordonnée de changements atomiques et complexes [65].

4. Co-évolution des méta-modèles

Un modèle doit toujours être conforme à son méta-modèle. Il est de même pour les modèles de transformation, qui consistent à générer un modèle cible, à partir d'un modèle source, en appliquant des règles de transformation. Ainsi, les modèles et les modèles de transformation sont liés à leur méta-modèle.

Un méta-modèle, comme tout système logiciel, est amené à évoluer au fils du temps pour différentes raisons : satisfaire de nouveaux besoins, correction des problèmes existants, etc. Mais une simple modification d'un méta-modèle peut avoir des conséquences directes sur les artefacts (modèles, modèles de transformation, contraintes) qui lui sont liés [53].

La co-évolution consiste à faire évoluer un artefact à l'évolution de son méta-modèle. Plusieurs travaux ont été effectués pour résoudre ces différents problèmes. Ils proposent des solutions basées sur le principe de co-évolution des différents artefacts avec leur méta-modèle.

Même si le sujet de notre mémoire concerne seulement l'étude de la co-évolution des modèles avec leur méta-modèle, il est intéressant de présenter le cas de co-évolution des autres artefacts. En effet, certains problèmes peuvent être communs à tous les artefacts et ainsi, leurs solutions peuvent être réutilisées [53].

4.1. Co-évolution modèle de transformation (opération) /méta-modèle

Une transformation de modèles prend en entrée un ou plusieurs modèles conformes à un méta-modèle source et génère un modèle conforme à un méta-modèle cible. Si l'un des méta-modèles évolue, la transformation peut devenir incohérente. Il devient nécessaire alors de faire évoluer le modèle de transformation pour qu'il puisse s'exécuter correctement.

Les approches proposés qui permettent de co-évoluer les modèles de transformations, comportent généralement deux étapes : ***l'étape de détection*** et ***l'étape de co-évolution*** [53].

4.1.1. L'étape de détection

Elle consiste à détecter des changements entre la version initiale et la version évoluée du méta-modèle. Les changements peuvent être "atomiques" ou "complexes". Les changements atomiques sont calculés à l'aide d'un "modèle de différences". "Des formules de la logique des prédicats" sont appliquées sur les changements "atomiques" pour déterminer ceux qui sont "complexes", puis représenter ces changements "complexes" sous forme d'un "modèle de différences". Ces "modèles de différences" sont conformes au méta-modèle MCL, c'est un langage visuel pour la description de l'évolution d'un méta-modèle [22].

4.1.2. L'étape de co-évolution

Cette étape prend en entrée la version initiale et la version évoluée du méta-modèle, ainsi que les changements calculés lors de l'étape précédente. Les changements sont classés selon leur impact sur la transformation en trois catégories "sans effet de bord, avec effet de bord et soluble, avec effet de bord et non soluble". Ils ont utilisé des transformations de type ***HOT*** pour faire évoluer les modèles de transformation [22].

4.1.3. Aperçu des approches de co-évolution opérations / méta-modèles

Les opérations qui sont définies sur les méta-modèles et s'appliquent aux modèles, peuvent également évoluer en réponse à des changements dans les exigences ou les changements dans les méta-modèles.

Ainsi, quelles approches existent pour aider à soutenir la co-évolution des opérations d'IDM (comme les transformations, les comparaisons, les fusions) et les méta-modèles? Les approches de la co-évolution des opérations suivent généralement des approches de la co-évolution de modèles/ méta-modèle.

Les étapes de co-évolution des opérations / méta-modèles sont :

- ✓ Le méta-modèle est modifié, et dans certaines approches un modèle de changement "qui enregistre les modifications spécifiques qui sont faites" est produit pour une utilisation ultérieure.
- ✓ Deuxièmement, on évalue la mesure dans laquelle les artefacts (opérations) dépendants ont été affectés par l'évolution du méta-modèle.

- ✓ Enfin, la propagation des changements est utilisée pour co-évoluer les artefacts dépendants.

Il y a moins d'approches pour la co-évolution opération / méta-modèle, le problème est beaucoup plus difficile, car il nécessite une évolution et une co-évolution de la sémantique opérationnelle ainsi que la structure [22].

- ➢ ***L'approche Méndez et Al***

Méndez et Al utilisent un moniteur de méta-modèle pour signaler les changements et traiter les événements de changement pour sélectionner et exécuter une stratégie de co-évolution appropriée pour les transformations du modèle en évolution.

L'approche n'est pas entièrement automatisée et des conseils d'ingénieur sont parfois requis pour sélectionner une stratégie. Plus généralement, cette approche pourrait également être utilisée pour la co-évolution du méta-modèle.

Encore plus généralement, ces approches peuvent être classifiées comme étant axées sur des événements, et les opérations axées sur des événements ont un potentiel significatif dans l'IDM pour répondre aux préoccupations d'évolutivité.

La présentation d'une méthodologie pour l'évolution couplée des méta-modèles et des transformations est particulièrement intéressante dans cette approche. C'est qu'il essaie d'évaluer le coût d'un changement et utilise cette information pour informer l'évolution des transformations ATL, par exemple : pour décider s'il faut ou non faire une transformation [23].

4.2. Co-évolution contraintes / méta-modèles

En IDM, les méta-modèles sont souvent décrits en utilisant le langage ***MOF***. Des contraintes ***OCL*** sont le plus souvent ajoutées pour affiner la syntaxe visée par un méta-modèle. Cependant durant l'évolution d'un méta-modèle ces contraintes ***OCL*** sont, soit omises, soit réécrites manuellement. Ceci est couteux en temps et source d'erreurs.

Des approches décriées pour aider les concepteurs à adapter les contraintes ***OCL*** attachées à un méta-modèle, lors de son évolution progressive. L'évolution d'un méta-modèle est vue comme une séquence des opérations élémentaires. Après chaque opération élémentaire, le système vérifie l'état de toutes les contraintes pour identifier celles qui sont impactées.

Pour chaque contrainte impactée, le système propose une adaptation, quand celle-ci est possible, sinon souligne l'incohérence de la contrainte.

L'approche de co-évolution contraintes/méta-modèles assure une cohérence syntaxique des contraintes adaptées automatiquement. Le concepteur doit ensuite s'assurer qu'elles

correspondent à son intention pour une validation sémantique. Pour cela, une approche collaborative a été mise en place. Le système propose des adaptations automatiques mais c'est au concepteur de les valider, de les modifier ou de les annuler [56].

4.3. Aperçu des autres approches de co-évolution

Bien que des recherches importantes aient porté sur la co-évolution modèle/méta-modèle et une très petite quantité a considéré la co-évolution opération/méta-modèle, il existe des approches qui se situent à l'extérieur de ces grandes catégories de recherche.

4.3.1. L'ingénierie Round-trip

Une des approches les plus fondamentales pour gérer l'évolution dans l'IDM est ***l'ingénierie Round-Trip*** parfois appelée ***synchronisation***. Les transformations de modèle sont utilisées par exemple pour produire un code exécutable à partir de modèles, et après la modification du code, les modèles sont régénérés à partir du code.

Ces approches sont spécifiques à l'évolution du modèle et supposent généralement que les méta-modèles restent inchangés. Les approches naïves ont tendance à lutter avec des bases de code non triviales et de grands modèles.

Les approches plus nuancées utilisent ***des deltas*** ou des ***modèles de changement*** pour ne faire que des changements pertinents aux modèles source, et ne changer pas les parties non affectées. Un exemple d'outils de l'ingénierie Round-Trip est ***FUJABA***, qui utilise des grammaires à trois graphes pour supporter la synchronisation incrémentielle [38].

4.3.2. L'évolution du modèle en tant que problème de transformation

Une catégorie importante de recherches a considéré que l'évolution du modèle est un problème de transformation, et elle peut être directement traitée à l'aide de langages de transformation de modèles. Il s'agit d'une perspective attrayante et il semble y avoir des preuves à l'appui de ce point de vue.

Le jeu d'outils de ***Henshin*** a examiné l'utilisation de transformations de graphe pour soutenir la co-évolution de modèle et l'évolution de méta-modèle. Les transformations de graphe (qui effectuent la co-évolution du modèle) sont générées automatiquement à partir des mappages d'évolution du méta-modèle, par exemple : qui relient les méta-éléments initiaux et cibles [18].

Rose a également examiné l'utilisation de transformations de modèle pour soutenir la co-évolution de modèle, mais observe qu'il existe des modèles inhérents aux transformations par exemple : le soi-disant motif de copie conservateur qui ne sont pas directement soutenus par des langages de transformation [64].

Ces questions sont abordées dans ***Flock,*** en incorporant certains schémas évolutifs en tant que constructions syntaxiques et d'autres, par exemple : copie conservatrice dans la

sémantique du langage, alors que dans ***Henshin***, des techniques génératives d'ordre supérieur sont appliquées [60].

Henshin et ***Flock*** sont des outils de co-évolution polyvalents [25].

4.3.3. Evolution du schéma

Une grande partie du travail sur l'évolution du modèle a été inspirée ou empruntée directement à la richesse de la recherche sur l'évolution du schéma dans les bases de données. Bien qu'une grande partie des idées clés issues de l'évolution du schéma aient été directement adoptées ou adaptées à l'évolution du modèle [29].

Une considération essentielle reste à résoudre, c'est qu'après l'évolution du schéma et la co-évolution des données, il faut également une évolution des applications pour être compatible avec le nouveau schéma. Alors que la recherche d'IDM sur la co-évolution opération/méta-modèle est une étape dans ce sens, il faut poursuivre les recherches, en particulier pour explorer comment les applications développées sans l'utilisation d'IDM, mais cette interface avec les méta-modèles, peuvent être co-évoluées automatiquement et efficacement [52].

4.3.4. Versionnement du modèle (Model Versioning)

Le versioning du modèle joue un rôle important dans la gestion de l'évolution dans le contexte d'IDM. Le but du versioning en génie logiciel est double. Tout d'abord, le contrôle de version concerne le maintien d'une archive historique d'un ensemble d'artefacts (modèles et méta-modèles dans le cas de l'IDM), car ils subissent une série de changements. Deuxièmement, le versioning supporte l'évolution simultanée d'un artefact par une équipe d'ingénieurs. En tant que tel, le versioning de modèle peut être applicable et utile dans des scénarios IDM collaboratifs impliquant la co-évolution de modèle, par exemple : lorsque deux ou plusieurs versions d'un méta-modèle doivent être prises en charge simultanément dans un projet [49].

La version des méta-modèles et des modèles est très différente de la version d'un tel code source. Cela est dû à la structure à base de graphes des artefacts de modélisation. Les systèmes de version à base de texte tels que ***Subversion*** ou ***Git*** ne considèrent que l'information textuelle d'un artefact de modélisation tel que sa sérialisation ***XMI***. Ce faisant, ils ne tiennent pas compte des informations structurelles de modèles tels que des références de confinement ou des multiplicités entre différents éléments de modèle. Par conséquent, les techniques et les systèmes de gestion de version basés sur les graphiques sont nécessaires.

Dans le cas de la modélisation d'artefacts, cette approche n'est pas simple. Les modèles dans IDM ont des représentations duales manifestées par leur syntaxe abstraite et concrète. La consolidation séparée de ces représentations peut constituer une tâche difficile [73].

D'autre part, la résolution automatique de l'évolution peut être obtenue en calculant toutes les combinaisons possibles d'opérations exécutées en parallèle menant à une version valide. ***Cicchetti*** et ***Al*** proposent un langage spécifique au domaine pour définir les modèles de coniques et les stratégies de rapprochement. Ces modèles et stratégies peuvent être utilisés pour détecter et concilier les deux coniques syntaxiques et sémantiques [8].

Les approches basées sur des politiques nécessitent une intervention de l'utilisateur dans des scénarios où aucune politique n'est spécifiée. Enfin, au lieu d'essayer de résoudre les coniques, un modèle de système de version peut simplement les tolérer. Les recherches font valoir que la tolérance temporaire des coniques peut être bénéfique car ces coniques mettent en évidence des zones du système où une analyse plus poussée est nécessaire [17].

4.4. Identification des changements de méta-modèle

Le but de l'identification du changement est d'obtenir une liste précise des changements atomiques et complexes et de leurs types qui avaient été appliqués au méta-modèle. Plus la détection de changements complexes est précise, sera meilleure l'automatisation et la correction de la co-évolution. Les tâches liées à l'identification du changement dépendent de la nature et de la précision des informations fournies par la collecte des changements. La figure II.2 résume les différentes variations de l'identification du changement [28].

4.4.1. Identification des changements atomiques

Les changements atomiques doivent être identifiés dans le cas de la collecte de modifications hors ligne, où les informations de changement ne sont fournies que sous la forme d'une paire de versions de méta-modèle. L'identification des changements atomiques se fait sur la base d'une différence des deux méta-modèles.

Cependant, dans le cas général, la précision des résultats n'est pas nécessairement optimale pour deux raisons.

- ✓ Premièrement, aucune information sur l'ordre chronologique de l'application des changements atomiques n'est disponible.
- ✓ Deuxièmement, il est possible que les effets soient masqués lorsque plusieurs changements sont appliqués. Par exemple : une modification "déplacer la propriété *Y* de la classe *A* vers la classe *B*" qui est suivie par une modification "changer le nom de la propriété *Y* à *X*" se produira sous la forme d'un groupe de deux changements atomiques seulement: "suppression de *Y* à *A*" et " Addition de *X* à *B* ". Les changements atomiques «ajouter *Y* à *B*» et «renommer *Y* à *X* »s ont masqués.

Notez que dans des cas particuliers, l'outil utilisé peut capturer suffisamment de méta-données pour compenser ces défauts [28].

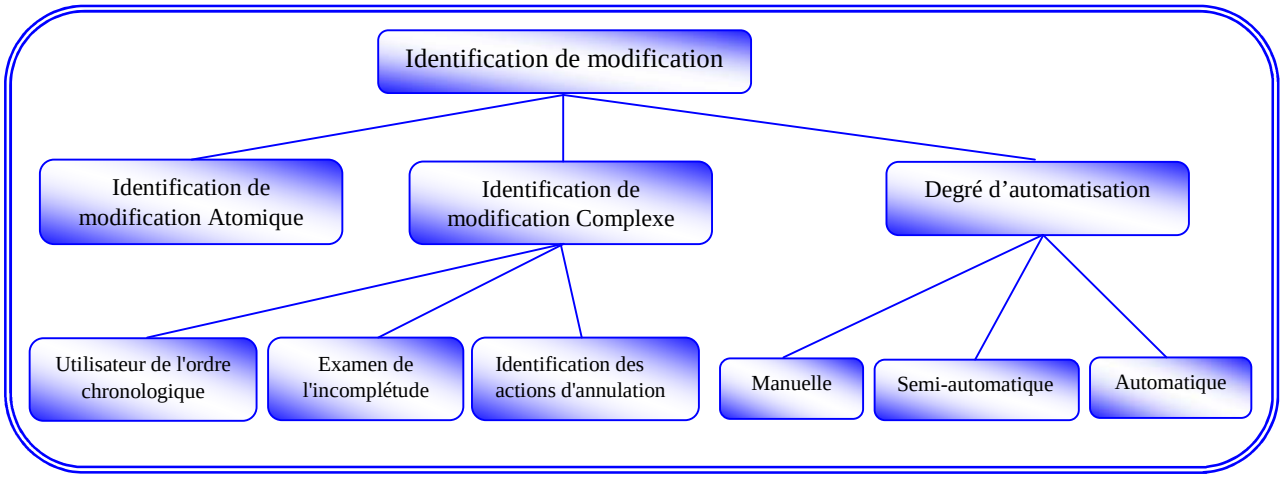


Figure II.2 : *Types et caractéristiques pour identifier les changements de méta-modèle [28]*

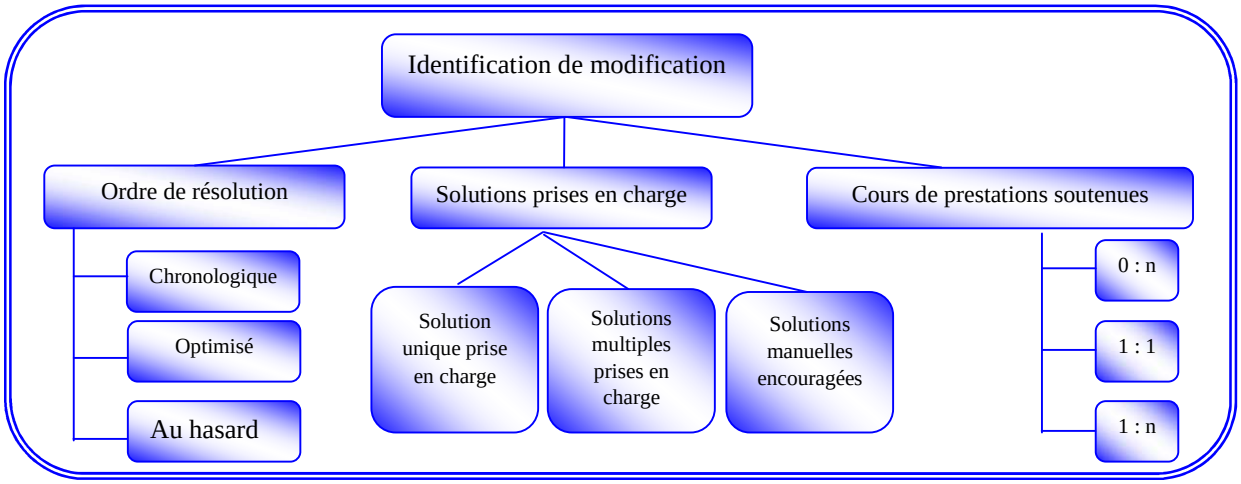


Figure II.3 : *Fonctionnalités pour la résolution de modèles [28]*

4.4.2. Identification de changement complexe

Les changements complexes aident à augmenter la quantité d'automation pendant la co-évolution du modèle. L'identification de changements complexes se fait sur la base d'une liste de changements atomiques et éventuellement d'une liste de changements complexes collectés. L'objectif est d'identifier les ensembles de changements dans ces listes qui forment ensemble un changement complexe.

En toute difficulté, la même liste de changements atomiques pourrait permettre d'autres listes de changements complexes. Par exemple : deux modifications "ajouter une propriété" et "supprimer une propriété" peuvent être interprétées comme "déplacer la propriété" ou "retirer la propriété" (de la sous-classe à la méta-classe). Ainsi, lors de l'identification de changement complexe, il ne suffit pas de rechercher des combinaisons de changements qui pourraient constituer un changement complexe. Plutôt, il faut estimer la probabilité de l'exactitude de l'alternative identifiée.

- ***Utilisation de l'ordre chronologique***

La distance chronologique des changements atomiques peut être considérée comme un facteur lors de l'estimation de la probabilité si les changements appartiennent au même changement complexe ou non. Notez que les informations sur l'ordre chronologique ne sont disponibles que lorsque la collecte de modifications en ligne a été utilisée.

- ***Prise en compte de l'incomplétude***

En outre, certains algorithmes nécessitant une identification par changement atomique peuvent prendre en compte que les changements atomiques identifiés peuvent être incomplets en raison des effets cachés.

- ***Identification des actions d'annulation***

Les approches basées sur une liste complète des changements atomiques doivent faire face à un problème différent: les traces de modification incluent également les actions qui ont été annulées. Par exemple : lorsque le modélisateur supprime accidentellement une propriété et l'ajoute à nouveau pour corriger cette action, cela peut être compté comme deux modifications, à savoir "supprimer la propriété" et "ajouter une propriété".

Cependant, l'application de la co-évolution aux deux changements peut être nocive, car la co-évolution de la suppression entraîne une perte d'information dans les modèles. Par exemple : lorsque la suppression accidentelle et la ré-addition concernent une propriété facultative x, l'application de la co-évolution sur les modèles signifierait d'abord supprimer des instances de cette propriété, y compris leurs valeurs. La lecture de la propriété sur le méta-modèle n'entraînerait aucune action automatisée sur les modèles car la propriété est facultative. Ainsi, alors que l'utilisateur qui a changé le méta-modèle a effectué un défilement, les valeurs des propriétés se perdent dans les modèles. Par conséquent, les approches basées sur une liste complète des changements atomiques doivent identifier et filtrer les modifications qui appartiennent aux actions annulées [28].

4.4.3. Degré d'automatisation

L'identification de changement peut être effectuée manuellement, semi-automatique ou entièrement automatique. Alors qu'une automatisation complète serait idéale, il est difficilement possible d'exclure l'imprécision [28].

4.5. Résolution de modèles

Sur la base des changements identifiés, la tâche réelle de la co-évolution du modèle peut être effectuée. Deux concepts centraux pour cela sont «*technique de résolution*» et «*stratégie*

de résolution». Une stratégie de résolution spécifie comment un modèle doit être résolu pour un changement donné.

En revanche, une technique de résolution est une approche globale pour créer et appliquer des stratégies de résolution. Les techniques de résolution diffèrent dans l'ordre de résolution des changements, la façon dont les utilisateurs peuvent contrôler l'application des stratégies de résolution et le rapport entre l'effort manuel et l'automatisation (voir Figure II.3) [28].

4.5.1. Ordre de résolution des modèles

L'évolution du méta-modèle comprend souvent des changements qui peuvent être plus ou moins indépendants les uns des autres. Les stratégies de résolution de ces changements pourraient être appliquées dans un ordre différent des modèles.

Les changements pourraient être traités dans ***l'ordre chronologique*** exact où ils se sont produits. Toutefois, en cas de collecte de modifications hors ligne, les informations sur cette commande ne sont pas disponibles. Ici, une technique de résolution peut soit appliquer un ***ordre aléatoire*** ou récupérer un ***ordre optimal*** pour la résolution.

Un méta-modèle peut être évolué en ajoutant une propriété obligatoire à une classe et en supprimant une des sous-classes de cette classe. L'ajout d'instances de propriété peut nécessiter des décisions humaines sur chaque valeur d'instance ajoutée. Il serait inutile d'ajouter d'abord des valeurs de propriété aux instances de la sous-classe supprimée, car ces instances seront supprimées lors de la résolution de la deuxième modification. Dans ce cas, l'ordre chronologique ne serait pas optimal pour la résolution [28].

4.5.2. Catégories de spécification des stratégies de résolution des modèles

Le rapport entre l'effort qu'un utilisateur investit pour spécifier une stratégie de résolution et l'ensemble des modèles qui peuvent être abordés avec cette stratégie, est un autre facteur pour permettre le contrôle des utilisateurs.

On a trois catégories de spécification des stratégies résolution des modèles :

- ✓ ***1: 1*** Une décision manuelle, telle que la configuration ou la sélection d'une stratégie de résolution, est requise par modèle.
- ✓ ***1: n*** Une décision manuelle, telle que la spécification d'une stratégie de résolution spécifique à un projet, est requise. Cette décision peut être appliquée à tous les modèles à co-évoluer.
- ✓ ***0: n*** Aucune décision manuelle ne doit être prise pour résoudre tous les modèles à co-évoluer. Cela se produit lors de la réutilisation des règles existantes qui sont indépendantes des méta-modèles concrets [28].

4.5.3. Solutions prises en charge

Par type de changement, une technique de résolution peut prendre en charge une ou plusieurs stratégies de résolution pour un changement de méta-modèle. Une stratégie est applicable à tous les changements conformes à son type de changement.

- ***Solution unique prise en charge***

  Il existe deux situations où les approches de la littérature ne prennent en charge qu'une seule solution pour un changement de méta-modèle. Premièrement, cela arrive souvent lorsque les approches possèdent une stratégie de résolution qui n'inclut pas l'intervention humaine. Alternativement, les approches pourraient (partiellement) générer une solution. Ainsi, les solutions uniques ne sont supportées que dans la catégorie *0: n*. Comme un avantage général, les approches de solution unique permettent une automatisation complète de la résolution. Cependant, les résultats de la résolution ne sont pas garantis pour être celui désiré pour les modèles concrets à portée de main. Par exemple : lorsqu'un modèle est utilisé comme entrée d'une génération de code, les instances automatiquement ajoutées avec des valeurs par défaut (comme c'est souvent le cas lorsque des attributs sont ajoutés à des méta-classes) peuvent devenir problématiques [76].

- ***Plusieurs solutions prises en charge***

  Pour donner aux utilisateurs un meilleur contrôle sur le résultat, certaines approches supportent plusieurs solutions par changement. La première stratégie est de permettre aux utilisateurs de spécifier librement de nouvelles stratégies de résolution, qui pourraient être appliquées à des modèles simples ou multiples, par exemple : qui appartiennent au même projet ou société. Deuxièmement, un ensemble de stratégies de résolution alternatives pourraient être présentés à l'utilisateur pour sélectionner. Troisièmement, une stratégie de résolution fournie peut nécessiter une configuration de paramètres spécifiés. Plusieurs solutions peuvent être prises en charge dans toutes les catégories de résolution. La commande acquise se fait au prix de l'interaction de l'utilisateur pendant la résolution (conduisant à un degré réduit d'automatisation). Une façon d'améliorer le degré d'automatisation est de rendre l'interaction utilisateur facultative, ce qui permet de réduire le nombre de décisions des utilisateurs, sans abandonner le contrôle de l'utilisateur. Cela peut être fait en fournissant une stratégie de solution qui peut éventuellement être adaptée. Enfin, il y a le cas particulier des approches basées sur les opérateurs, en particulier celles qui travaillent avec les opérateurs de diffusion. Ici, la décision concernant la stratégie de résolution ainsi que les configurations sont déplacées jusqu'au moment où le

changement de méta-modèle est appliqué. En fournissant différents opérateurs, des stratégies de résolution multiples par changement de méta-modèle peuvent être prises en charge. Après le changement de méta-modèle, la résolution du modèle ne nécessite aucune autre décision [76].

- ***Solutions manuelles encouragées:***

  Outre les deux formes mentionnées ci-dessus, certaines approches ne présentent pas de solutions uniques ou multiples, mais encouragent et permettent une création manuelle d'une résolution pour un modèle. Cette approche est placée dans la catégorie *1: 1* [76].

5. Co-évolution des modèle / méta-modèle

5.1. Définitions

Un méta-modèle décrit les structures et les règles d'une famille de modèles. Lorsqu'un modèle utilise les structures et adhère aux règles définies par un méta-modèle, le modèle est dit conforme au méta-modèle (Bezivin en 2005). Un changement dans un méta-modèle pourrait nécessiter des changements aux modèles pour assurer la préservation de la conformité. Le processus d'évolution du méta-modèle et de ses modèles pour préserver la conformité est appelé ***co-évolution modèle/méta-modèle*** [64].

Les solutions de co-évolution modèle/méta-modèle s'appliquent à deux artefacts IDM à la fois, l'intention est de mettre à jour les modèles afin qu'ils soient conformes à un méta-modèle évolué. Une variété de solutions a été proposée pour la co-évolution modèle/méta-modèle qui varie en fonction de leur degré d'automatisation, de l'environnement nécessaire à la co-évolution et de la tenue de la relation de conformité entre le modèle et le méta-modèle [78]. Une analyse plus approfondie des approches de co-évolution est effectuée au niveau de ce chapitre.

5.2. Théorie de la co-évolution modèle / méta-modèle

Un processus de co-évolution modèle/méta-modèle consiste à changer un méta-modèle et à actualiser les modèles d'instance pour préserver la conformité. Souvent, les deux activités sont considérées séparément, et ce dernier est appelé ***migration***. *Sprinkle* et *Karsai* sont les premiers à identifier le besoin d'approches qui tiennent compte des exigences spécifiques de la co-évolution (*Sprinkle et Karsai* en 2004). En particulier, *Sprinkle* et *Karsai* décrivent les migrations comme distinctes des défis uniques par rapport à l'activité plus générale de la transformation du modèle au modèle.

Comprendre les situations dans lesquelles la co-évolution doit être gérée est important pour formuler les exigences appropriées pour les outils de co-évolution. La migration est

parfois rendue inutile par l'évolution d'un méta-modèle de telle sorte que la conformité des modèles ne soit pas modifiée, par exemple en ne faisant que des changements additifs (*Herrmannsdoerfer et Al* en 2009). La co-évolution peut être réalisée par plus d'une personne et dans certains cas, les développeurs de méta-modèle et les utilisateurs de modèles ne peuvent pas communiquer (*Cicchetti et Al* en 2008). Non pas en tenant compte de ces observations, la littérature sur la co-évolution renseigne rarement sur la manière dont la co-évolution est gérée dans la pratique [64].

5.3. Modèles de co-évolution modèle/méta-modèle

Une grande partie de la littérature suggère que le processus de migration / co-évolution de modèles devrait varier en fonction du type de changements de méta-modèle effectués, par exemple (*Gruschko et Al* en 2007 / *Cicchetti et Al* en 2008). En particulier, sur la co-évolution la littérature classifie les changements de méta-modèle en deux types, ces derniers qui affectent la manière dont la migration est effectuée. Selon le type de changement de méta-modèle, la migration peut être inutile, automatisée ou ne peut être automatisée que lorsque l'ambiguïté est résolue par un développeur (*Gruschko Et coll* en 2007). Les changements de méta-modèle peuvent également être considérés comme étant indépendants du méta-modèle "observés dans l'évolution de plus d'un méta-modèle" ou spécifiques au méta-modèle "observés dans l'évolution d'un seul méta-modèle" (*Herrmannsdoerfer et Al* en 2008).

D'autres recherches sont nécessaires pour identifier les catégories de changements de méta-modèle, car de nombreuses approches automatisées de co-évolution sont étayées par les catégorisations. Bien qu'il ait été suggéré qu'une grande partie des changements de méta-modèle se reproduisent (*Herrmannsdoerfer et Al* en 2008), l'étude dans laquelle cette allégation a été faite ne considère que deux méta-modèles, tous deux issus de la même organisation. L'évaluation de la mesure dans laquelle les changements se reproduisent dans une plus grande et plus large gamme de méta-modèles est un défi de recherche ouvert [64].

5.4. Caractéristiques du cadre de modélisation

5.4.1. Séparation modèle / méta-modèle

Dans les environnements de développement IDM, les modèles et les méta-modèles sont séparés. Les méta-modèles sont développés et distribués aux utilisateurs. Les méta-modèles sont installés, configurés et combinés pour former un environnement de développement IDM personnalisé. Les développeurs de méta-modèle n'ont aucun accès programmatique aux modèles d'instance, qui résident dans un espace de travail différent et potentiellement sur une machine différente. Par conséquent, l'évolution du méta-modèle se produit indépendamment

pour modéliser la co-évolution. La figure II.4 montre les activités typiquement impliquées dans l'évolution. Tout d'abord, le développeur de méta-modèle développe le méta-modèle et peut créer une stratégie de co-évolution. Par la suite, les utilisateurs du méta-modèle découvrent des problèmes de conformité après l'installation de la nouvelle version du méta-modèle et migrent /co-évoluent leurs modèles.

En raison de la séparation entre le modèle et le méta-modèle, la co-évolution est soit développée par le développeur "le développeur de méta-modèle développe une stratégie de co-évolution exécutable, qui est distribuée à l'utilisateur du méta-modèle avec le méta-modèle évolué" soit pilotée par l'utilisateur "le développeur de méta-modèle ne fournit aucune stratégie de co-évolution". Dans les deux cas, la co-évolution du modèle se produit sur la machine de l'utilisateur du méta-modèle, après et indépendamment de l'évolution du méta-modèle [64].

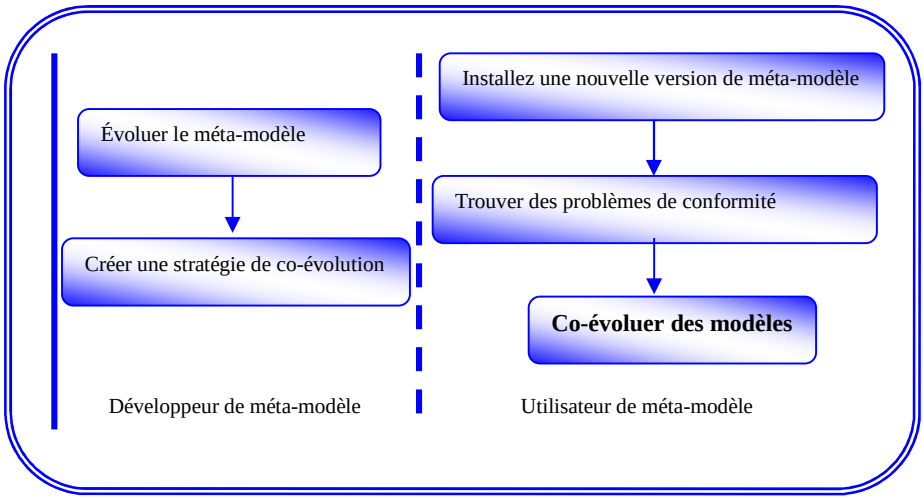


Figure II.4 : *Les activités de co-évolution [64]*

5.4.2. Conformité implicite

Les environnements de développement IDM impliquent implicitement la conformité. Un modèle est lié à son méta-modèle, généralement en construisant une représentation dans le langage de programmation sous-jacent pour chaque élément de modèle et valeur de données. Souvent, la liaison est fortement typée, chaque type de méta-modèle est mappé à un type correspondant dans le langage de programmation sous-jacent en utilisant des mappages définis par le méta-modèle. Par conséquent, les cadres de modélisation ne permettent pas de modifier un modèle qui le ferait ne plus se conformer à son méta-modèle. Le chargement d'un modèle non conforme à son méta-modèle provoque une erreur. En bref, les cadres de modélisation IDM ne peuvent pas être utilisés pour gérer des modèles qui ne sont pas conformes à leur méta-modèle.

Comme les frameworks de modélisation ne peuvent charger que des modèles conformes à leur méta-modèle. Habituellement, l'utilisateur du méta-modèle ne peut exécuter la co-évolution qu'en modifiant directement le modèle, en manipulant normalement sa représentation sous-jacente, par exemple : ***XMI***. Les éditeurs de modèles et les opérations de gestion de modèles, qui font habituellement partie intégrante de l'IDM, ne peuvent pas être utilisés pour gérer des modèles qui ne sont pas conformes à leur méta-modèle et ne peuvent donc pas être utilisés pendant la co-évolution du modèle.

Une autre conséquence de la conformité implicitement appliquée est que les outils de modélisation doivent produire des modèles conformes à leur méta-modèle, et par conséquent, la co-évolution modèle ne peut pas être décomposée. Par conséquent, la co-évolution de modèle ne peut pas être réalisée en combinant des techniques de co-évolution, parce que les étapes intermédiaires doivent produire des modèles conformes [64].

5.5. Catégorisation des approches de co-évolution modèles / méta-modèles

Cette section contribue à catégoriser les approches existantes en matière de co-évolution de modèles et discute des mérites et des limites relatifs de chaque catégorie. Lors de l'évolution du méta-modèle, un développeur peut appliquer toute approche décrite dans cette section pour générer une stratégie de co-évolution correspondante, qui peut ensuite être exécutée pour effectuer la co-évolution de modèle.

On distingue quatre catégories : *les approches de spécification manuelle, les approches d'opérateur, les approches d'inférence* et *les approches hybrides.*

5.5.1. Les approches de spécification manuelle

Cette catégorie considère des contraintes supplémentaires est présentée dans ***Taentzer et Al***, elle définit un processus d'évolution en plusieurs étapes. Cette approche est plus stricte en termes d'application de la conformité au méta-modèle est garantie par la construction après chaque étape de co-évolution, c'est-à-dire après l'application d'un ensemble des transformations graphiques "les règles de co-évolution sont spécifiées à l'aide des transformations graphiques et la conformité à un méta-modèle et la satisfaction des contraintes appliquées aux modèles". Ceci est particulièrement utile pour le raisonnement formel sur la validité des stratégies de co-évolution [36].

Dans la spécification manuelle, la stratégie de co-évolution est codée manuellement par le développeur de méta-modèles, généralement en utilisant un langage de programmation à usage général, par exemple ***Java***, ou un langage de transformation de modèle à modèle tel que ***QVT*** ou ***ATL***. La stratégie de co-évolution peut manipuler les instances du méta-modèle en tout cas permises par le cadre de modélisation, et par conséquent, la spécification manuelle

permet au développeur de méta-modèle d'avoir le plus de contrôle sur la co-évolution du modèle [80].

Cependant, la spécification manuelle nécessite généralement le plus d'efforts de la part du développeur de méta-modèle pour le raison : Ainsi que la mise en œuvre de la stratégie de co-évolution, le développeur de méta-modèle doit également produire un code pour exécuter la stratégie de co-évolution. En règle générale, cela implique l'intégration de la stratégie de co-évolution avec le cadre de modélisation "pour charger et stocker des modèles" et éventuellement avec des outils de développement "pour fournir une interface utilisateur" [54].

Parmi les approches de co-évolution manuelles, nous citons les plus importants:

- ***L'approche Epsilon Flock (Rose et Al)***

  Une approche plus abstraite et automatisée est *Epsilon Flock,* qui soutient une notion de copie conservatrice, dans laquelle les parties du modèle initial qui restent conformes au méta-modèle cible sont automatiquement copiées.

  En tant que tel, les stratégies de co-évolution de *Flock* tendent à être beaucoup plus concises que d'autres approches manuelles et même d'autres approches non manuelles de co-évolution [63].

  La conformité du modèle au méta-modèle s'applique lorsque la stratégie de co-évolution a terminé son exécution. Ce qui est révélateur de Flock, puisque cette approche ignore les contraintes sur les modèles, c'est-à-dire les modèles conformes sont définis entièrement en termes de contraintes modèle/ méta-modèle et ne tiennent pas compte des contraintes supplémentaires qui peuvent s'appliquer au méta-modèle.

  Dans Flock, la co-évolution du modèle est spécifiée manuellement. Flock copie automatiquement uniquement les éléments du modèle qui sont toujours conformes au méta-modèle évolué. Par conséquent, l'utilisateur spécifie la co-évolution uniquement pour les éléments de modèle qui ne sont plus conformes au méta-modèle évolué. En raison de l'algorithme de copie automatique, une stratégie de co-évolution de Flock vide donne toujours un modèle conforme au méta-modèle évolué.

  Par conséquent, un utilisateur commence généralement par une stratégie de co-évolution vide et l'affine automatiquement pour co-évoluer des éléments non conformes. Cependant, il n'y a aucun support pour s'assurer que tous les éléments non conformes sont co-évolués [60].

5.5.2. Les approches d'opérateurs

Les approches d'opérateurs telles que ***COPE / Edapt***, ***Wachsmuth*** ou ***MCL (Model Change Language)*** sont basées sur un ensemble prédéfini de ***micro-stratégies*** "des modèles qui mappent des éléments de méta-modèle initiaux à évolués" [70].

La prémisse de ces approches est qu'un ensemble de ces opérateurs, lorsqu'ils sont appliqués à un méta-modèle, produira un méta-modèle évolué, puis des transformations d'ordre supérieur peuvent être appliquées pour générer une stratégie de co-évolution pour les modèles conformes au méta-modèle initial [35].

De telles approches sont généralement extensibles, de sorte que de nouveaux opérateurs peuvent être conçus et spécifiés pour faire face à des scénarios évolutifs non anticipés. Cependant, ces approches exigent normalement l'utilisation d'un éditeur spécialisé pour la construction de méta-modèles. Ces éditeurs enregistrent les changements effectués au fur et à mesure de l'évolution d'un méta-modèle "en termes d'opérateurs utilisés pour modifier le méta-modèle", afin de rendre la génération automatique des stratégies de co-évolution traitable que possible. Toutes ces approches vérifient la conformité des modèles aux méta-modèles une fois le processus de co-évolution terminé.

Chaque opérateur de co-évolution spécifie une évolution du méta-modèle avec une stratégie de co-évolution modèle correspondante. Par exemple : l'opérateur « *Make Reference Containment* » évolue le méta-modèle de sorte qu'une référence sans confinement devient une référence de confinement et migre /co-évolue des modèles de sorte que les valeurs de la référence évoluée soient remplacées par des copies. En composant des opérateurs de co-évolution, l'évolution du méta-modèle peut être effectuée et une stratégie de co-évolution peut être générée sans écrire de code [5].

- ✓ ***La bibliothèque d'opérateurs Lerner***

  L'utilité d'une approche de co-évolution basée sur l'opérateur dépend fortement de la richesse de la bibliothèque d'opérateurs co-évolutifs qu'elle fournit. Lorsqu'aucun opérateur co-évolutif approprié n'est disponible, le développeur de méta-modèle doit utiliser une autre approche pour effectuer une co-évolution de modèle.

  Cependant, *Lerner* est une grande bibliothèque d'opérateurs, elle augmente la complexité de spécification de la co-évolution. Pour démontrer, *Lerner* considère le déplacement d'une caractéristique d'un type à l'autre. Ceci pourrait être exprimé par l'application séquentielle de deux opérateurs appelés, par exemple : supprimer la fonctionnalité et ajouter une fonctionnalité.

Néanmoins, la sémantique d'un opérateur de fonctionnalité de suppression dicte probablement que les valeurs de cette fonctionnalité seront supprimées pendant la co-évolution, et par conséquent la fonctionnalité de suppression n'est pas appropriée lors de la spécification de cette fonctionnalité a été déplacée. Pour résoudre ce problème, un opérateur de fonctionnalité de déplacement pourrait être introduit, mais le développeur de méta-modèle doit comprendre la différence entre les deux façons de déplacer un type et de sélectionner soigneusement le bon [16].

Lerner fournit d'autres exemples qui élucident davantage ce problème "comme l'introduction d'un nouveau type en divisant un type existant". À mesure que la taille de la bibliothèque des opérateurs co-évolutifs augmente, il en va de même de la complexité de la sélection des opérateurs appropriés et donc de la complexité de l'évolution du méta-modèle.

Une communication claire des effets de chaque opérateur co-évolutif "sur le méta-modèle et ses modèles d'instance" peut améliorer la navigabilité des grandes bibliothèques d'opérateurs co-évolutifs. COPE, par exemple : fournit un nom, une description, une liste de paramètres et des contraintes d'applicabilité pour chaque opérateur co-évolutif.

D'autres techniques peuvent être utilisées pour essayer d'améliorer la navigabilité des grandes bibliothèques d'opérateurs co-évolutifs. Par exemple : COPE restreint le choix des opérateurs uniquement à ceux qui peuvent être appliqués à l'élément de méta-modèle actuellement sélectionné. Trouver un équilibre entre la richesse et la navigabilité est un défi majeur dans la définition des bibliothèques d'opérateurs co-évolutifs pour les approches de co-évolution basées sur les opérateurs [54].

Pour effectuer l'évolution du méta-modèle à l'aide d'opérateurs co-évolutifs, la bibliothèque des opérateurs co-évolutifs doit être intégrée aux outils d'édition des méta-modèles. COPE, par exemple : fournit une intégration avec l'éditeur de méta-modèle en arborescence *EMF*. Cependant, certains développeurs éditent leurs méta-modèles à l'aide d'une syntaxe textuelle, comme ***Emfatic***. En général, l'édition de texte en format gratuit est moins restrictive que l'édition arborescente (car dans ce dernier, le méta-modèle est toujours sonore, alors que dans le premier, le texte n'a pas toujours besoin de compiler).

Par conséquent, il n'est pas clair si la co-évolution basée sur l'opérateur peut être utilisée avec toutes les catégories d'outils d'édition de méta-modèle [39].

- ***L'approche de Wachsmuth***

Cette approche propose une bibliothèque d'opérateurs co-évolutifs pour les méta-modèles MOF. L'approche de Wachsmuth est une approche transformationnelle pour aider à l'évolution du méta-modèle par adaptation progressive. Les étapes sont mises en œuvre comme des transformations dans les relations QVT. Chaque étape forme une adaptation de méta-modèle. La transformation est classée en fonction de ses propriétés de sémantique et préservation de l'instance en trois groupes, à savoir *refactoration*, *construction* et *destruction* [35].

Cette approche se caractérise par la possibilité de réutiliser des scripts d'adaptation dans un scénario d'adaptation similaire et d'éviter les incohérences. Mais cette approche est très limitée en raison de l'atomicité des changements, qui est loin d'être réaliste [35].

- ***L'approche de Levendovszky et Al (MCL)***

MCL (*Model Change Language*) peut être utilisé pour spécifier les règles de co-évolution pour les modèles entre deux versions d'un méta-modèle. Les différences entre deux méta-modèles sont analysées et utilisées pour générer une transformation qui peut être utilisée pour co-évoluer les modèles correspondants. Un méta-modèle est introduit pour décrire l'évolution du méta-modèle [15].

Sur la base de cette description, la nouvelle version du méta-modèle et une transformation pour la co-évolution des modèles correspondants générées [58].On peut utiliser des transformations de modèles d'ordre supérieur pour soutenir la co-évolution des modèles aux changements dans le méta-modèle correspondant [35].

- ***L'approche de Herrmannsdoerfer et Al (COPE / Edapt)***

Est une approche de co-évolution basée sur l'opérateur pour EMF. *COPE* permet aux développeurs de méta-modèles de spécifier des stratégies de co-évolution personnalisées lorsqu'aucun opérateur co-évolutif n'est approprié, en utilisant un langage de programmation à usage général. Par conséquent, les stratégies de co-évolution personnalisées dans COPE subissent une des mêmes limites que les approches de spécifications manuelles.

Par conséquent, il semble que les approches de co-évolution basées sur l'opérateur devraient chercher à fournir une bibliothèque complète d'opérateurs co-évolutifs, de sorte qu'au moins un opérateur soit approprié pour toute co-évolution qu'un développeur de méta-modèle souhaitera effectuer.

COPE fournit une bibliothèque d'opérateurs co-évolutifs. Chaque opérateur co-évolutif spécifie à la fois une évolution de méta-modèle et une stratégie de co-

évolution modèle correspondante. Par exemple : l'opérateur « *Introduce Reference Class* » de COPE évolue le méta-modèle de telle sorte qu'une référence est remplacée par une classe et co-évolue des modèles tels que les liens conformes à la référence sont remplacés par des instances de la classe de référence.

Un utilisateur COPE applique des opérations couplées au méta-modèle initial pour former le méta-modèle évolué. Chaque opération couplée spécifie une évolution de méta-modèle avec un fragment correspondant de la stratégie de co-évolution de modèle. Un historique des opérations appliquées est ensuite utilisé pour générer une stratégie de co-évolution complète. Comme COPE est destiné à la co-évolution de modèles et de méta-modèles, l'ingénierie d'un grand méta-modèle peut être difficile. Déterminer quelle séquence d'opérations produira une co-évolution correcte n'est pas toujours simple. Pour aider l'utilisateur, COPE permet de dérouler les opérations. Pour aider au processus de co-évolution, COPE offre la vue Convergence qui utilise *EMF Compare* pour afficher les différences entre deux méta-modèles.

Bien que cela ait été utile, il est compréhensible que cela ne fournisse qu'une liste de différences explicites et non pas la sémantique d'un changement de méta-modèle. Par conséquent, l'ingénierie d'un méta-modèle grand et inconnu est difficile et la co-évolution ne peut être complétée qu'avec des conseils importants de l'auteur de COPE [15].

5.5.3. Les approches d'inférence (Meta-Model Matching)

Dans l'adaptation de méta-modèle, une stratégie de co-évolution est déduite en analysant le méta-modèle évolué et l'historique du méta-modèle. Les approches d'inférence utilisent l'une des deux catégories d'historique du méta-modèle, Soit le méta-modèle initial "approches de différenciation", soit les modifications apportées au méta-modèle initial pour produire le méta-modèle-évolué.

L'analyse du méta-modèle évolué et le méta-modèle initial donne ***un modèle de différence*** [12]. "une représentation des changements entre le méta-modèle initial et évolué". Le modèle de différence est utilisé pour inférer une stratégie de co-évolution, généralement en utilisant une transformation de modèle à modèle d'ordre supérieur pour produire une transformation de modèle à modèle à partir du modèle de différence. *Cicchetti et Al* [12].et *Garcés et Al* [80].décrivent les approches d'inférence [54].

Les approches d'inférence sont basées sur l'analyse et la comparaison des méta-modèles initiaux et évolués. De telles approches, par exemple *Cicchetti et Al* [51], ou ***AML (A****tlanMod* ***M****atching* ***L****anguage)* [11], utilisent des algorithmes de correspondance de méta-modèle tels que S***imilarity Flooding*** [19] pour identifier les différences entre les méta-modèles, et

l'utiliser pour inférer un enregistrement des changements qui peuvent être utilisés pour générer automatiquement une stratégie de co-évolution. Comme pour les approches d'opérateur, une transformation d'ordre supérieur est utilisée par exemple : dans ATL [35].pour générer la stratégie de co-évolution. Les approches d'inférence sont incomplètes, le même changement pour un méta-modèle peut, en général, être produit de différentes façons, et les approches d'inférence nécessitent une intervention de développeur afin d'éliminer l'ambiguïté à ces cas. Cependant, les approches d'inférence ne nécessitent pas l'utilisation d'un éditeur/environnement spécialisé, et la conformité du modèle co-évolué au méta-modèle évolué est vérifiée à la fin du processus de co-évolution [76].

Par rapport à la spécification manuelle et à la co-évolution basée sur l'opérateur, le matching des méta-modèles nécessite le moindre effort du développeur de méta-modèle qui n'a besoin que d'évoluer le méta-modèle et de fournir un historique de méta-modèle. Toutefois, pour certains types de changements de méta-modèle, il existe plus d'une stratégie de co-évolution modèle possible. Par exemple : lorsqu'une méta-classe est supprimée, une stratégie de co-évolution possible consiste à supprimer toutes les instances de la méta-classe supprimée. Alternativement, le type de chaque instance de la méta-classe supprimée pourrait être changé en une autre méta-classe qui spécifie des caractéristiques structurelles équivalentes.

En général, une approche d'inférence des méta-modèles nécessite des conseils pour sélectionner la stratégie de co-évolution la plus appropriée à partir de toutes les alternatives possibles, les changements de méta-modèles seuls ne fournissent pas suffisamment d'informations pour distinguer correctement les stratégies de co-évolution possibles. Les méthodes d'adaptation des méta-modèles existantes utilisent des heuristiques pour déterminer la stratégie de co-évolution la plus appropriée.

Ces heuristiques conduisent parfois à la sélection d'une mauvaise stratégie de co-évolution. Une alternative à l'utilisation de l'heuristique serait d’intervenir le développeur pour choisir la stratégie de co-évolution la plus adéquate. Parce que les méthodes d'inférence des méta-modèles utilisent des heuristiques pour sélectionner une stratégie de co-évolution, il peut parfois être difficile de raisonner sur quelle stratégie de co-évolution sera sélectionnée. Pour les domaines où la prévisibilité, l'intégralité et l'exactitude sont une préoccupation principale, par exemple : les systèmes critiques de sécurité, ou les systèmes qui doivent être certifiés par rapport à une norme pertinente, l’intervention du développeur est nécessaire afin de démontrer les bons résultats prévisibles.

La figure suivante explique comment le processus de matching de méta-modèle se déroule.

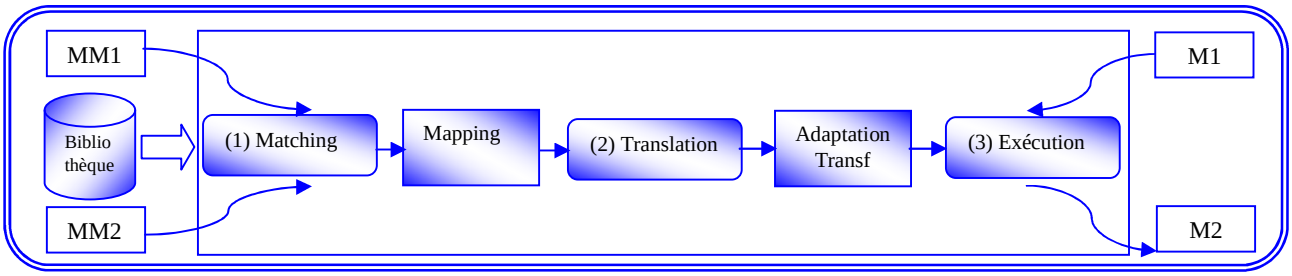


Figure II.5 : *Processus des approches d'inférence [16]*

Trois étapes ont été proposées par les auteurs pour une approche basée sur le matching (voir figure II.5) :

- ✓ ***La première étape*** consiste à l'utilisation d'une bibliothèque pour le calcul des différences entre deux méta-modèles. Ces différences sont représentées par un modèle de matching (modèle de différences).
- ✓ ***La deuxième étape*** transforme ce modèle de matching en une transformation d'adaptation en utilisant des transformations de modèle de type **HOT** (***H****igher* ***O****rder* ***T****ransformation*).
- ✓ Enfin, ***la troisième étape*** exécute les transformations obtenues [16].

➢ ***L'approche de Cicchetti et Al***

Est une approche de co-évolution donnée comme une transformation de modèle d'ordre supérieur qui prend le modèle de différence enregistrant l'évolution du méta-modèle et génère une transformation de modèle capable de produire la co-évolution de modèles.

L'approche se compose des étapes suivantes:

- ✓ Premièrement la décomposition automatique du modèle de différence dans deux sous modèles disjoints, qui dénotent des modifications avec effet de bord et soluble et des modifications avec effet de bord et non soluble.
- ✓ Si ces deux sous modèles sont parallèles indépendants. Alors les co-évolutions correspondantes sont générées séparément.
- ✓ Mais si ces deux sous modèles sont dépendantes en parallèle, elles sont affinées pour identifier et isoler les interdépendances d'interférences [14].

Cette approche ne spécifie pas explicitement comment les modèles de différence sont calculés, mais seulement qu'ils peuvent être obtenus en utilisant un outil tel que ***EMFCompare*** ou ***SiDiff***. L'isolement des interdépendances entre les changements n'est pas toujours possible [12].

- ***L'approche de Garcés et Al***

Consiste en une co-évolution en trois étapes pour adopter des modèles à leurs méta-modèles évolués et suivre ainsi les stratégies des approches de co-évolution d'inférence. Ces étapes sont :

- ✓ Tout d'abord, un processus de co-évolution calcule automatiquement les équivalences et les différences entre deux versions de méta-modèles en exécutant de manière incrémentielle un ensemble d'heuristiques. Les équivalences et différences calculées sont sauvegardées dans un modèle de différence.
- ✓ Deuxièmement, une transformation de co-évolution est dérivée par une transformation d'ordre supérieur qui prend en entrée le modèle de différence. La transformation produite est écrite dans un langage de transformation particulier (par exemple ATL, ***XSLT***, ***SQL-like***).
- ✓ Enfin, la transformation de co-évolution est exécutée.

Les auteurs prouvent que l'approche proposée permet d'obtenir une grande précision dans la détection des changements atomiques et complexes et qu'une bonne performance des stratégies d'inférence est également prouvée.

Un langage spécifique au domaine ***DSL*** (***D****omain* ***S****pecific* ***L****anguage*) pour l'expression des stratégies d'inférence [55].

Cette approche est puissante, car elle permet de calculer les équivalences et les différences entre n'importe quelle paire de méta-modèles, et exécute à la demande des heuristiques modulaires (peuvent être exécuté ou non).

L'approche est générique, les heuristiques sont décrites en termes de concepts ***KM3*** et il peut être mis en œuvre en utilisant d'autres formalismes tels que MOF ou EMFEcore.

Cependant, l'aide de l'utilisateur est requise dans certaines stratégies et noter que la combinaison sémantiquement invalide d'heuristiques peut provoquer une erreur d'exécution, tandis qu'une combinaison incorrecte entraîne la génération d'une transformation de co-évolution incorrecte. L'utilisation de l'heuristique est également ambiguë [57].

- ***L'approche de Gruschko et Al***

Les étapes envisagées de cette approche de co-évolution de modèle proposée sont les suivantes:

- ✓ Premièrement, les versions de modèles sont comparées et les différences sont traduites dans ***le modèle delta***. Les changements trouvés sont classés en catégories.
- ✓ Ensuite, l'aide de l'utilisateur est nécessaire pour les modifications avec effet de bord et non soluble (non automatisée).
- ✓ Enfin, un algorithme approprié pour la migration du modèle doit être déterminé et la co-évolution est exécutée.

L'approche proposée minimise l'effort manuel requis pour effectuer la co-évolution du modèle face aux changements de méta-modèle. Mais les changements sont supposés se produire individuellement et l'utilisation de relations au lieu de modèles de différence ne permet pas de distinguer les mises à jour de méta-éléments des modèles (suppression / addition) [15].

5.5.4. Les approches hybrides.

Une approche hybride « combine des éléments d'inférence et d'opérateurs » qui élimine le besoin de suivi basé sur l'éditeur de l'application de l'opérateur. Il apporte également une nouvelle contribution à la détection de l'opérateur composite fondée sur des modèles de différence, ce qui permet de répondre aux préoccupations liées à la granularité et à l'évolutivité qui s'appliquent aux approches d'opérateur et d'inférence existantes [74].

Pour assurer la co-évolution du modèle avec ses méta-modèles. Le processus de co-évolution suit les étapes suivantes :

- ✓ Dans la première étape, Les différences entre deux versions de méta-modèle doivent être déterminées en utilisant une technique d'inférence.
- ✓ Dans la deuxième étape, en utilisons un moteur d'inférence pour générer différentes stratégies d'évolution en assemblant les changements atomiques dans les éventuels composés.
- ✓ Dans la troisième étape, une bibliothèque d'opérateurs a exploré pour obtenir différentes procédures de co-évolution, qui seront rassemblées pour constituer des stratégies de co-évolution.
- ✓ Dans la dernière étape, les utilisateurs utilisent une stratégie d'évolution sélectionnée, et par conséquent la stratégie de co-évolution sera appliquée sur un modèle spécifique conforme à l'ancienne version afin d'obtenir un nouveau modèle conforme à la nouvelle version du méta-modèle [1].

5.6. Exemple de co-évolution modèles/méta-modèles

Un exemple de méta-modèle basé sur un langage simplifié de la machine à état et le modèle correspondant sont représentés respectivement à la figure II.6 et à la figure II.7.

La figure II.7 montre un exemple d'évolution du méta-modèle simplifié, l'évolution du méta-modèle comprend trois étapes :

- ✓ Extraire des sous-classes de la classe "état" résultant dans InitialState, SimpleState et FinalState.
- ✓ Faire abstraction de la classe "état".
- ✓ Créer et affiner les multiplicités des références prédécesseur / successeur pour les sous-classes [87].

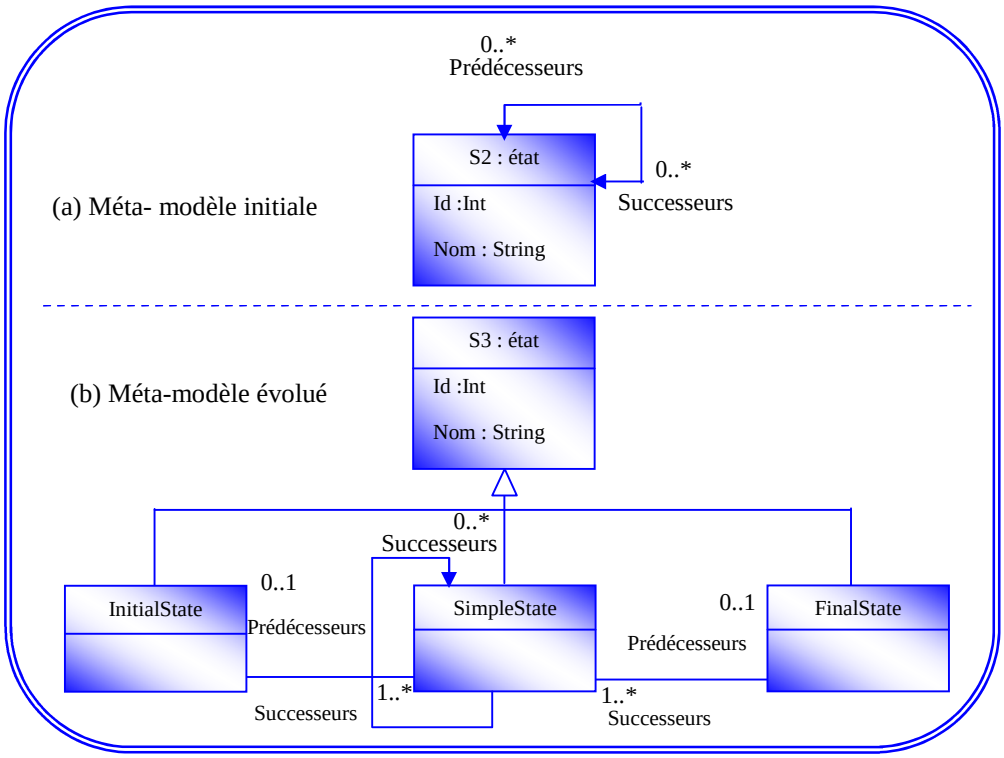


Figure. II.6 : *Exemple d'évolution d'un méta-modèle simplifié [87]*

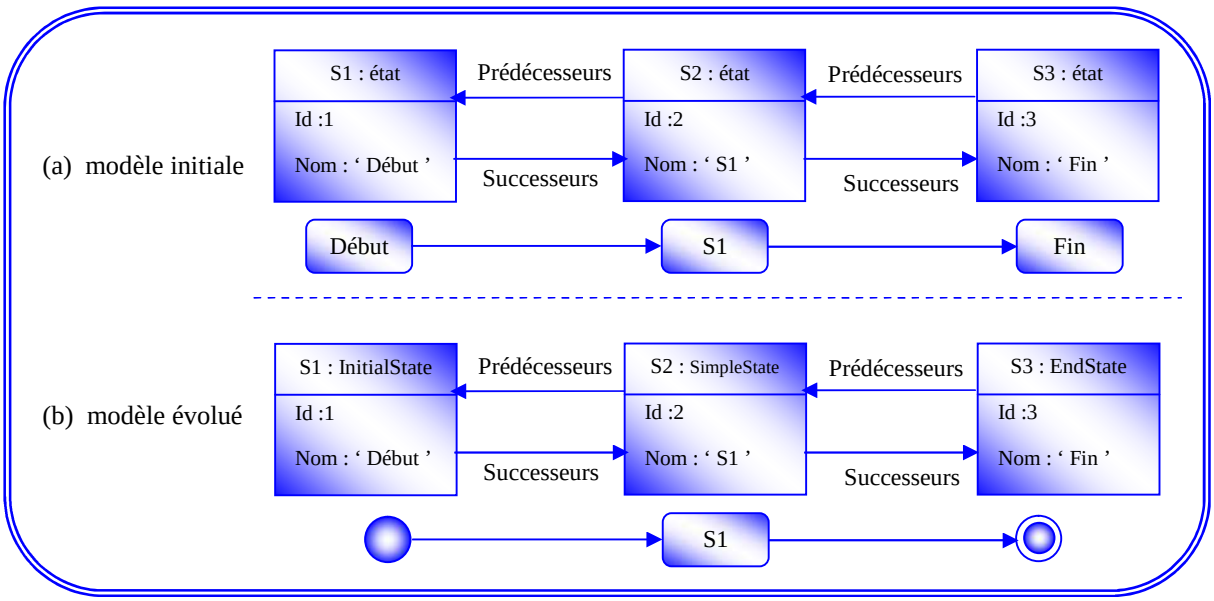


Figure. II.7 : *Exemple de co-évolution du modèle [87]*

CONCLUSION

La revue de la littérature effectuée dans ce chapitre a permis d'identifier l'évolution qui se produit dans les projets IDM. Bien que plusieurs articles proposent des structures et des processus pour gérer l'évolution dans l'IDM, peu de travaux ont considéré la manière dont les artefacts IDM évoluent dans la pratique. Dans ce chapitre, nous étudiaient l'évolution dans l'IDM, ce qui nous a permis d'approfondir la compréhension des problèmes conceptuels et techniques rencontrés lors de l'identification et de la gestion de la co-évolution. Un exemple de co-évolution des modèles / méta-modèles a été identifié, il a été utilisé pour comprendre comment la co-évolution est réalisée dans la pratique.

CHAPITRE III

Étude comparative des approches de Co-évolution modèles / méta-modèles

INTRODUCTION

La co-évolution des méta-modèles a déjà été identifiée comme un problème majeur dans la littérature. Au cours de la dernière décennie, la littérature a introduit plusieurs approches qui se concentrent exclusivement sur la co-évolution de modèles lorsque le méta-modèle est évolué. C'est donc un défi pour les utilisateurs d'identifier quelle approche peut répondre le mieux à leurs besoins [9].

Alors dans ce chapitre, on vise à aider, à choisir et à comparer cette jungle d'approches proposées pour la co-évolution du modèle/méta-modèle en fonction des besoins et des contraintes des utilisateurs.

1. Approches identifiées

Dans cette étude nous avons exclu, les approches qui ont porté sur la co-évolution de méta-modèles avec d'autres artefacts sui diffèrent des modèles. Nous avons convenu d'inclure dans notre étude 07 approches distinctes concernant la co-évolution modèles/méta-modèles. Nous avons sélectionné les approches les plus représentatives de chacune des trois catégories: Les approches de *spécification manuelle*, les approches *d'opérateurs* et les approches *d'inférence*.

Le tableau III.1 résume les sept approches de co-évolution modèle/méta-modèles choisies pour notre étude, leurs années de proposition et leurs catégories [24].

Approches	Année de proposition	Catégorie de l'approche
Cicchetti et Al	2008-2009	Approche d'inférence
COPE / Edapt (Herrmannsdoerfer et Al)	2008-2011	Approche d'opérateurs
Epsilon Flock (Rose et Al)	2009-2014	Approche de spécification manuelle
Garcés et Al	2008-2009	Approche d'inférence
Gruschko et Al	2007	Approche d'inférence
MCL (Levendovszky et Al)	2009-2014	Approche d'opérateurs
Wachsmuth	2007	Approche d'opérateurs

Tableau III.1 : *Les approches de co-évolution modèles/méta-modèles identifiées, et indication de leurs (année de proposition, catégorie)*

2. Critères de comparaison

Dans la section qui suit, nous allons présenter une étude comparative des approches de co-évolution modèles/méta-modèles présentées dans le chapitre précédent. Nous commençons d'abord par décrire quelques critères qui nous ont permis d'effectuer cette comparaison.

2.1. Collection des modifications

On précise si les approches abordent le défi de la collecte des modifications subies par le méta-modèle en mode hors ligne ou en ligne. En outre, nous montrons l'influence de ces approches sur l'éditeur utilisé pour changer le méta-modèle (les approches UI-conserving et UI-intrusive).

2.2. Identification des modifications de méta-modèles

Ce critère permet de comparer les approches en terme de capacité d'identification des modifications atomiques et complexes.

2.3. La résolution de modèles

Dans ce critère, nous comparons les approches à travers la précision d'ordre de résolution des modèles et de montrer la catégorie de spécification des stratégies résolution des modèles

2.4. Correction des résultats requis

Lorsque les modèles sont encore utilisés comme entrée, par exemple pour la génération de code, il se peut qu'il ne soit pas suffisant de récupérer simplement des modèles conformes à la syntaxe définie par le nouveau méta-modèle, mais de récupérer un modèle sémantiquement "correct" [36].

2.5. Automatisation

L'automatisation est un critère important permettant d'évaluer et de comparer les solutions de co-évolution, car le coût de fonctionnement de la co-évolution est lié à l'effort manuel qui doit être investi.

2.6. Outils et mis en œuvre des approches de co-évolution modèles/méta-modèles

Les approches que nous avons examinées ont été partiellement mises en œuvre et certaines de ces implémentations ont même été mises à la disposition du public. Lorsque la capacité d'investissement d'une entreprise est faible, la réutilisation directe d'un outil pourrait

être nécessaire, car une approche pour laquelle une implémentation peut être réutilisée est plus facile à adopter.

2.7. Contrôle du processus de co-évolution

Une caractéristique particulière de la co-évolution des méta-modèles et des modèles est que ses sous-tâches ne doivent pas nécessairement être réalisées dans les mêmes unités organisationnelles ou même dans l'entreprise. Compte tenu de la situation selon laquelle une entreprise, qui veut co-évoluer des modèles après une modification de méta-modèle, pourrait ne pas avoir d'influence sur la décision de modifier le méta-modèle, et par conséquent sur la façon dont les modifications peuvent être collectées.

2.8. Compétences de la personne qui change le méta-modèle

Nous proposons de poser la question suivante : " Est-ce que la personne qui modifie le méta-modèle est un expert dans l'approche co-évolution envisagée ? "

2.9. Compétences de la personne qui co-évolue les modèles

La résolution des modèles comprend le plus souvent des tâches manuelles, soit au niveau des définitions de la stratégie de résolution, soit au niveau de la prise de décision pour les modèles uniques. Cependant, définir des stratégies de résolution pour un domaine ou une entreprise nécessite une compréhension de base de l'approche de co-évolution. De même, prendre des décisions de résolution par modèle nécessite des connaissances sur les modèles ainsi que sur les logiciels et les projets auxquels ils appartiennent.

3. Processus de comparaison

Le processus de comparaison des approches de co-évolution de modèles/méta-modèles a été réalisé en appliquant chacun des critères de comparaison discutés dans la section précédente.

3.1. Collection des modifications

Comme résumé dans le tableau III.2 :

- ✓ 04 approches fonctionnent avec le défi de la collecte des modifications [35].
- ✓ Les 03 autres approches sont basées sur l'hypothèse qu'une liste de modifications est donnée [35].
- ✓ La plupart des approches fonctionnent avec la collection des modifications hors ligne [35].
- ✓ L'exception est une seule approche qui fonctionne avec la collection de modifications en ligne, où le modélisateur utilise une journalisation de préservation de l'interface utilisateur pour obtenir des informations sur les modifications atomiques : l'approche COPE / Edapt est basé sur l'opérateur et donc intrusives [69].

Approches	**Collection des modifications de méta-modèle**		
	Collecte des modifications hors ligne	**Collecte des modifications en ligne**	
		Les approches de préservation de l'interface utilisateur	***Les approches intrusives de l'interface utilisateur***
Cicchetti et Al	Oui	Non	Non
COPE / Edapt	Oui	Non	Oui
Epsilon Flock	-	-	-
Garcés et Al	Oui	Non	Non
Gruschko et Al	-	-	-
MCL	Oui	Non	Non
Wachsmuth	-	-	-

Tableau III.2 : *La collection des modifications de méta-modèle (hors ligne/en ligne) dans les approches de co-évolution modèles/méta-modèles identifiées*

3.2. Identification des modifications de méta-modèles

Dans ce qui suit, les propositions existantes pour l'identification des modifications sont discutées. Le tableau III.3 résume les résultats.

3.2.1. Identification des modifications atomiques

- ✓ Les 04 approches qui permettent la collecte des modifications hors ligne permettent également la détection de la modification atomique.
- ✓ La plupart des approches proposent un calcul de la différence entre la version originale et la version évoluée du méta-modèle [24].
- ✓ L'approche MCL rejoigne l'identification des modifications atomiques avec les modifications complexes [85].

3.2.2. Identification des modifications complexes avec la collection hors ligne

- ✓ Nous avons trouvé 02 approches qui traitent l'identification des modifications complexes automatique ou semi-automatique : COPE / Edapt et Garcés et Al [11].
- ✓ En revanche, l'approche COPE / Edapt soutienne explicitement l'utilisateur à identifier manuellement des modifications complexes [11].
- ✓ Les approches restantes utilisent des modifications complexes, mais ne précisent pas les mécanismes d'identification appropriés [67].
- ✓ L'identification des modifications complexes est basée principalement sur la détection, employant des modèles pour définir quelles constellations des modifications atomiques se conforment à une modification complexe. comme dans l'approche Garcés et Al, elle se base sur des «***heuristiques***» pour identifier les modifications complexes [57].

Enfin, une approche ne fournit pas de mécanisme supplémentaire pour identifier les modifications complexes, mais réutilise le mécanisme de transformation du langage ou du formalisme pour l'application des règles : le formalisme sous-jacent des graphes dans MCL. Bien que cette approche semble naturelle, elle réduit le contrôle, puisque ces mécanismes réutilisés n'incluent pas les concepts d'hiérarchisation entre les modifications complexes contradictoires, et ne supporte pas les choix manuels [85]. Les deux approches : Garcès et Al et COPE / Edapt

permettent la correction manuelle et l'enrichissement de l'ensemble des modifications complexes détectées [57].

- ✓ Une seule approche considère une éventuelle incomplétude de la liste de modifications atomiques dérivées de la différenciation du modèle : l'approche présentée par Garcés et Al permet la détection des modifications complexes, même si certaines parties de ses effets sont cachés par d'autres modifications [57].

3.2.3 Identification des modifications complexes avec la collecte en ligne

- ✓ Une seule approche permet de l'identification des modifications complexes supplémentaires : COPE / Edapt permet à l'utilisateur de corriger manuellement les listes de modification [70].
- ✓ Un support explicite pour les opérations d'annulation est donné pour soulager l'utilisateur de rechercher manuellement ces opérations dans la liste des modifications atomiques [70].
- ✓ Cependant, jusqu'à présent il n'existe aucune approche pour la détection de modification complexe qui profite de la connaissance de l'ordre chronologique des modifications [70].

Approches	**Identification des modifications de méta-modèle**			
	Identification des modifications atomiques	***Degré d'automatisation***	***Identification des modifications complexe***	***Degré d'automatisation***
Cicchetti et Al	Oui	Automatisé	-	-
COPE / Edapt	Oui	Automatisé	Oui	Manuelle
Epsilon Flock	-	-	-	-
Garcés et Al	Oui	Semi-automatique	Oui	Semi-automatique
Gruschko et Al	-	-	-	-
MCL	Oui	Automatisé	-	-
Wachsmuth	-	-	-	-

Tableau III.3 : *Identification des modifications de méta-modèle (atomique et complexe) dans les approches de co-évolution modèles/méta-modèles identifiées*

3.3. Résolution des modèles

Le tableau ci-dessous résume les résultats. Notez que, dans ce tableau, certaines approches n'ont que des champs indiqués par «-». C'est le cas lorsque les approches ne prennent pas en charge la résolution des modèles directement.

3.3.1. Ordre de résolution des modèles

- ✓ La plupart des approches ne spécifient pas explicitement un ordre de résolution [33].
- ✓ Cependant, COPE / Edapt implique un ordre chronologique lorsque les modifications ont été recueillies en ligne. Pour les modifications qui ont été collectées en mode hors ligne, un ordre spécifié par l'utilisateur est appliqué [70].
- ✓ En revanche, Cicchetti et Al propose l'utilisation d'un ordre optimisé et présente une approche pour analyser les dépendances entre les modifications pour récupérer un ordre pour la résolution [9].

Approches	**Ordre de résolution des modèles**		
	Chronologique	***Optimisé***	***Aléatoire***
Cicchetti et Al	Non	Oui	Non
COPE/Edapt	Oui (en ligne)	Non	Oui (hors ligne)
Epsilon Flock	-	-	-
Garcés et Al	-	-	-
Gruschko et Al	-	-	-
MCL	-	-	-
Wachsmuth	-	-	-

Tableau III.4 : *Ordre de résolution des modèles dans les approches de co-évolution modèles / méta-modèles identifiées*

3.3.2. La catégorie de spécification des stratégies de résolution des modèles 0: n

La résolution de modèle se produit dans la catégorie 0: n, lorsque aucune décision manuelle n'est nécessaire pour dériver une stratégie de résolution pour une modification de méta-modèle.

- ✓ Les approches MCL et Epsilon Flock ne prennent pas en charge les solutions dans la catégorie 0: n [70].
- ✓ Cependant, l'approche Epsilon Flock permet un nettoyage automatisé en supprimant automatiquement les éléments du modèle qui ne sont plus conformes à la nouvelle version du méta-modèle [61], [60].

  Le nettoyage est le résultat de la technique de résolution choisie, où les éléments du modèle qui doivent rester dans le modèle doivent être manipulés explicitement. En conséquence, les éléments non conformes sont implicitement supprimés [5].
- ✓ De plus, pour MCL il est affirmé, sans fournir d'explications supplémentaires, que des cas simples spéciaux de modifications atomiques sont résolus automatiquement [85].
- ✓ Garcés et Al est la seule approche qui propose de générer une seule stratégie de migration soutenue [57].
- ✓ COPE / Edapt est une approche non basée sur l'opérateur avec des stratégies de résolution prédéfinies, mais prise en charge de la catégorie 0: n avec des solutions uniques par modification de méta-modèle ou opérateur «***stratégie prédéfinie***» [70].

Approches	**La catégorie de spécification des stratégies résolution des modèles 0 :n**	
	Une seule solution	***Plusieurs solutions***
Cicchetti et Al	Oui	Non
COPE / Edapt	Non	Oui
Epsilon Flock	Oui	Non
Garcés et Al	Oui	Non
Gruschko et Al	Oui	Non
MCL	Oui	Non
Wachsmuth	Oui	Non

Tableau III.5 : *Résolution des modèles dans la catégorie 0:n pour chaque approches de co-évolution modèle /méta-modèles identifiées*

3.3.3. La catégorie de spécification des stratégies résolution des modèles 1: n

Une résolution se produit dans la catégorie 1: n, lorsque après une modification de méta-modèle, une décision humaine est nécessaire pour dériver ou configurer une stratégie de résolution qui peut ensuite être utilisée pour résoudre tous les modèles à co-évoluer.

- ✓ La spécification libre des stratégies de migration est soutenue par toutes les approches suivantes : MCL [92], Epsilon Flock [60], Gruschko et Al [15] et COPE / Edapt [70].
- ✓ Enfin, certaines approches reposent sur des manipulations restreintes des stratégies de résolution. Là, des paramètres prédéfinis doivent être configurés. Cette configuration de stratégie par évolution de méta-modèle est soutenue par Wachsmuth [35], Cicchetti et Al [9] et Gruschko et Al [15].

Approches	**La catégorie de spécification des stratégies résolution des modèles 1 :n (Plusieurs solutions)**	
	Spécification de la stratégie	***Configuration de la stratégie***
Cicchetti et Al	Non	Oui
COPE/Edapt	Oui	Non
Epsilon Flock	Oui	Non
Garcés et Al	-	-
Gruschko et Al	Oui	Oui
MCL	Oui	Non
Wachsmuth	Non	Oui

Tableau III.6 : *Résolution des modèles dans la catégorie 1:n pour chaque approches de co-évolution modèles / méta-modèles identifiées*

3.3.4. La catégorie de spécification des stratégies résolution des modèles 1: 1

Une résolution est en catégorie 1: 1, lorsque des décisions humaines sont requises pour chaque modèle.

- ✓ Deux approches permettent une configuration de stratégie qui est valide par modèle: Cicchetti et Al [11] et COPE / Edapt [70].

Approches	**La catégorie de spécification des stratégies résolution des modèles 1:1**	
	Plusieurs solutions	***Solutions manuelle***
Cicchetti et Al	Oui (Configuration de la stratégie)	Non
COPE/Edapt	Oui (Configuration de la stratégie)	Non
Epsilon Flock	-	-
Garcés et Al	-	-
Gruschko et Al	-	-
MCL	-	-
Wachsmuth	-	-

Tableau III.7 : *Résolution des modèles dans la catégorie 1:1 pour chaque approches de co-évolution modèles / méta-modèles identifiées*

3.4. Correction des résultats requis

Après la co-évolution des modèles, une vérification des résultats se fait pour confirmer si les modèles co-évolués sont sémantiquement correctes.

- ✓ Par exemple, lorsqu'une restriction de type est appliquée à une propriété dans un méta-modèle, la solution générique implémentée dans l'approche COPE/Edapt supprime les instances de propriétés qui ne sont plus conformes [36].
- ✓ En revanche, l'approche de Wachsmuth permet à l'utilisateur de spécifier une conversion de type [61].

3.4.1. Exactitude des résultats requis

En raison de l'observation selon laquelle les approches permettent des résultats distinctes et différents niveaux de contrôle, Il faut vérifier la conformité syntaxique de méta-modèle co-évolué.

- ✓ Si c'est suffisant, toutes les approches présentées ci-dessus pourraient être utilisées.
- ✓ Toutefois, si une conformité syntaxique n'est pas suffisante, nous suggérons à l'utilisateur d'envisager des approches qui permettent un contrôle total du résultat.

3.4.2. Opportunités de généralisation de la résolution

Nous savons qu'il existe différents degrés de contrôle sous la forme des deux catégorie 1:n (contrôle de la stratégie, mais pas par modèle) et 1:1 (contrôle de la résolution par modèle). Lors de la sélection d'une approche nécessitant des résultats spécifiques, l'utilisateur doit donc considérer quel degré de contrôle est requis et de décider la possibilité de généraliser la résolution par modification de méta-modèle.

- ✓ Si une généralisation est possible, nous proposons d'utiliser l’approche qui correspond aux critères énumérés ci-dessus et permet une spécification sans restriction des stratégies de résolution pour les modifications de méta-modèle: COPE / Edapt [70].

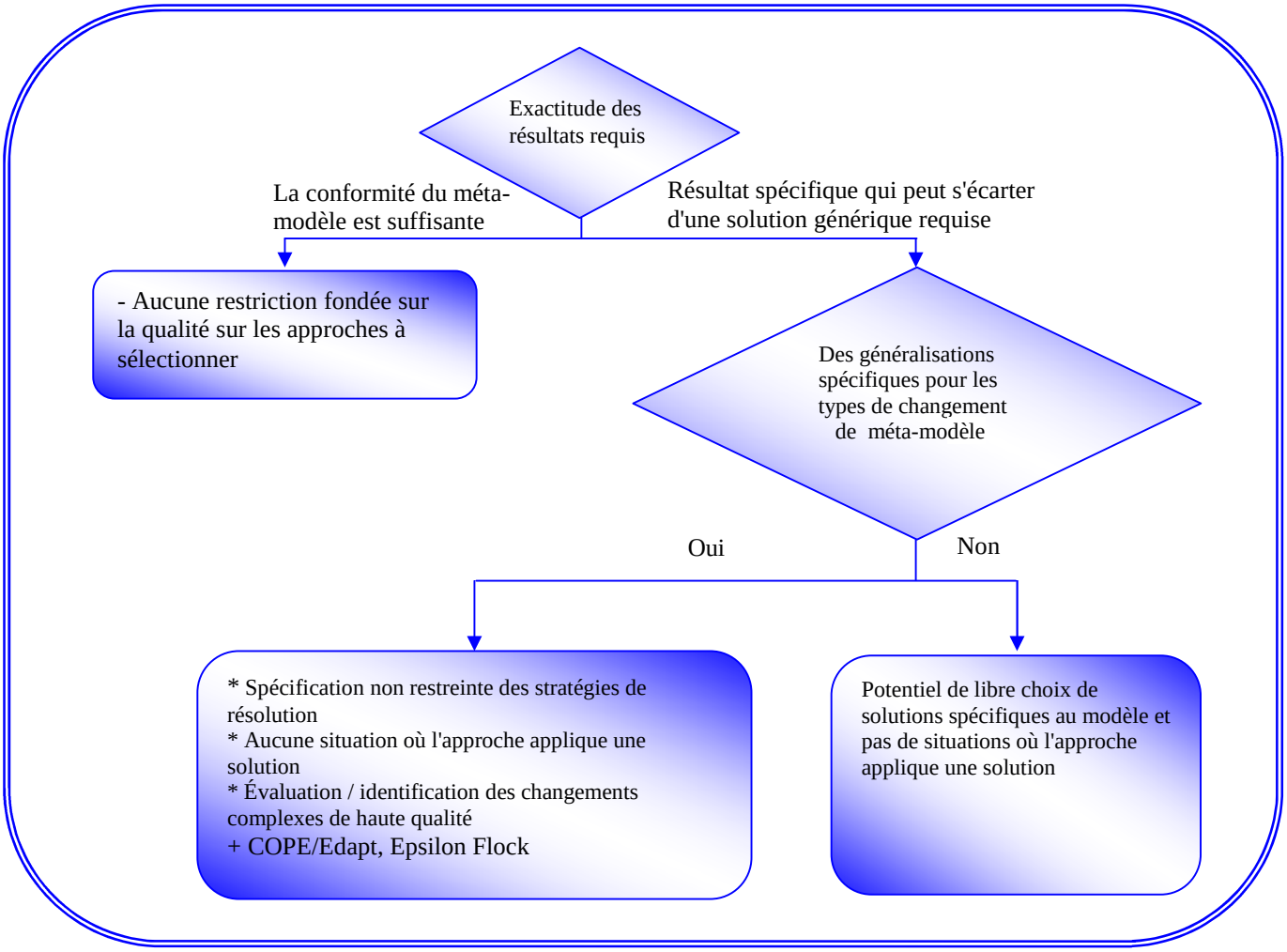


Figure III.1 : *Exactitude des résultats requis [70]*

3.5. Automatisation

- ✓ Les approches qui devraient être adoptées sont celles qui fournissent une automatisation complète pour l'identification des modifications et comprennent des résolutions dans la catégorie 0:n. Ce sont les approches de Garcés et Al [57], COPE / Edapt [70].

3.5.1. Soutien d'automatisation des tâches manuelles

Comme mentionné ci-dessus, les tâches manuelles lors de la co-évolution déterminent les coûts de fonctionnement de l'application d'une approche. Par conséquent, même si une automatisation complète n'est pas nécessairement prioritaire sur l'exactitude, lorsque les tâches manuelles sont prises en considération, nous proposons de considérer la nécessité de

réduire l'effort manuel. Alors, l'utilisateur devrait prendre en compte des démarches qui aident à réduire l'effort manuel.

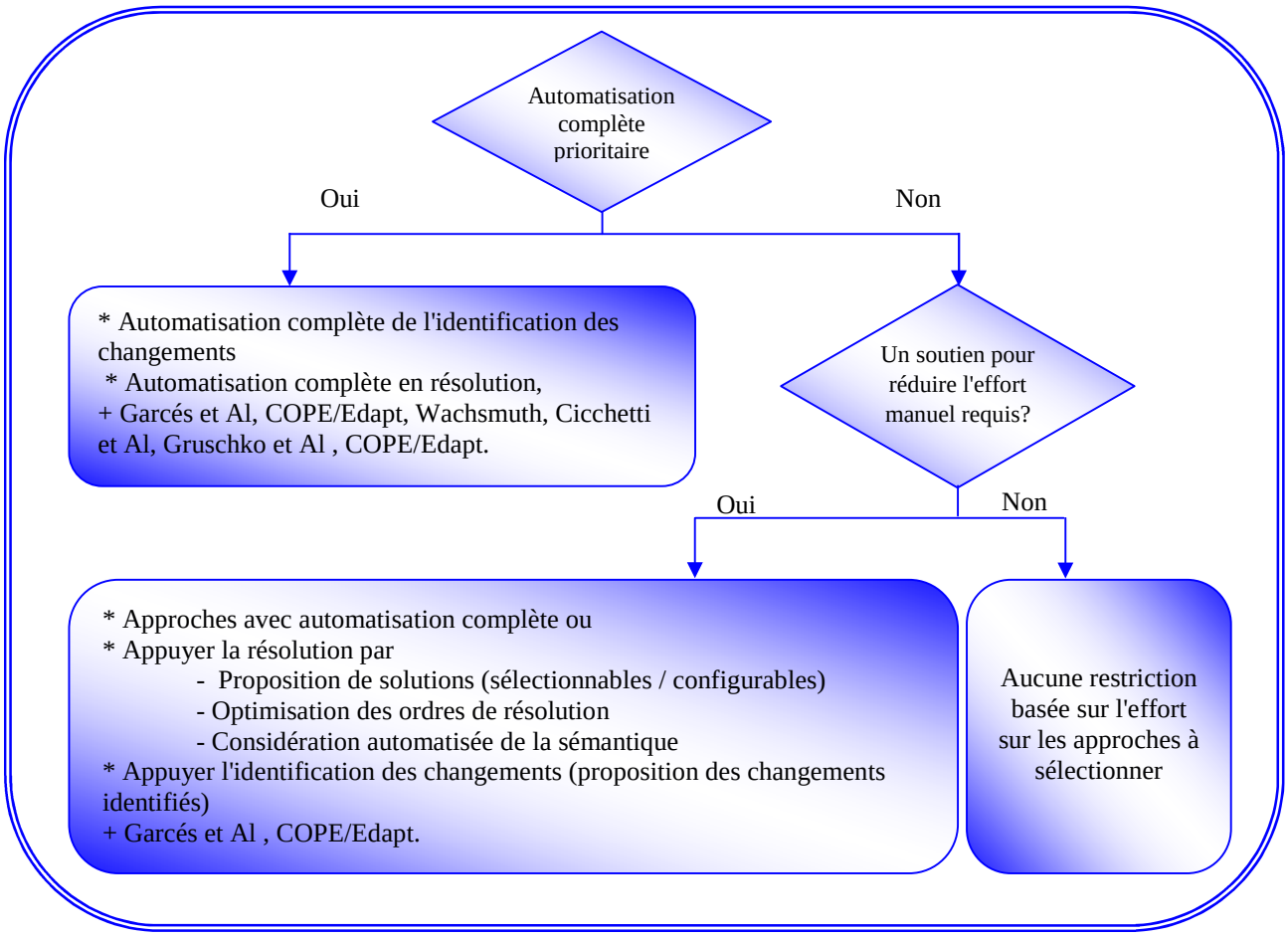


Figure III.2 : *Automatisation et assistance pour les tâches manuelles [70]*

Ces stratégies peuvent être trouvées dans les approches suivantes (comme le montre la figure III.2): Garcés et Al [57], COPE / Edapt [70].

3.6. Outils et mise en œuvre des approches de co-évolution modèles/méta-modèles

- ✓ Lors de l'adoption d'une approche, il faut décider de construire un outil propriétaire ou de réutiliser un outil existant.
- ✓ Le critère de sélection des approches devrait être que les implémentations sont déjà disponibles.

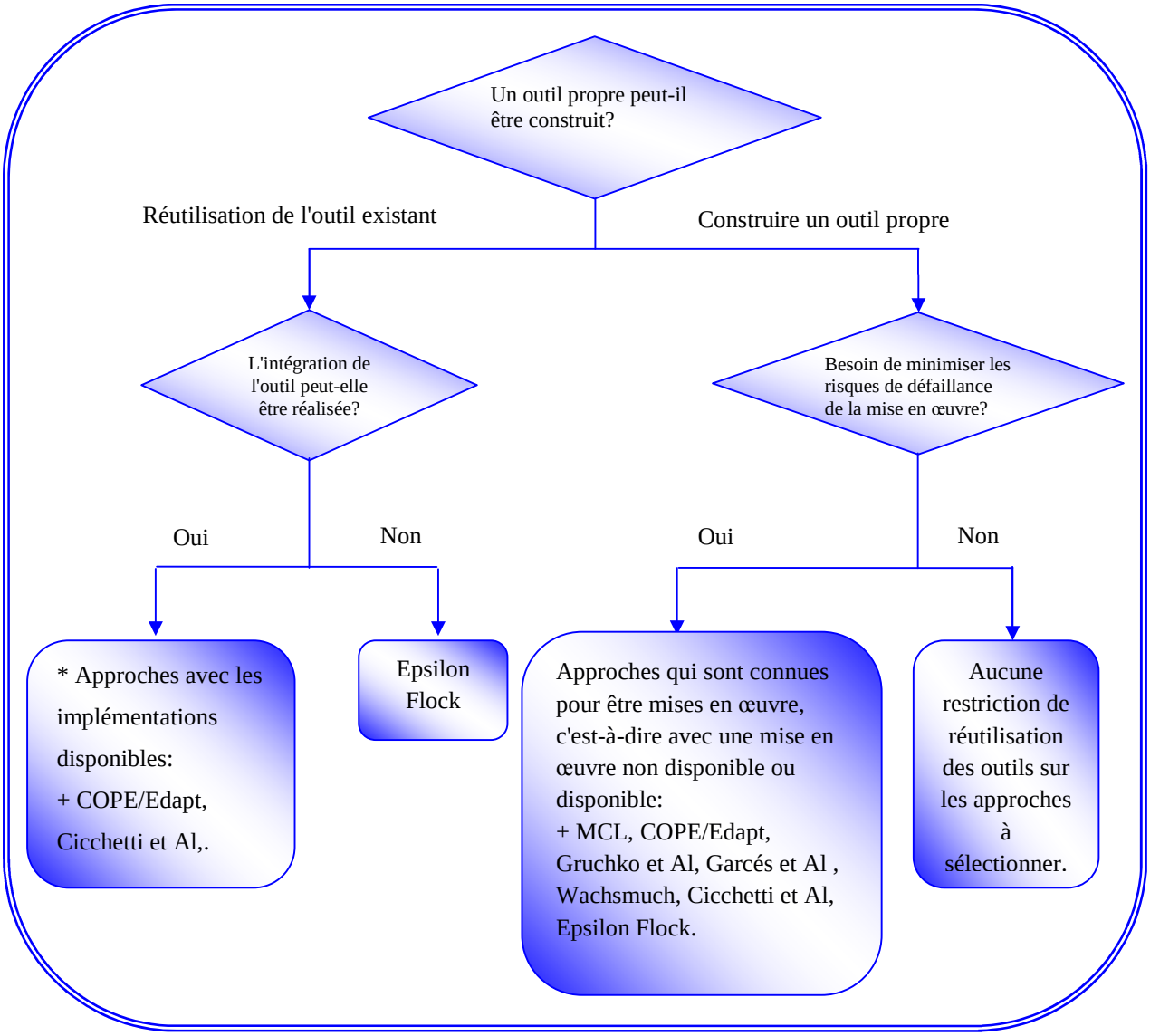


Figure III.3 : *Support d'outils [70]*

3.6.1. Intégration des outils des approches de co-évolutions modèles/méta-modèles

Si une intégration peut être construite, il est possible d'utiliser toutes les approches pour lesquelles le critère susmentionné (les implémentations sont accessibles au public) est rempli. Cependant, si l'effort de construction d'une intégration n'est pas souhaité, nous proposons de mettre l'accent sur les approches : COPE / Edapt [53], Cicchetti et Al [11].

3.6.2. Prise en charge des outils et risque de première mise en œuvre

La création d'un outil pour une approche de recherche peut être risquée, en particulier lorsque l'approche est de nature théorique et n'a pas été mise en œuvre auparavant.

Si cela est nécessaire, nous proposons à l'utilisateur de considérer des approches qui sont au moins connues pour être mises en œuvre, c'est-à-dire pour lesquelles une implémentation est publiquement disponible ou au moins est mentionnée ou décrite dans les publications correspondantes. Ainsi, une des approches suivantes pourrait être utilisée: MCL [85], Epsilon Flock [60], Garcés et Al [57], Wachsmuth [35], Cicchetti et Al [9], Gruschko et Al [15], ou COPE/Edapt [70].

3.7. Contrôle du processus de co-évolution

Lorsqu'un méta-modèle utilisé est changé en dehors d'une entreprise (par exemple : comme pour les normes), les approches basées sur la détection de modification en ligne ne sont souvent pas applicables [70].

Par conséquent, si la modification de méta-modèle n'est pas sous le contrôle de l'entreprise, la collecte de modifications en ligne pourrait ne pas se produire [37].

Dans le cas où les modifications de méta-modèle ne sont pas effectuées sous le contrôle de l'entreprise ou les résultats de la collection de modification en ligne ne seront pas disponibles, l'identification par modification ne peut être effectuée qu'avec des approches ne tiennent pas compte la collecte de modifications en ligne.

Cela signifie que les approches suivantes qui utilisent l'identification de modification hors ligne peuvent être utilisées: MCL [85], Garcés et Al [57], Cicchetti et Al [9], COPE/Edapt [70].

3.8. Compétences de la personne qui change le méta-modèle

Les approches de collecte de modification intrusive de l'interface utilisateur exigent que l'utilisateur qui modifie le méta-modèle soit conscient d'un maximum de 60 opérateurs et plus pour les appliquer strictement, bien que des moyens plus simples et de changer le méta-modèle dans la manière prévue pourrait exister (par exemple en utilisant des opérateurs atomiques au lieu de complexes). Cependant, les erreurs dans l'utilisation de ces approches

conduisent à une mauvaise identification des modifications et, ce qui conduit à une correction plus faible du résultat [6].

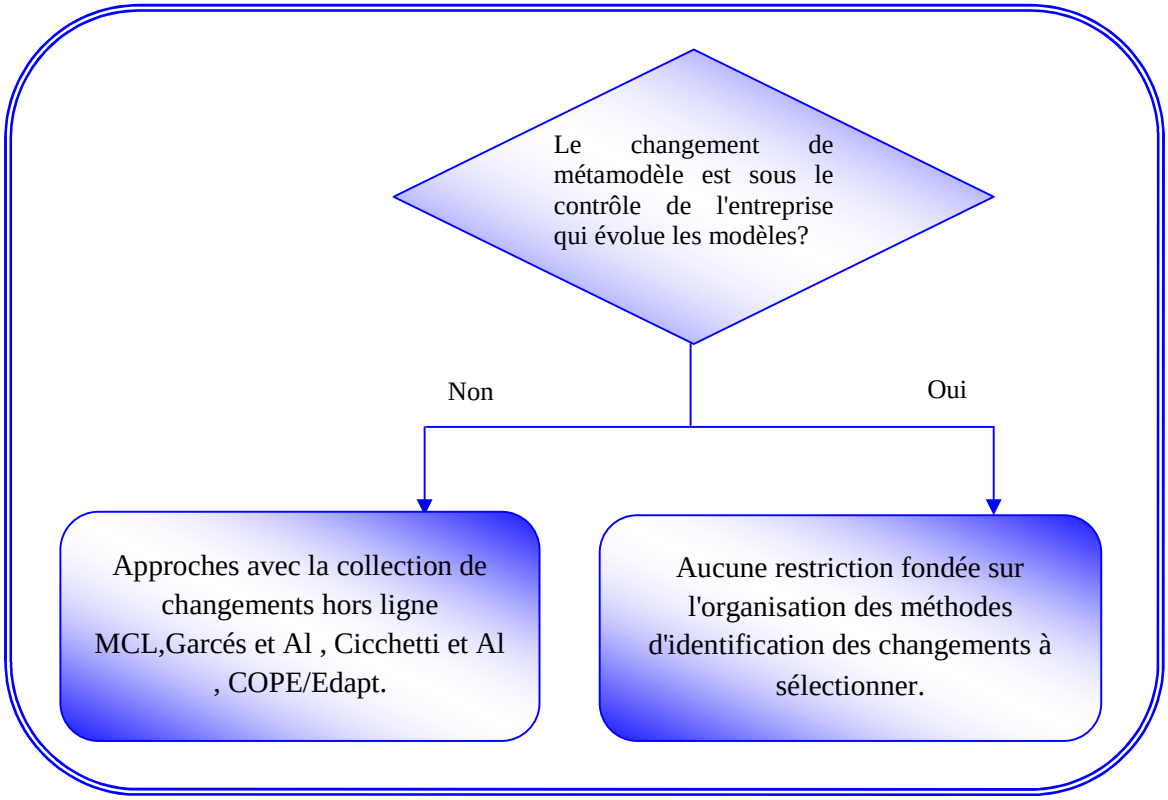


Figure III.4 : *Contrôle du processus de co-évolution des modèles/méta-modèles [6]*

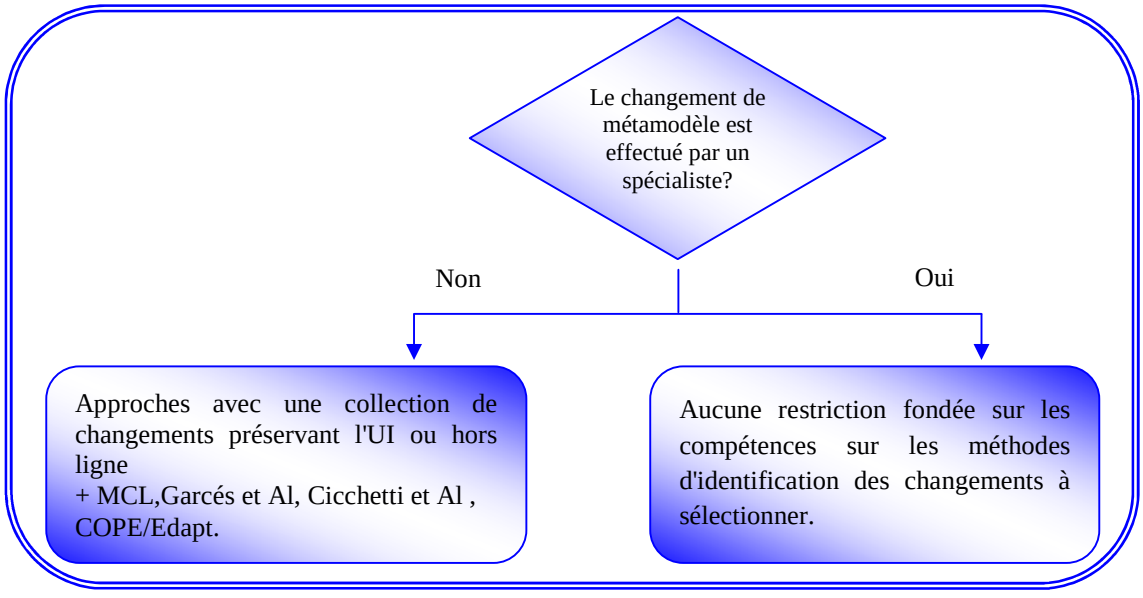


Figure III.5 : *Compétences de la personne qui change le méta-modèle [6]*

En conséquence, les approches suivantes qui sont basées sur la collecte de modifications hors ligne ou UI peuvent être appliquées: MCL [85], Garcés et Al [57], Cicchetti et Al [9], COPE/Edapt [70].

3.9. Compétences de la personne qui co-évolue les modèles

Un expert de domaine qui doit spécifier des stratégies de résolution n'est pas nécessairement un spécialiste des technologies de résolution de modèles et un développeur qui effectue la migration du modèle n'est pas nécessairement entièrement familier avec les modèles d'origine.

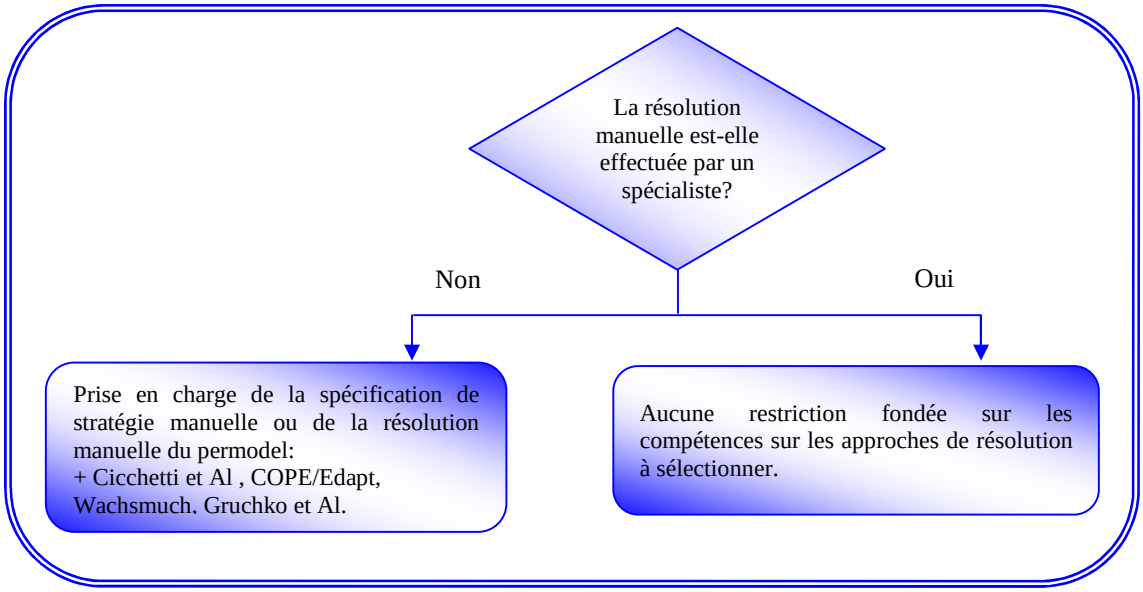


Figure III.6 : *Compétences de la personne qui co-évolue les modèles [70]*

Les approches suivantes remplissent au moins l'un des deux critères : Cicchetti et Al [9], COPE/Edapt [70].

CONCLUSION

Dans ce chapitre, nous avons présenté un sondage sur les approches qui soutiennent la co-évolution du modèle / méta-modèle. 07 approches de la littérature ont été étudiées et classées. Sur la base des idées de l'étude, nous avons présenté un soutien à la décision qui peut être utilisé par les praticiens pour choisir les approches de co-évolution qui sont de bons choix pour leur cas. Les besoins en automatisation et en contrôle des utilisateurs sont pris en compte.

CONCLUSION GENERALE

De nombreuses approches ont déjà été proposées pour traiter le problème de la co-évolution. Chacune d'entre elles prétendait utiliser des techniques et des outils propres à gérer l'adaptation d'une classe spécifique d'artefacts de modélisation. Malgré le problème est un domaine de recherche actif et un certain nombre de solutions ont été proposées, plusieurs difficultés exigent toujours d'être atténuées.

L'objectif principal de ce mémoire est de mettre la lumière sur le méta-modèle et la co-évolution du modèle. Tout d'abord, nous avons étudié les approches existantes qui traitent la migration des modèles en réponse à l'évolution du méta-modèle. Nous avons parlé des différentes catégorisations de ces approches. Ensuite, nous avons sélectionné certains critères pour évaluer les approches étudiées, après le processus d'analyse et de comparaison est exécuté.

Par conséquent, cette analyse a permis de définir des lignes directrices pour résoudre le problème de la co-évolution modèle / méta-modèle avec plus d'expressivité et de clarté et de soutenir les changements et l'extensibilité de la stratégie de la co-évolution pour en assurer l'exactitude.

Les travaux futurs se réfèrent à l'extension et au raffinement du cadre et à son application à diverses méthodes de méta-modalisation pour permettre la sélection de l'approche appropriée pour des tâches spécifiques et pour raisonner les lacunes manquantes dans ces approches.

Nous savons que l'évolution des méta-modèles nécessite également la co-évolution d'autres artefacts, tels que des transformations ou des contraintes. Nos connaissances sur la collecte et l'identification des changements de méta-modèle peuvent certainement être utiles pour d'autres cas de co-évolution sur les méta-modèles. À l'avenir, il pourrait être intéressant d'étudier les similitudes entre les techniques de résolution pour différents artefacts. Par exemple, il sera utile de comparer les différentes approches des contraintes du méta-modèle et de la co-évolution de la transformation du méta-modèle face à la résolution et s'il existe des similitudes avec les stratégies de résolution du modèle. En outre, il serait intéressant d'étudier si ces techniques de co-évolution pourraient être partiellement réutilisées pour des cas d'évolution logicielle, c'est-à-dire lorsque différents artefacts logiciels, tels que des modèles et des codes co-évoluent.

LISTE DES ACRONYMES

Acronyme	*Signification*
AML	**A**tlanMod **M**atching **L**anguage
API	**A**pplication **P**rogramming **I**nterface
ATL	**A**tlas **T**ransformation **L**anguage
AUTOSAR	**AUT**omotive **O**pen **S**ystem **Ar**chitecture
CAMPaM	**C**omputer **A**utomated **M**ulti-**Pa**radigm **M**odeling de l'Université McGill
CIM	**C**omputation **I**ndependent **M**odel "Le Modèle indépendant du calcul"
DSL	**D**omain **S**pecific **L**anguage
EAST-ADL	**A**rchitecture **D**escription **L**anguage, Est un langage de description d'architecture (*ADL*) pour les systèmes embarqués automobiles, développé dans plusieurs projets de recherche européens
Eclipse GMT	**Eclipse G**enerative **M**odel **T**ransformer
EMF	**E**clipse **M**odeling **F**ramework
EMF Compare	**E**clipse **M**odeling **F**ramework **Compare**
EUGENIA	http://www.eclipse.org/epsilon/doc/eugenia/ *EuGENia* est un outil qui génère automatiquement les modèles *.gmfgraph*, *.gmftool* et *.gmfmap* nécessaires à la mise en œuvre d'un éditeur GMF à partir d'une seule métaphore Ecore annoté
EVL	**E**psilon **V**alidation **L**anguage
FUJABA	*Fujaba* Tool Suite (bientôt: *Fujaba*): est un outil open source CASE permettant aux développeurs de prendre en charge l'ingénierie logicielle et la réingénierie modèle
Git	*Git* est un système de contrôle de version distribué gratuit et open source conçu pour gérer tout, de petits à très grands projets avec rapidité et efficacité. https://git-scm.com
GME	**G**eneric **M**odeling **E**nvironment, l'environnement de modélisation générique
GMF	**G**raphical **M**odeling **F**ramework
GenMyModel7	*GenMyModel* est un outil UML en ligne pour les diagrammes de classes et utilise des diagrammes de cas. https://www.genmymodel.com/fr
Graphiti	Infrastructure d'outils graphiques. ***Graffiti*** est un cadre graphique basé sur Eclipse qui permet un développement rapide des éditeurs de diagrammes à la fine pointe de la technologie pour les modèles de domaine

HOT	**H**igher **O**rder **T**ransformation
IDM	**I**ngénierie **D**irigée par **M**odèles
IBM	**I**nternational **B**usiness **M**achines
JMI	**J**ava **M**etadata **I**nterface
MCL	**M**odel **C**hange **L**anguage
MDA	**M**odel **D**riven **A**rchitecture **"A**rchitecture **D**irigée par les **M**odèles"
MDD	**M**odel **D**riven **D**evelopment
MDE	**M**odel **D**riven **E**ngineering "equivalent anglais de l'IDM"
MDSD	**M**odel **D**riven **S**oftware **D**evelopment
MIC	**M**odel-**I**ntegrated **C**omputing de Vanderbilt
MOF	**M**eta **O**bject **F**acility
MONDP FP7 de l'UE	(Union Européenne) http://www.mondo-project.org.
NoSQL	**N**ot **O**nly **SQL**
OCL	**O**bject **C**onstraint **L**anguage
OMG	**O**bject **M**anagement **G**roup
PIM	**P**latform **I**ndependent **M**odel "Modèle indépendant de la plateforme"
PSM	**P**latform **S**pecific **M**odel "Modèle spécifique à la plateforme"
QVT	**Q**uery **V**iew **T**ransformation
SiDiff	*SiDiff* est un outil de différence universel pour les fichiers texte de toutes sortes. Il se concentre actuellement sur les fichiers de code source et les fichiers texte, mais ne se limite pas.
Similarity Flooding	L'algorithme **SF** décri les approches existantes concernant le problème d'appariement
SQL-like	L'opérateur *LIKE* est utilisé dans la clause WHERE des requêtes SQL
Subversion	Est un système de contrôle de version distribué gratuit et open source conçu pour gérer tout, de petits à très grands projets avec rapidité et efficacité
UML	**U**nified **M**odeling **L**anguage
WFRs	**W**ell-**F**ormedness **R**ule**s**
XMI	**XML M**etadata **I**nterchange
XML	e**X**tensible **M**arkup **L**anguage
XSLT	e**X**tensible **S**tylesheet **L**anguage **T**ransformations

REFERENCES

[1] Abdelmalek Amine, Ladjel Bellatreche, Zakaria Elberrichi, Erich J. Neuhold, et Robert Wrembel. Computer Science and Its Applications - 5th IFIP TC 5 International Conference, CIIA 2015, Saida, Algeria, May 20-21, 2015, Proceedings. IFIP Advances in Information and Communication Technology 456, Springer, ISBN 978-3-319-19577-3, (2015).

[2] Adrian Rutle. Diagram Predicate Framework: A Formal Approach to MDE. Thèse de doctorat, University of Bergen Norway, (2010).

[3] Alessandro Rossini. Diagram Predicate Framework meets Model Versioning and Deep Metamodelling. Thèse de doctorat, University of Bergen Norway, (2011).

[4] Alin Deutsch et Val Tannen. MARS. A System for Publishing XML from Mixed and Redundant Storage. International Journal on Very Large Data Bases, pages 201-212, (2003).

[5] Anantha Narayanan, Tihamer Levendovszky, Daniel Balasubramanian, et Gabor Karsai. Automatic Domain Model Migration to Manage Metamodel Evolution, MoDELS, pages 706-711, (2009).

[6] Andreas Demuth, Markus Riedl-Ehrenleitner, Roberto E. Lopez-Herrejon, et Alexander Egyed. Co-evolution of metamodels and models through consistent change propagation. Journal of Systems and Software 111, pages 281-297, (2016).

[7] Antoine Beugnard, Fabien Dagnat, Sylvain Guérin, et Christophe Guychard. Des situations de modélisation pour évaluer les outils de modélisation, INFORSID, pages 181-197, (2014).

[8] Antonio Cicchetti, Davide Di Ruscio, Alfonso Pierantonio. Managing Model Conflicts in Distributed Development, MODELS, pages 311-325, (2008).

[9] Antonio Cicchetti, Davide Di Ruscio, et Alfonso Pierantonio. Managing Dependent Changes in Coupled Evolution, ICMT, pages 35-51, (2009).

[10] Antonio Cicchetti, Davide Di Ruscio, et Alfonso Pierantonio. Model Patches in Model-Driven Engineering, MODELS Workshops, pages 190-204, (2009).

[11] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo et Alfonso Pierantonio. Metamodel differences for supporting model co-evolution. Model-Driven Software Evolution-MODSE, (2008).

[12] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, et Alfonso Pierantonio. Automating Co-evolution in Model-Driven Engineering, EDOC, pages 222-231, (2008).

[13] Antonio Cicchetti, F. Ciccozzi, T. Leveque et Alfonso Pierantonio. On the concurrent Versioning of Metamodels and Models: Challenges and possible Solutions. Workshop on Model Comparison in Practice, ACM SIGSOFT, (Juin 2011).

[14] Antonio Cicchetti. Difference Representation and Conflict Management in Model-Driven Engineering. Thèse de doctorat, University of Aquila, (2008).

[15] B. Gruschko, D. Kolovos et R. Paige. Towards synchronizing models with evolving metamodels. Proceedings of the International Workshop on Model-Driven Software Evolution, (2007).

[16] Barbara Staudt Lerner. A model for compound type changes encountered in schema evolution. ACM Trans. Database Syst. 25(1), pages 83-127, (2000).

[17] Bashar Nuseibeh, Steve M. Easterbrook, et Alessandra Russo. Making inconsistency respectable in software development. Journal of Systems and Software 58(2), pages 171-180, (2001).

[18] Christian Krause, Johannes Dyck, et Holger Giese. Metamodel-Specific Coupled Evolution Based on Dynamically Typed Graph Transformations, ICMT, pages 76-91, (2013).

[19] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, et Björn Regnell. Experimentation in Software Engineering. Springer, ISBN 978-3-642-29043-5, pages I-XXIII, 1-236, (2012).

[20] D. C. Schmidt. Guest Editor's Introduction. Model-Driven Engineering. Computer, vol 39, No 2, pages 25-31, (2006).

[21] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, et Ed Merks. EMF, Eclipse Modeling Framework 2.0 (2nd Edition), (2008).

[22] David Méndez, Anne Etien, Alexis Muller et Rubby Casallas. Transformation migration after metamodel evolution. In Proc, ME Workshop, (2010).

[23] Davide Di Ruscio, Ludovico Iovino, et Alfonso Pierantonio. A Methodological Approach for the Coupled Evolution of Metamodels and ATL Transformations, ICMT, pages 60-75, (2013).

[24] Davide Di Ruscio, Ludovico Iovino,et Alfonso Pierantonio. Evolutionary Togetherness: How to Manage Coupled Evolution in Metamodeling Ecosystems, ICGT, pages 20-37, (2012).

[25] Davide Di Ruscio, Ralf Lämmel, et Alfonso Pierantonio. Automated co-evolution of GMF editor models. CORR abs/1006.5761, pages 143-162, (2010).

[26] Dimitrios S. Kolovos, Nicholas Drivalos Matragkas, Horacio Hoyos Rodriguez, et Richard F. Paige. Programmatic Middle Management. XM@ MODELS, pages 2-10, (2013).

[27] Dimitrios S. Kolovos, Richard F. Paige, et Fiona A. C. Polack. On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages. Rigorous Methods for Software Construction and Analysis, pages 204-218, (2009).

[28] E. Gamma, R. Helm, R. Johnson et J. Vlissides. Design patterns. Elements of reusable object-oriented software, (1994).

[29] Erhard Rahm, et Philip A. Bernstein. A survey of approaches to automatic schema matching. VLDB J. 10(4), pages 334-350, (2001).

[30] F. Anguel, 'Université Chadli Bendjedid', A. Amirat, Université Mohammed Chérif Messaadia', et N. Bounour, 'Laboratoire LISCO, Université Badji Mokhtar'. Comparison Study of Metamodels and Models Co-Evolution Approaches, Symposium on Complex Systems and Intelligent Computing (CompSIC) University of Souk Ahras - Université Mohamed Chérif Messaadia de Souk-Ahras, (2014).

[31] Florian Mantz et Yngve Lamo ' Université de Bergen College', Alessandro Rossini et Uwe Wolter 'Université de Bergen', Gabriele Taentzer '-Université de Philipps Marburg', Formalising Metamodel Evolution based on Category Theory, (2011).

[32] Florian Mantz. Syntactic Quality Assurance Techniques for Software Models. Thèse de doctorat, Philipps University in Marburg, Germany, (Aout 2009).

[33] Fouzia Anguel, Abdelkrim Amirat, et Nora Bounour. Hybrid Approach for Metamodel and Model Co-evolution, CIIA, pages 563-573, (2015).

[34] Frank Budinsky et Stephen A Brodsky. Merks, eclipse modeling framework, (2003).

[35] Guido Wachsmuth. Metamodel adaptation and model coadaptation. Object-Oriented Programming, Springer, pages 600-624, (2007).

[36] Gabriele Taentzer, Florian Mantz, Thorsten Arendt, et Yngve Lamo. Customizable Model Migration Schemes for Meta-model Evolutions with Multiplicity Changes, MoDELS, pages 254-270, (2013).

[37] Hauke Wittern. Determining the Necessity of Human Intervention When Migrating Models of an Evolved DSL, EDOC Workshops, pages 209-218, (2013).

[38] Holger Giese, et Robert Wagner. Incremental Model Synchronization with Triple Graph Grammars. MoDELS, pages 543-557, (2006).

[39] http://www.alphaworks.ibm.com. visité le: 30 Avril 2017.

[40] http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/history_of_uml.htm. visité le: 20 Février 2017.

[41] http://www.omg.org. visité le: 20 Février 2017.

[42] http://www.omg.org. visité le: 25 Mars 2017.

[43] http://www.omg.org. visité le : 8 Avril 2017.

[44] http://www.omg.org. visité le: 25 Mars 2017.

[45] http://www.planetmde.org/. visité le: 15 Avril 2017.

46] http://www:omg:org. visité le : 29 Avril 2017.

[47] http://www:omg:org. visité le : 31 mars 2014.

[48] http://www:omg:org/mda. visité le: 29 Avril 2017.

[49] Jacky Estublier, David B. Leblang, André van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter F. Tichy, et Darcy Wiborg Weber. Impact of software engineering research on the practice of software configuration management. ACM Trans. Softw. Eng. Methodol. 14(4), pages 383-430, (2005).

[50] Jernej Kovse, et Theo Härder. Generic XMI-Based UML Model Transformations, OOIS, pages 192-198, (2002).

[51] John Edward Hutchinson, Jon Whittle, Mark Rouncefield, et Steinar Kristoffersen. Empirical assessment of MDE in industry, ICSE, pages 471-480, (2011).

[52] John F. Roddick. A survey of schema versioning issues for database systems. Information & Software Technology 37(7), pages 383-93, (1995).

[53] Jokin García, Oscar Díaz et Maider Azanza. Model transformation coevolution. A semi-automatic approach. Software Language Engineering Springer, pages 144-163, (2013).

[54] Joshua J. Bloch.How to design a good API and why it matters, OOPSLA Companion, pages 506-507, (2006).

[55] Kelly Garcés, Frédéric Jouault, Pierre Cointe et Jean Bézivin. A Domain Specific Language for Expressing Model Matching, (2009).

[56] Kahina Hassam, Salah Sadou, Vincent Le Gloahec, et Régis Fleurquin.Assistance System for OCL Constraints Adaptation during Metamodel Evolution, CSMR, pages 151-160, (2011).

[57] Kelly Garcés, Frédéric Jouault, Pierre Cointe, et Jean Bézivin. Managing Model Adaptation by Precise Detection of Metamodel Changes, ECMDA-FA, pages 34-49, (2009).

[58] Krzysztof Czarnecki, et Simon Helsen. Feature-based survey of model transformation approaches, IBM Systems Journal 45(3), pages 621-646, (2006).

[59] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, et Fiona A. C. Polack. Model Migration with Epsilon Flock, ICMT, pages 184-198, (2010).

[60] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, Fiona A. C. Polack, et Simon M. Poulding. Epsilon Flock: a model migration language. Software and System Modeling 13(2), pages 735-755, (2014).

[61] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, et Fiona A. C. Polack . Enhanced Automation for Managing Model and Metamodel Inconsistency, ASE, pages 545-549, (2009).

[62] Louis M. Rose, Esther Guerra, Juan de Lara, Anne Etien, Dimitris S. Kolovos, et Richard F. Paige. Genericity for model management operations. Software and System Modeling 12(1), pages 201-219, (2013).

[63] Louis M. Rose, Markus Herrmannsdoerfer, James R. Williams, Dimitrios S. Kolovos, Kelly Garcés, Richard F. Paige,et Fiona A. C. Polack. A Comparison of Model Migration Tools. MODELS (1), pages 61-75, (2010).

[64] Louis M. Rose. Structures and processes for managing model-metamodel co-evolution, Thèse de doctorat, University of York. UK. (2011).

[65] M. Koegel, M. Herrmannsdoerfer, Y. Li, J. Helming et J. David. Comparing state- and operation-based change tracking on models. Enterprise Distributed Object Computing Conference (EDOC. IEEE), pages 163-172, (2010).

[66] Marco Brambilla, Jordi Cabot, et Manuel Wimmer. Model-Driven Software Engineering in Practice. Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers, (2012).

[67] Mark van den Brand, Zvezdan Protic, et Tom Verhoeff. A Generic Solution for Syntax-Driven Model Co-evolution, TOOLS (49), pages 36-51, (2011).

[68] Markus Herrmannsdoerfer, Sander Vermolen, et Guido Wachsmuth. An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models, SLE, pages 163-182, (2010).

[69] Markus Herrmannsdoerfer, Sebastian Benz, et Elmar Jürgens. Automatability of Coupled Evolution of Metamodels and Models in Practice, MoDELS, pages 645-659, (2008).

[70] Markus Herrmannsdoerfer. COPE - A Workbench for the Coupled Evolution of Metamodels and Models, SLE, pages 286-295, (2010).

[71] Michael J. Butler, Luigia Petre, Kaisa Sere. Integrated Formal Methods, Third International Conference. IFM 2002. Turku. Finland. Proceedings. Lecture Notes in Computer Science 2335. Springer. ISBN 3-540-43703-7 [contents], (15-18 Mai 2002).

[72] Object Management Group (OMG). OMG Unified Modeling Language TM (OMGUML) - Version 2.5, (2015).

[73] Petra Brosch, Gerti Kappel, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer. An Introduction to Model Versioning, SFM, pages 336-398, (2012).

[74] Philip Langer, Manuel Wimmer, Petra Brosch, Markus Herrmannsdörfer, Martina Seidl, Konrad Wieland, et Gerti Kappel. A posteriori operation detection in evolving software models. Journal of Systems and Software 86(2), pages 551-56, (2013).

[75] R.L. Acko, John Wiley et Sons. Scientific Method: Optimizing Applied Research Decisions, (1962).

[76 Regina Hebig, Djamel Eddine Khelladi, et Reda Bendraou. Approaches to Co-Evolution of Metamodels and Models: A Survey. IEEE Trans. Software Eng. 43(5), pages 396-414, (2017).

[77] Richard Barker. CASE Method - Entity Relationship Modellierung. Addison-Wesley, ISBN 978-3-89319-397-4, pages I-XIV. 1-247, (1992).

[78] Richard F. Paige, Nicholas Drivalos Matragkas, et Louis M. Rose. Evolving models in Model-Driven Engineering: State-of-the-art and future challenges. Journal of Systems and Software 111, pages 272-280, (2016).

[79] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa et Klaus Lange. Autosar-a worldwide standard is on the road. In 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden, (2009).

[80] Soraya Mesli. Adaptation des contraintes à l'évolution de leur méta-modèle, these de doctorat .Aout 2013.

[81] Stecklein, J. M. Dabney, J.Dick, B. et al. Error Cost Escalation Through the Project Life Cycle. disponible sur http://ntrs:nasa:gov/archive/nasa/casi:ntrs:nasa:gov/20100036670:pdf. visité le : 12 Avril 2017.

[82] Steven Kelly, Juha-Pekka Tolvanen. Domain-Specific Modeling-Enabling Full Code Generation. Wiley. ISBN 978-0-470-03666-2-, pages I-XVI- 1-427, (2008).

[83] The Eclipse Foundation. MDT / OCL Project. http://projects.eclipse.org/projects/modeling.mdt.ocl. visité le: 07 Avril 2017.

[84] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, et Simon Helsen. Model-driven software development - technology, engineering, management. Pitman, ISBN 978-0-470-02570-3, pages I-XVI, 1-428, (2006).

[85] Tihamer Levendovszky, Daniel Balasubramanian, Anantha Narayanan, Feng Shi, Christopher P. van Buskirk,et Gabor Karsai. A semi-formal description of migrating domain-specific models with evolving domains. Software and System Modeling 13(2), pages 807-823, (2014).

[86] Vincent Debruyne, Françoise Simonot-Lion, et Yvon Trinquet. Eastadlan architecture description language. In Architecture Description Languages. Springer, pages 181-195, (2005).

[87] Wael Kessentini, Houari A. Sahraoui,et Manuel Wimmer. Automated Metamodel/Model Co-evolution Using a Multi-objective Optimization Approach. ECMFA, pages 138-155, (2016).