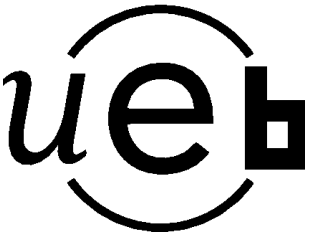
N° d'ordre : 4407 ANNÉE 2011



THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Mickaël CLAVREUL

préparée à l'unité de recherche IRISA – UMR6074
Institut de Recherche en Informatique et Systèmes Aléatoires

Composition de modèles et de métamodèles :

Séparation des correspondances et des interprétations pour unifier les approches de composition existantes

Thèse à soutenir à Rennes
le 7 décembre 2011

devant le jury composé de :

Philippe LAHIRE
Professeur à l'Université de Nice – Sophia Antipolis / *Rapporteur*

Siobhán CLARKE
Full Professor, Trinity College, Dublin / *Rapporteur*

Olivier RIDOUX
Professeur à l'Université de Rennes 1 / *Examinateur*

Olivier BEAUDOUX
Enseignant–Chercheur à l'ESEO / *Examinateur*

Jean-Marc JÉZÉQUEL
Professeur à l'Université de Rennes 1 / *Directeur de thèse*

Olivier BARAIS
Maître de Conférence à l'Université de Rennes 1 / *Co-directeur de thèse*

Acknowledgements

This thesis is the result of three years of work within the Triskell team which excellence in both research and atmosphere brings motivation and leads to active collaborations.

First of all, I would like to thank Jean-Marc Jézéquel who does me the honors to integrate the Triskell team. His support and efficient advice let me focus on the main topic of my thesis while I was free to explore alternatives. I am grateful that he lets me participate to first-class research that helped me to discover a vocation for myself as a scientist.

I would like also to thank Olivier Barais. While he is a very busy person, he took the necessary amount of time in helping me put the pieces of my thesis together and finalize the composition of this thesis.

My thanks go also to Benoit Combemale and Benoit Baudry with whom I participated in interesting developments of my work. Our discussions were always fruitful and gave birth to multiple tracks of research that we are still exploring.

I would like to thank the whole Triskell team and their members whom good-nature participates in the very pleasant work atmosphere.

I would like also to acknowledge in advance Olivier Ridoux that does the honors to take the chair of the jury along with Philippe Lahire and Siobhán Clarke who had accepted to review this work.

Within the MOPCOM-I project, I would like to thank the various partners and especially Christian Nicolas for the interesting feedback on the proposed tools and approaches.

Last but not least, I would like to thank Olivier Beaudoux about the discussions that we had at the beginning of these three years of work on the definition of mappings between models and the specific requirements that exist in the domain of graphical interfaces.

On a personal basis, I would like to thank my family and my friends for their support on an everyday basis.

Résumé en français

Introduction

Les logiciels font maintenant partie intégrante de notre société. La plupart des activités telles que l'économie, la finance, le transport ou encore les communications reposent sur des systèmes logiciels qui permettent de définir, de gérer, de modéliser, d'améliorer ou encore de mettre en valeur les activités humaines. L'omniprésence de la technologie amène à la définition de systèmes de plus en plus complexes, ce qui impacte fortement les techniques traditionnelles de développement logiciel : la composition de programmes, modules ou fonctions est au cœur de cycles de développement logiciel contrôlés et éventuellement géographiquement distribués dans lesquels participent de multiples acteurs.

La gestion voire la diminution de la complexité est une problématique récurrente à tout raisonnement scientifique. La décomposition d'un problème en sous-problèmes est la clé pour comprendre une situation donnée et proposer des solutions. Lorsque qu'une solution est disponible pour chaque sous-problème, on produit une représentation globale de la solution sur laquelle on peut raisonner. Le degré de séparation et de recomposition de ces sous–problèmes est le principe même de *modularité*.

L' Ingénierie Dirigée par les Modèles (IDM) est basée sur le principe d'*abstraction* qui imite le raisonnement scientifique dans le sens où, pour produire des logiciels de bonne qualité, les ingénieurs s'appuient sur des représentations abstraites partielles dédiées à la résolution d'un problème particulier. L'IDM propose de traiter les modèles comme des entités de premier ordre et encourage l'utilisation des modèles pour la production de systèmes et non plus seulement pour la documentation de ces mêmes systèmes. Les modèles sont alors au centre de processus de développement complexes visant à rendre le cycle de développement d'un logiciel plus efficace, moins cher et plus sûr.

Bien que l'application des principes de *modularité* et d'*abstraction* vise à gérer efficacement la complexité des développement logiciels, les équipes de développement font face à une augmentation du nombre de représentations partielles à manipuler. Pour pouvoir raisonner sur la construction, la validation ou encore la vérification des inconsistances d'un système en développement, il est nécessaire de disposer d'un moyen de recomposer les représentations partielles entre elles. Cette étape de recomposition est, par nature, complexe et chronophage et sujette à l'introduction involontaire d'erreurs. Dans le cadre de l'IDM, la composition de modèles est un domaine de recherche très actif qui s'intéresse à l'automatisation des tâches de recomposition des représentations partielles, autrement dit, des modèles.

Alors que de plus en plus de techniques sont proposées pour composer des modèles dans des contextes particuliers, la quasi-inexistence de consensus pour comparer les techniques existantes de composition de modèles et pour identifier des artéfacts réutilisables entraîne une explosion de l'effort nécessaire pour produire de nouveaux outils spécifiques de composition de modèles à partir de techniques existantes.

En pratique, il est nécessaire d'identifier les points communs aux techniques de

composition existantes pour non plus définir des opérateurs de composition spécifiques à un usage mais plutôt des opérations de composition paramétrables, s'appuyant sur des opérateurs existants, et traitant un large spectre d'usages.

La contribution principale de cette thèse est de proposer une **définition originale de la composition de modèles comme étant une paire correspondance-interprétation**. Une correspondance définit la similarité entre deux modèles ou plus à partir d'un ensemble de règles d'alignement entre des «patterns» d'éléments de modèles. Une interprétation représente à la fois l'intention du processus de composition de modèles ainsi que les exigences de l'utilisateur en termes de sous-produits de la composition de modèles. Cette définition de la composition de modèles permet d'identifier des correspondances et des interprétations réutilisables pour spécifier une multitude de techniques de composition de modèles. A partir de cette définition, nous proposons un cadre théorique qui aide à (1) unifier les représentations des techniques existantes de composition de modèles et à (2) automatiser les processus de développement d'outils de composition de modèles dédiés.

La pertinence et l'utilisabilité du cadre théorique implique la proposition de deux sous-contributions supplémentaires :

- Nous proposons un ensemble de **catégories pour classer les techniques de correspondance entre modèles et les interprétations existantes de ces correspondances entre modèles**. Ces catégories nous permettent de proposer une grille de lecture pour l'analyse et la comparaison des techniques existantes de composition de modèles.
- Nous proposons un **langage de modélisation spécifique inspiré des catégories pour la définition de correspondances génériques entre modèles et pour la définition d'interprétations**. Ce langage de modélisation outille la spécification et la construction de nouvelles approches pour la composition de modèles.

Les contributions proposées dans cette thèse ont été validées au travers de deux expérimentations principales : *(i)*les catégories de correspondances et d'interprétations ont été comparées en termes de précision et de pertinence à un ensemble significatif d'approches extraites de la littérature ; *(ii)*un prototype logiciel a été développé et utilisé dans le cadre du projet MOPCOM–I du pôle de compétitivité Images & Réseaux de la région Bretagne. La validation du langage de modélisation ainsi que l'approche globale de composition de modèles a été mise en œuvre sur un cas d'étude proposé par Technicolor pour l'intégration de librairies existantes dédiées à la gestion d'équipements numériques de diffusion vidéo.

Composition de Modèles et Ingénierie des Logiciels

L'état de l'art sur la composition de modèles dans le cadre de l'ingénierie du logiciel est décrit dans cette thèse au travers d'un processus en quatre étapes illustré en Figure 1. Nous proposons dans cette section *(i)*d'explorer l'état de la pratique du développement de logiciels de manière à faire apparaître comment la modularité, l'IDM et la composition de modèles participent à la construction de logiciels complexes ; *(ii)*de

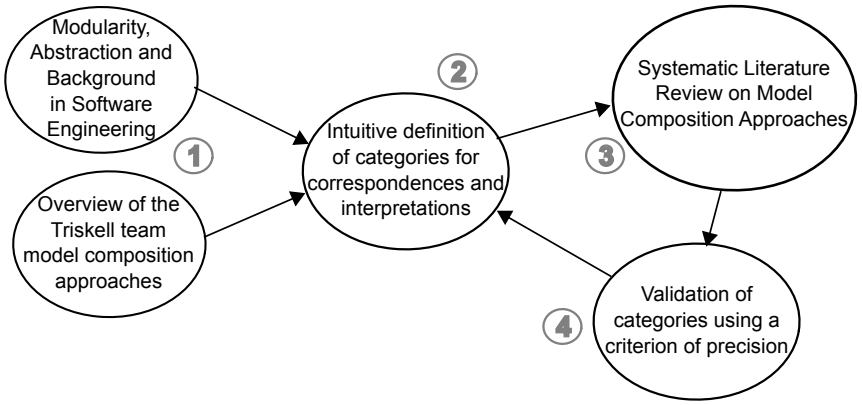


FIGURE 1 – Processus d'exploration de l'état de l'art de la composition de modèles

définir de manière intuitive les concepts principaux de la composition de modèles et ainsi construire une grille de lecture pour comparer les techniques de composition de modèles existantes ; *(iii)*de réaliser une étude systématique de la littérature pour valider les concepts principaux et ainsi proposer un cadre théorique unifié visant à améliorer la réutilisation et l'adaptation des techniques de composition de modèles.

Modularité, Abstraction et Composition de Modèles

La construction de modèles dans le cadre de l'IDM conduit à la production de langages dédiés et de modèles spécifiques qui facilitent la spécification de systèmes logiciels. Le principe de *modularité* impose la décomposition des problèmes en divers modèles qui participent à définir l'ensemble des préoccupations d'un système particulier. De ce fait, la récomposition des modèles est une tâche sujette à l'introduction d'erreurs, et particulièrement chronophage. La composition de modèles propose des solutions à ces deux problèmes en automatisant tout ou partie du processus de recomposition.

Pour illustrer la problématique que nous adressons dans cette thèse qui est la difficulté de comparer des techniques de composition de modèles et par conséquence la difficulté de les réutiliser ou de les adapter, nous avons analysé les approches de composition de modèles proposées par l'équipe–projet Triskell. Nous voulons ainsi démontrer qu'au sein même d'un groupe de travail partageant la même culture et la même compréhension de l'IDM, les différences de contexte et d'objectif de chaque technique de composition de modèles entrave fortement la capitalisation de l'expérience acquise au travers de ces travaux. Nous voulons également souligner que, bien que ces techniques de composition de modèles aient des objectifs différents, il est possible de détecter des points communs tels que :

- Chaque approche compose une paire de modèles.
- Chaque approche utilise un mécanisme de détection d'éléments de modèles similaires ou équivalents (*i.e.*, appariement)
- Chaque approche propose un mécanisme de composition basé sur les appariements

détectés au préalable.

Au sein même de ces points communs, on observe de la variabilité. Cette variabilité dépend *(i)*du résultat escompté du processus de composition, *(ii)*des caractéristiques propres au modèles, ou bien *(iii)*du degré de généricité proposé par ces techniques de composition de modèles.

Bien qu'inspirées de travaux existants dans la littérature, ces techniques ont été majoritairement conçues et redéveloppées à partir de nouvelles spécifications et non adaptées à partir de modules de composition existants, ce qui démontre une fois de plus la difficulté à identifier les parties communes à un ensemble de techniques de composition de modèles.

Jeanneret *et al.* proposent un cadre pour comparer des techniques de composition [JFB08 ; Jea08] et identifient sur une douzaine d'approches existantes, quels éléments sont composés (*quoi ?*), *où* ces éléments devront être ajoutés ou remplacés et *comment* le processus de composition s'applique. Bézivin *et al.* proposent également de dériver un «...ensemble de définitions de base...» pour la composition de modèles de façon à créer un consensus sur la terminologie employée [BBDF+06].

En étendant le concept de composition de modèles à un ensemble d'opérations plus large, communément appelé «gestion de modèles» dans la terminologie de Bernstein [Ber03], nous pouvons citer les travaux de Brunet *et al.* [BCE+06] qui utilisent un ensemble d'opérations sur les modèles pour comparer les techniques de composition.

On entrevoit aisément, au travers de ces tentatives pour caractériser et classifier les techniques de composition, que la définition de critères de comparaison ainsi que la proposition d'une théorie formalisant la composition de modèles et son outillage est une attente forte dans la communauté des chercheurs en informatique en général, et des chercheurs en ingénierie des modèles en particulier.

Revue Systématique des Techniques de Composition de Modèles

L'utilisation étendue des langages dédiés pour le développement de logiciels nécessite une compréhension accrue des concepts clés de la composition de modèles. La première étape vers cette compréhension nécessité de capitaliser les connaissances acquises sur la composition de modèles. Au vu des points communs observés sur un petit échantillon de techniques de composition et au vu des tentatives de classification décrites précédemment, nous avons observé que les correspondances entre éléments de modèles et l'interprétation de ces correspondances influencent les caractéristiques des techniques de composition de modèles. En conséquence, nous proposons six types de relations de correspondances (*i.e.*, relations permettant d'identifier des appariements) ainsi que quinze types d'interprétations de ces correspondances (*i.e.*, la sémantique d'une relation de correspondance dans un contexte et pour un objectif donné) pour améliorer la comparaison et la classification des différentes approches ainsi que pour détecter des modules réutilisables à moindre effort.

La pertinence et la précision de ces deux catégories ont été validées au travers d'une revue systématique de 88 techniques de composition de modèles existantes dans la littérature. La revue systématique suit le protocole d'analyse adapté à partir

des travaux de Kitchenham [Kit04] et proposé par Biolchini *et al.* [BMA+05].

En bref, les observations faites sur l'état courant de la pratique dans le domaine de la composition de modèles corroborent la classification intuitive des types de correspondances et des types d'interprétations. Avec une précision globale de 65% pour la catégorie de correspondances et un précision de 92% pour la catégorie d'interprétations, nous démontrons que d'une part, **toutes les techniques de composition de modèles peuvent être analysées au travers de leurs correspondances et de leurs interprétations**, et que d'autre part, **les catégories proposées sont suffisamment exhaustives et précises pour classer toutes les techniques de composition de modèles considérées**.

Le protocole d'expérimentations et la totalité des résultats de l'étude sont présentés dans le Chapitre 1.

Travaux connexes

En comparant les approches génériques de composition de modèles (cf. Table 1), nous proposons de traiter dans cette thèse les points suivants :

- Le couplage entre un mapping et une interprétation est généralement très élevé, ce qui implique de définir de nouvelles relations de correspondances dès que l'objectif change et ce qui limite fortement l'adaptation et la réutilisation. Une diminution voire une absence complète de couplage entre un mapping et une interprétation faciliterait d'autant la réutilisation de ces deux concepts dans des contextes et pour des objectifs différents.
- Le paramétrage de l'objectif global d'une technique de composition est généralement manuel. Il serait préférable qu'une partie au moins de ce paramétrage puisse être formalisé de façon à optimiser la construction d'approches génériques et paramétrables pour la composition de modèles.

Les critères proposés pour comparer les approches génériques de composition sont détaillés en Section 1.4.4.

Characteristics / GCF	Predefined Interpretations for mappings	Coupling Mapping / Interpretation	Supports various model composition operations	Composition Process Customization
ATLAS Model Weaver (AMW) [DFB+05b]	manual	medium / high	*	manual (Java Methods)
Object–Relational Mapping (ORM) [GG10]	*	medium	1	manual (Interpreter)
Relation–based Approach [CNM11]	*	high	*	manual
Canonical Scheme [BBDF+06]	manual	high	1	manual
Model Management [BHP00]	*	high	*	manual (Operators)
Kompose [FFR+07 ; FBF+08]	1	high	1	manual (Algorithm)
GeKo and SmartAdapter [MKB+08 ; MPL+09]	1	high	2	manual
ReuseWare [HHJ+08]	1	high	*	manual (Combine two atomic operations)
Generic Aspect–Oriented Modeling Framework [MBJ+07]	manual	medium	1	manual
DUALLy [MMP+10]	manual	medium	1	manual (State Machines)

TABLE 1 – Comparison of existing generic model composition frameworks (GCFs)

Un Cadre Théorique pour la Composition de Modèles

Nous proposons une définition originale de la composition de modèles qui s'inspire de la notion de *structure* en logique mathématique et de la notion de *signe* en linguistique. La définition d'un cadre spécifique pour la composition de modèles est une paire constituée d'un mapping et d'un ensemble d'interprétations de ces mappings.

La définition d'une structure en logique mathématique nous permet de relier les concepts de mapping et interprétation à une théorie fondée qui sépare concrètement différents concepts et qui définit explicitement les relations entre ces concepts. La nature exacte de ces relations entre mapping et interprétation n'est cependant que partiellement capturée dans l'état courant de la pratique. Nous proposons alors d'utiliser des notions empruntées à la linguistique pour détailler en quoi l'objectif d'une composition de modèles et en quoi la part d'interaction avec l'humain influencent à la fois ces relations entre mapping et interprétation, et l'objectif global d'une approche

générique pour la composition de modèles.

A partir de ces deux parallèles, nous proposons une définition de la composition de modèles comme étant une paire constituée d'un mapping et d'un ensemble d'interprétations pour ces mapppings tel que $MC = \langle MM, I \rangle$, où MC est la définition d'un cadre spécifique pour la composition de modèles, MM est un mapping et I est un ensemble d'interprétations.

Pour répondre à la problématique du couplage entre mapping et interprétation, nous proposons une théorie unifiée pour la composition de modèles qui définit de façon formelle les différents types de mappings et les différents types d'interprétations (cf. Chapitre 2).

La construction d'un opérateur de composition de modèles spécifique à un contexte de composition particulier est une tâche difficile et chronophage. La contribution de ce chapitre permet d'obtenir une définition formelle des différents types de mappings et des différents types d'interprétations. La formalisation aide à comprendre la sémantique de chaque mapping et de chaque interprétation et facilite ainsi la définition de nouvelles opérations de composition.

Ce chapitre nous permet également de mettre en exergue les deux étapes de la définition d'un nouvel opérateur de composition :

- Les utilisateurs sélectionnent une paire d'un mapping et d'un ensemble d'interprétations qui participent à la réalisation de l'objectif de l'opérateur de composition. Cette paire devient de fait la spécification de l'opérateur.
- Le paramétrage de l'opérateur ainsi défini permet de prendre en compte le contexte dans lequel l'opérateur de composition sera utilisé. Le contexte fait référence à la fois aux types de modèles manipulés par l'opération mais aussi aux spécificités de l'opération de composition qui ne peuvent être capturée par les interprétations.

La Figure 2 présente ce processus en deux étapes dans le cas particulier du développement d'un opérateur de fusion («merge ») de modèles. La sélection d'un mapping et d'une interprétation de ce mapping définit une opération de composition dont l'objectif général est la fusion de modèles. L'utilisateur doit paramétrer cet objectif en proposant plusieurs algorithmes. Ces algorithmes permettent l'exécution de l'opérateur de fusion sur différents formalismes. Sur la Figure 2 par exemple, le paramétrage permet à partir d'une unique paire d'un mapping et d'un ensemble d'interprétations de proposer une opération de fusion de modèles UML ou ECore, et ce de manière homogène (les modèles sont conformes au même méta-modèle) ou hétérogène (les modèles sont conformes à des méta-modèles différents). Bien entendu, le paramétrage de l'opération de fusion est réalisée en fonction des exigences de l'utilisateur et de ces besoins.

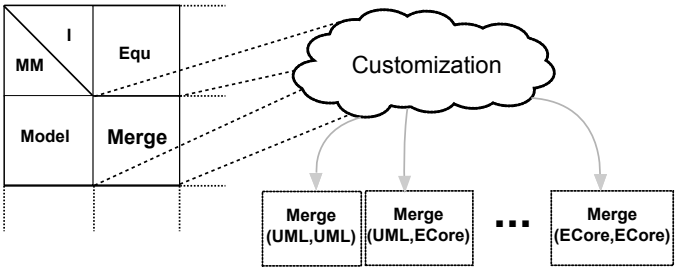


FIGURE 2 – Une paire d'un mapping et d'une interprétation est paramétrée pour définir plusieurs opérations spécifiques de fusion de modèles.

ModMap : Un Langage de Modélisation pour la Composition de Modèles

Une approche générique pour la composition de modèles outille un objectif particulier de composition, représentant concrètement un type d'opérations particulières entre modèles. Le Chapitre 3 de cette thèse présente l'outil ModMap (MODel MAPping). Cette outil permet de définir des mappings et leur associer des interprétations pour construire des langages de composition de modèles et des opérateurs de composition concrets.

L'implémentation de ModMap propose *(i)*un langage de définition de mappings entre modèles et/ou méta–modèles ; *(ii)*une sémantique opérationnelle pour chaque type d'interprétations ; *(iii)*une syntaxe concrète graphique qui simplifie la spécification de mappings et la sélection des interprétations nécessaires ; *(iv)*une méthodologie pour construire de nouvelles approches générique de composition de modèles.

La méthodologie pour la construction de nouvelles approches de composition de modèles s'appuie sur l'architecture de ModMap décrite en Figure 3.

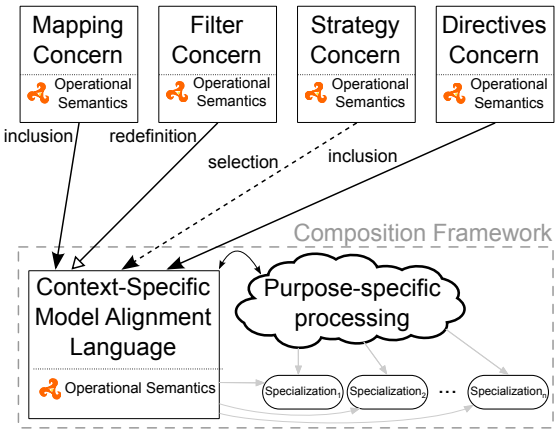


FIGURE 3 – Processus de définition d'approche de composition de modèles spécifiques.

Le langage de définition de mappings entre modèles et méta-modèles (*i.e.*, langage

d'alignement) est décomposé en quatre préoccupations telles que :

- *Mapping* est la représentation de mappings entre éléments de modèles (voir Section 3.2.2.1).
- *Filter* offre aux utilisateurs la possibilité de proposer des fonctionnalités de filtrage sur les mappings (voir Section 3.2.2.1).
- *Strategy* représente les différents types d'interprétations (voir Section 3.2.2.2) formalisés dans le Chapitre 2 et permet d'associer des interprétations spécifiques à un mapping.
- *Directives* représente un ensemble d'opérations atomiques applicables sur les éléments de modèles (voir Section 3.2.2.2). Ces opérations servent à paramétrer les interprétations, offrant un moyen aux utilisateurs de paramétrer l'opération de composition de modèles pour leurs propres besoins.

Mise en œuvre, la méthodologie amène à produire des langages d'alignements spécifiques pour un objectif de composition donné. La prise en compte du contexte de composition est représentée par un processus spécifique (*i.e.*, un ensemble d'algorithmes) fourni par les utilisateurs. Chaque algorithme devrait permettre alors de traiter un cas particulier de l'objectif global de la composition de modèles, lui-même définit par le langage d'alignement.

L'outil ModMap répond à la problématique du paramétrage de l'objectif d'une approche de composition de modèles. La construction et le paramétrage d'une approche générique pour la composition de modèles sont basées sur des modules réutilisables et combinables entre eux pour traiter un objectif particulier.

Mise en oeuvre et Validation

La validation de notre approche et de l'outil ModMap s'appuie sur deux expérimentations principales : la première consiste à évaluer la capacité de ModMap à redéfinir des techniques existantes pour la composition de modèles homogènes ; la seconde consiste à évaluer l'utilisation de ModMap pour des problématiques de composition différentes telles que l'intégration ou la synchronisation de modèles hétérogènes.

L'intégration de modèles hétérogènes fait partie du cas d'étude proposé par Technicolor pour l'intégration de librairies existantes dédiées à la gestion d'équipements numériques de diffusion vidéo. Au travers de ce cas d'étude industriel, nous avons mis en application les principes de ModMap. L'outil dédié à l'intégration de modèles fait actuellement l'objet de plusieurs expérimentations sur différentes librairies. L'outil proposé pourrait également répondre à d'autres problématiques au sein du projet MOPCOM–I. Des études préliminaires sont en cours avec nos partenaires de France Télécom ainsi que ceux de Thalès Systèmes Aéroportés.

Les cas d'études ainsi que la validation sont détaillés dans le Chapitre 4.

Perspectives

Cette thèse propose un cadre de modélisation outillé qui permet la définition de mappings et d'interprétations dans le but de construire des opérateurs de composition spécifiques. Ce cadre de modélisation est une boîte à outils que les experts peuvent utiliser et paramétrer pour répondre à des problèmes particuliers de composition de modèles.

Nous pensons que la contribution principale de cette thèse est un pas important vers la définition d' opérations de composition paramétrables, s'appuyant sur des opérateurs existants, et traitant un large spectre d'usages. Dans ce contexte, la contribution de cette thèse ouvre de nouvelles perspectives de recherche.

Extension de la Revue Systématique de Littérature

Dans le cadre de cette thèse, nous avons mis un effort particulier sur la définition des concepts principaux de la composition de modèles, qui sont les correspondances et les interprétations. La variabilité de ces deux concepts au sein des techniques de composition de modèles existantes est présentée dans le Chapitre 1. Les catégories de correspondances et d'interprétations vont évoluer à mesure que le cadre théorique sera mis en pratique. En plus de cette évolution liée à l'utilisation du cadre théorique, nous envisageons trois pistes de recherche supplémentaires.

Influence des Activités de Développement Logiciel

Nous sommes convaincu que les activités de développement logiciel influencent la définition et la spécification des correspondances et des interprétations. Nous jugeons utile de poursuivre l'analyse des données issues de la revue systématique pour extraire des informations sur l'influence de ces activités de développement. Nous pensons que cette information permettrait de *(i)*lister les paires de correspondances et d'interprétations pertinentes pour une activité de développement donnée, et ultimement de *(ii)*fournir une liste des techniques de composition de modèles qui supportent une activité de développement particulière. Nous fournissons quelques pistes à ce sujet dans la revue systématique mais de plus amples analyses sont nécessaires.

Adaptation des Techniques de Composition Existantes

Dans les résultats de la revue systématique de littérature, on observe que certaines techniques de composition proposent déjà différentes opérations sur un ensemble de modèles. Dans des cas particuliers, la réutilisation ou l'adaptation de ces techniques semblent être des pistes intéressantes pour construire de nouvelles opérations de composition de modèles.

Complétude de la Classification

La présentation des résultats de la revue systématique de littérature sous la forme de tables permet de détecter des paires d'un mapping avec une interprétation ainsi que des paires d'un mapping avec une activité de développement pour lesquelles aucune approche n'a été identifée. Dans l'optique de fournir une classification complète des techniques de composition, il serait intéressant d'identifier les raisons pour lesquelles de telles paires n'ont pas encore été proposées dans la littérature et quels sont les défis scientifiques sous-jacents.

Composition de Modèles : Une Entité de Premier Ordre pour l'IDM

La composition de modèles dans des environnements d'ingénierie logicielle multi-vues est une activité clé. Bézivin *et al.* sont persuadés que les techniques de composition de modèles devraient être promues au statut d'éléments de premier ordre, tout comme il a été fait avec les techniques de transformation de modèles [BBDF+06]. En proposant une sémantique opérationnelle pour Unified Modeling Language (UML), Siobhá Clarke propose que «les modèles de conception orientés acteurs puisse supporter un nouveau type de concept appelé une relation de composition qui devrait pouvoir spécifier comment les modèles doivent être composés.» [Cla02, §3, p.6]. Conformément à la définition de composition de modèles proposée dans cette thèse et en accord avec ces deux propositions, nous proposons deux perspectives de recherche sur ce point précis.

Une Relation «Composable»dans le Meta-Object Facility (MOF)

Les observations proposées dans cette thèse confirment le fait que la composition de modèles est une technique pertinente pour adresser une large panel d'activités sur les modèles. La composition de modèles serait alors, au même titre que la transformation, un concept clé. La prochaine étape vers une définition de la composition de modèles serait de proposer une représentation à un niveau d'abstraction supérieur de faç à offrir aux utilisateurs un outil intégré dans leurs modélisations. Cette proposition revient à créer un nouveau concept dans le meta-meta-modèle Meta-Object Facility (MOF). En d'autres termes, le concept *Property* du MOF pourrait intégrer un nouveau type d'association tel que proposé en Figure 4. Nous pourrions alors lui donner la sémantique suivante qui s'inspire des propositions de sémantiques pour les associations du MOF [OMG10a, §12.5, p.45] :

Property : :isComposition==true

- Un objet peut être *composé* avec plusieurs objets
- La composition cyclique est valide : l'ordre de composition est déterminé par l'implémentation concrète des instances d'associations.
- Chaque association de type composition devrait recevoir une sémantique particulière en utilisant un langage d'action tel que Kermeta [MFJ05 ; JBF10].

La sémantique de cette nouvelle association de composition devrait inclure le Domain-Specific Modeling Language (DSML) de mappings et d'interprétations présenté

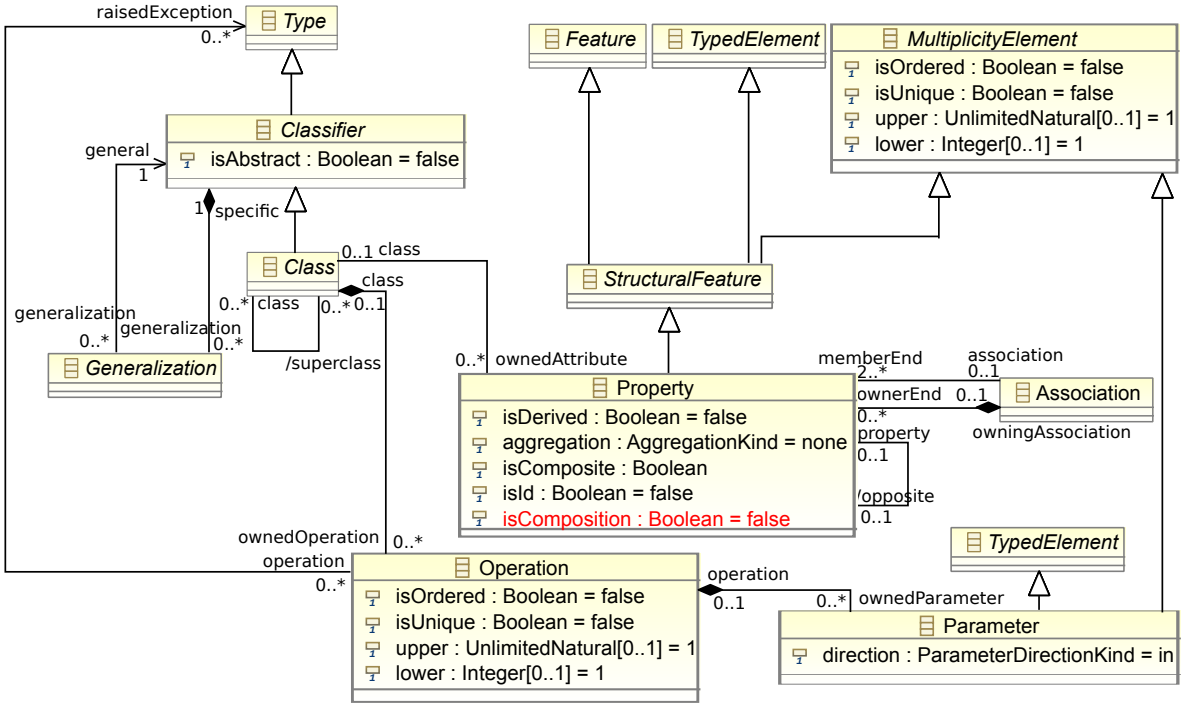


FIGURE 4 – Méta-modèle MOF : Classes de EssentialMOF et adaptées à partir de [OMG10a, §12.2, p.40]

dans le Chapitre

The semantics of the new *composition* relationship would include the mapping and interpretation DSML semantics that are presented in 3. Une telle association devrait alors améliorer la spécification d'opérations de composition sur les modèles et devrait intégrer cette spécification au sein même de l'activité de conception des systèmes et des langages.

Composition d'Ordre Supérieur

Une des conséquences directes de la promotion de la composition de modèles à un niveau d'abstraction supérieur est le fait de pouvoir manipuler cette composition de la même façon que tous les autres concepts. L'application de la composition de modèles à un ordre supérieur (HOC) permettrait de composer des compositions de modèles existantes and le but de faciliter la conception de certains systèmes. Par exemple, une HOC pourrait définir la composition de deux langages $DSML_1$ et $DSML_3$ à partir de deux compositions existantes telles que $DSML_1$ avec $DSML_2$ et $DSML_2$ avec $DSML_3$. Proposer de manipuler des HOC devrait permettre d'ouvrir de nouveaux espaces de recherche et d'applications pour la génération d'opérations de composition à partir de techniques existantes.

ModMap

Extension du Champ d'Application de ModMap

Les résultats préliminaires obtenus dans la validation permettent de généraliser la définition de la fusion de modèles et de redéfinir quatre techniques existantes avec un module d'appariement et un module de fusion génériques. Les perspectives sur ce travail sont doubles : *(i)*évaluer le passage à l'échelle d'une telle généralisation et découvrir les limites d'application du cadre théorique ; mettre en place un dépôt de composants de composition «sur étagère»reutilisables et extensibles pour différentes opérations sur les modèles.

L'utilisation de ModMap pour l'intégration et la synchronisation de modèles appelle à poursuivre les expérimentations pour d'autres activités liées au développement logiciel telles que la dérivation de modèles, l'orchestration de modèles, la vérification de consistence ou encore la reconfiguration dynamique de systèmes à base de modèles.

Collaborations

Dans le cadre du projet MOPCOM-I and au vu des résultats encourageant de ModMap sur le cas d'étude de Technicolor, nous envisageons de futures collaborations avec différents partenaires du projet tels que France Télécom et Thales Systèmes Aéroportés. Des réunions de travail dédiées sont actuellement menées avec nos partenaires de France Télécom sur la définition de mappings entre des WebServices et les fonctionnalités d'un système existante pour automatiser la conception des interfaces de WebServices. Thales Systèmes Aéroportés propose trois cas d'étude : *(i)*la définition de mappings modèle-à-modèle pour la spécification de transformations entre DSML ; *(ii)*la définition de mappings entre documents semi-structurés et un DSML pour assurer la persistence de données ; *(iii)*la définition de mappings entre un DSML et une représentation abstraite de l'environnement pour la conception d'interfaces logicielles. En supplément des nouvelles expériementations proposées, nous envisageons de continuer le développement du prototype de façon à permettre son utilisation dans un contexte plus industriel et nous envisageons de mettre en place des métriques pour évaluer l'adéquation de ModMap avec des contextes d'utilisation particuliers.

Table of contents

Introduction

Information Technology and Complexity

Information Technology has become predominant and pervasive in industry, economics, finance, communication or transportation, to cite only a few ones. These activities all rely on systems which help to design, to manage, to model, to improve, to enhance or to support those activities. Pervasiveness of information technology leads to the definition of more and more complex systems to handle a wider and wider range of situations and standard techniques for systems development and engineering are dramatically impacted. Combination of programs, modules or functions become a matter of multiple actors within controlled life–cycles, possibly geographically distributed.

Dealing with Complexity

Managing and pruning complexity are recurrent problems in any scientific reasoning. The activities of decomposing problems into more manageable subproblems and propose abstract representations to hide unnecessary details are the keys to properly understand a given situation and to successfully provide solutions. The degree to which these subproblems may be separated and recombined is the principle of modularity.

Model-Driven Engineering (MDE) is based on the principle of abstraction which consists in proposing partial and abstract representations to solve a given subproblem. The MDE also proposes to shift practices from "models for documentation" to "models for production" thus promoting models as first-class entities and productive artifacts to make software engineering more efficient, cost–effective and safer.

Model Composition in MDE

While the application of modularity and abstraction principles helps tackling complexity, designers are required to manipulate an increasing number of partial representations. The activity of decomposing problems needs a step of recomposition at a specific time to get a global representation of a system under construction and to reason about the system as a whole for verification, validation and consistency checking purposes. The recomposition step is however an error–prone and time–consuming activity. Within the framework of MDE, model composition is an active field of research that

focuses on automating the composition of model–based artifacts in a multi–modeling environment.

Genericity in Model Composition Techniques

Model composition is a challenging topic of interest in which the definition of new approaches should benefit from existing model composition techniques. In the perspective of proposing adaptable model composition approaches, genericity plays a significant role. Generic model composition frameworks (GCFs) deal with various kinds of models in various contexts. However GCFs are initially designed to tackle a specific goal. When new goals emerge, building new GCFs may benefit from existing GCFs and things go round and round.

Identifying Model Composition Key Concepts

Reuse and adaptation imply to identify commonalties among GCFs and shift from composition as an operator (a single definition for a single use – whatever large the spectrum of composable models could be) to composition as an operation (a customizable definition and a choice of existing operators to achieve a possibly large range of goals). However, the lack of a common formalism for analyzing and comparing the multitude of model composition approaches hinders the identification of reusable artifacts.

The main contribution of this thesis is to propose a **`novel definition of model composition as a pair of a mapping and an interpretation`**. A mapping is a set of alignment rules between patterns of model element to detect how similar a set of models are. An interpretation is the representation of both the purpose of the model composition process and the user's expectations on the model composition process by-products. This definition paves the way to a theoretical framework that helps *(i)*unifying the definition of existing model composition techniques and *(ii)*automating the process of building problem–specific model composition tools.

The main contribution is supported by two subsidiaries propositions :

- We propose **`categories to classify existing model mapping techniques and existing model mapping interpretations`**. This leads to the definition of an interpretive lens for analyzing and comparing existing model composition approaches.
- We define a **modeling language that supports the definition of generic mappings among models and the definition of interpretations**. The language is inspired from the proposed categories.

Validation

We implemented a prototype that supports the novel definition of model composition. We validate the main contribution of this thesis through three experiments : *(i)*we use the theoretical framework to unify four existing model merging techniques and

propose a unique kernel for model composition ; *(ii)*we demonstrate the applicability of the framework on the integration of legacy Application Programming Interface (API) for the configuration and management of heterogeneous video and broadcasting equipments in collaboration with industrial partners from Technicolor [1] ; *(iii)*we demonstrate the applicability of the framework on the synchronization of heterogeneous models in the context of modeling service–oriented architectures (SOA).

Outline

Chapter 1 presents the thesis background, the classification of the key concepts of model composition, and the systematic literature review protocol and results. Chapter 2 proposes a formal definition of the theoretical framework for unifying model composition approaches. Chapter 3 details the ModMap modeling language for the definition of mappings and interpretations and how it helps building new model composition frameworks. Chapter 4 lists experiments that validate the relevance of the ModMap modeling language with regard to the definition of model composition approaches. We conclude this thesis by providing perspectives and insights of this work regarding future research on model composition techniques.

1. http ://www.technicolor.com/en/hi/technology/research-and-innovation-centers/rennes

Chapitre 1

Model Composition in Software Engineering

This chapter presents the context in which this thesis has been conducted and draws up the state of the art in literature about the importance of this thesis topic in computer science in general and software engineering in particular.

The purpose of this chapter is threefold : *(i)*Explore the current state of practice in software development and give an overview of how modular design, MDE and model composition bring pragmatic solutions to deal with the growing complexity of systems ; *(ii)*Propose an intuitive definition of the key concepts in model composition to provide an interpretive lens for comparing existing model composition approaches ; *(iii)*Conduct a systematic literature review on model composition techniques to validate the proposed key concepts and ultimately propose a unified theoretical framework on model composition for enhancing model composition tools reuse and adaptation.

Figure 1.1 illustrates the outline of this chapter : Section 1.1 presents how model composition helps tackle complexity in systems design and development. Section 1.2 proposes categories of the key concepts in model composition to classify existing model composition techniques. Section 1.3 presents the protocol and results of the systematic literature review that we conducted.

1.1 Modularity in Software Engineering

Facing a given problem, the scientific reasoning implies the decomposition of a problem into simpler and more manageable subproblems. The activity of decomposing problems thus put scientists in situations that they may already know or for which they are most likely to successfully find a solution. The degree to which these subproblems may be separated and recombined is the very principle of modularity.

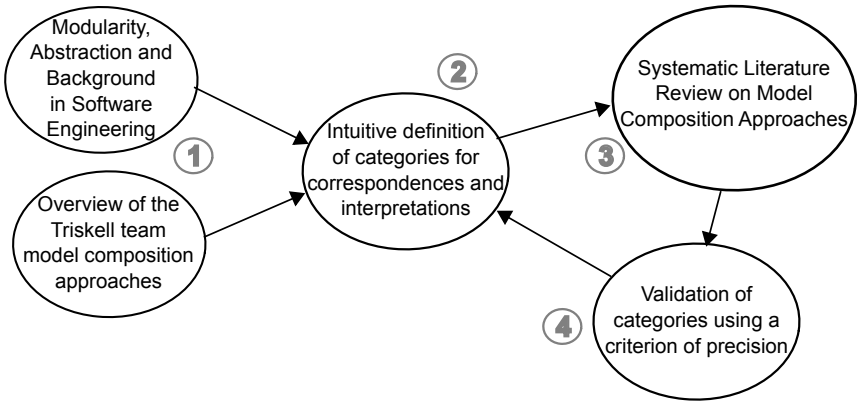


FIGURE 1.1 – Illustration of the process followed in Chapter 1

1.1.1 Modularity

In Computer Science, modularity allows the reuse and the composition of small chunks of programs to achieve a bigger purpose.

In the fifties, automatic programming was a very active field of research. Automatic programming refers to combining various routines for a single machine to perform the final computation. For instance, Curry proposed a linear notation to express the combination of program routines in [Cur54].

In the seventies, Dijkstra [Dij97] and Parnas [Par72] introduced the Separation of Concerns (SoC) principle that invites designers to break down systems into units of behavior or units of function to improve program composition and to control the ever-growing complexity of systems.

In the nineties, Chandy and Taylor [CT90], Vargas–Vera [VV95] and Jackson [Jac90] approaches addressed the challenge of software complexity in proposing to automate the combination of units of decomposition in imperative programs. These approaches provide answers to Jackson's statement about "[h]aving divided to conquer, we must reunite to rule" [Jac90].

Within the Object–Oriented (OO) paradigm, the application of the SoC principle helps designers to decompose problems into objects that represent functional unit of the overall system. OO programming is a technology that can fundamentally aid software engineering because the underlying object model provides a better fit with real domain problems. However according to Kiczales *et al.*, in many situations, the OO paradigm is not sufficient to clearly capture all the important design decisions that a program must implement.

Thus, Kiczales *et al.* proposes to separate the main concern of a system from the non functional and cross-cutting concerns [KLM+97]. He proposes to capture non functional and cross–cutting code of a given system into units of decomposition called *aspects*. In the event of building a consistent and global view of a system, aspects are composed with one another within an operation called *weaving*. An aspect is decomposed into two parts : *(i)*the *advice* contains the code of the concern that is to

be woven in specific places (*i.e.*, *join points* in the control flow of the main concern ; *(ii)*a *pointcut* descriptor that allows identifying join points within the main concern. Most tools that support the Aspect–Oriented Programming (AOP) paradigm extend existing programming languages such as C or Java. For instance, the AspectJ extends Java both with a new class named "aspect" that contains pointcuts and advices, and with a specific syntax for the definition of both pointcuts and advices.

Considering the evolution of practices in modular design and the difficulty to tackle the ever growing complexity of systems, the MDE approach emerged as a new approach for software engineering.

1.1.2 Abstraction and Model–Driven Engineering

The MDE approach is based on the principle that "everything is a model". Going further from the use of *design patterns* [GHJ+95] and *aspects* [KLM+97], MDE targets the manipulation of a large number of models to capture the various concerns of a system. The definition of a model, proposed by the Object Management Group (OMG) is as follows :

> [A] model is a representation of a part of the function, structure and/or behavior of a system. [A model is a] specification [that] is said to be formal when it is based on a language that has a well–defined form ("syntax"), meaning ("semantics"), and possibly rules of analysis, inference, or proof for its constructs. The syntax may be graphical or textual. The semantics might be defined, more or less formally, in terms of things observed in the world being described (*e.g.*, message sends and replies, object states and state changes, etc.), or by translating higher-level language constructs into other constructs that have a well–defined meaning. [OMG01, §2.2.1, p.3]

A model encapsulates a partial and/or simplified description of a real "object" from a specific point-of-view and for a given purpose. Software specification and documentation are examples of abstractions that allow various people to participate, interact and exchange specific information about a given system under development.

The OMG proposed to go one step further and to move from "design for documentation" to "design for production", promoting models (*i.e.*, abstractions) as first-class entities. In other words, models should both document software and be machine readable to make software development safer, cheaper and more efficient.

Building models require a modeling language that defines the concepts (*i.e.*, model elements), the structure and the semantics of a model. Concepts, structure and semantics are captured into a representation called a meta–model : a model at a higher level of abstraction that defines the modeling language of a specific model. The conformance relationship between a model and its meta–model is such that : a model only contains concepts from the meta–model and satisfies the meta–model constraints.

Since a meta-model is yet another model, it conforms to its own meta-model that we call meta-meta-model. Meta-meta-models are usually reflexive (*i.e.*, they define their own elements, structure and semantics) to avoid the multiplication of the levels of abstraction. In the terminology of MDE, this thesis focus on the technical space

of ModelWare as shown in the pyramidal representation of meta-models (see Figure 1.2) [FEB06].

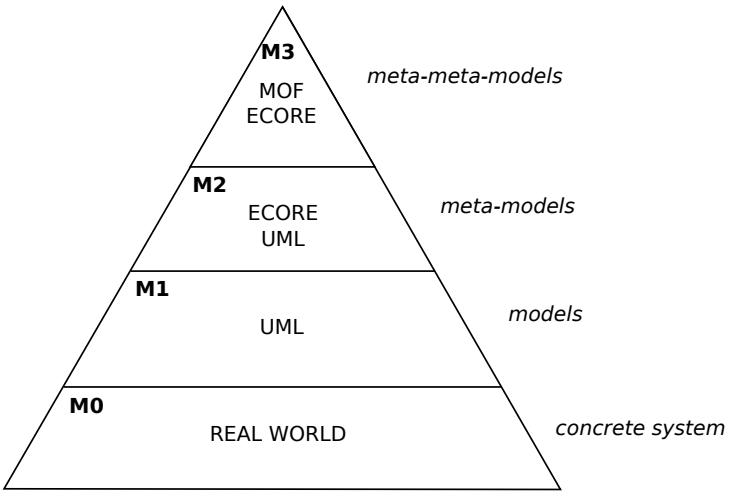


FIGURE 1.2 – A pyramidal representation of the levels of abstraction in Model–Driven Engineering.

In the context of this thesis, we focus on the Modelware technical space which own general-purpose modeling languages (GPML) such as MOF, UML or EMFCore (ECore) and domain-specific modeling languages (DSML) such as SysML.

1.1.3 Modularity, Abstraction and Model Composition

Building models following the approach of MDE leads to the development of dedicated languages and models to ease the specification of systems. Dealing with complexity requires to build modular model. Modular design subsequently imposes to create models for each concern that the system should tackle. Thus, designers are required to manipulate an increasing number of models. This situation makes hand-made composition an error-prone and time-consuming activity. Model composition approaches helps in automating the process of recombining models with one another to get dedicated and consistent views of the system under design/development.

1.1.4 A brief Overview of Model Composition Techniques

Most model composition techniques available are built for a specific purpose in a given context. As an illustration, Sections 1.1.4.1 to 1.1.4.4 briefly describe the model composition approaches proposed by the Triskell [1] team and its partners. This demonstrates that even among persons who share the same culture and knowledge about MDE and model composition, different contexts and purposes prevent embodying

1. http ://www.irisa.fr/triskell

the experience of individuals in the definition of a common approach for composing models.

1.1.4.1 Kompose : A Generic Model Composition Tool

Kompose [FBF+08] is a model composition technique that supports merging homogeneous models (*i.e.*, models that conform to the same meta–model). The merging process is decomposed into four steps as follows :

1. **Adapting** the models that are going to be merged with directives, generalized from [RGF+06] to prepare or force the merge of two models elements.
2. **Matching** model elements with one another using signatures. Signatures are the specification of the matching mechanism. Signatures state on which data two model elements should be compared with one another.
3. **Merging** the model elements uses reflexivity and introspection. Model elements that match are merged into a single model element while the elements with no counterpart are added untouched in the output model.
4. **Finalizing** the merging process using directives to tune the output model to get the expected result.

The Kompose tool has an extension mechanism called "specialization". A specialization is the adaptation of the model merging tool to support various MOF–based modeling languages. The Kompose approach allows automating most of the merging process whilst providing a certain degree of customization and user interaction if the situation requires it.

1.1.4.2 SmartAdapters : A Model Weaver that supports Variability

The SmartAdapters [MPL+09 ; LMV+07] approach has been developed in collaboration with the Triskell team, the CoCoa team [2] from the LIFL French laboratory and the MODALIS team [3] from the I3S French laboratory. SmartAdapters has been originally designed to provide capabilities for functional or extra–functional concerns to be reused in the context of variability.SmartAdapters is a homogeneous and asymmetric approach to weave reusable concerns (*i.e.*, aspects) into one or several base models. Each aspect is related to an adapter that declares a composition protocol. A composition protocol is a set of atomic operations and a set of target model elements that specifies how the aspect should be woven with other aspects. The adapter specification is the basis for identifying reusable aspects. Inspired by Software Product Line (SPL), the composition protocol supports optional parts, variants definition and dependency constraints to ensure consistency. The SmartAdapters process is composed of five steps as follows :

1. **Generating** an extensible Aspect-Oriented Modeling (AOM) framework specific to a meta–model.

2. http ://www2.lifl.fr/GOAL/cocoa/
3. http ://modalis.polytech.unice.fr/

2. **Defining** aspects and associated weaving directives.
3. **Matching** all the places that match the aspect model. This step is supported by a pattern–matching engine that relies on Drools [4].
4. **Processing** the Drools rules in memory.
5. **Weaving** the base model with the aspects models at runtime.

1.1.4.3 GeKo : A Generic Aspect Oriented Composer

With GeKo (Generic Komposer) [MKB+08], Morin *et al.* explore the use of mappings between different views of an aspect in the context of software product lines. GeKo relies on the definition of a pointcut model, possibly automatically generated from an automatic Prolog-based pattern–matching mechanism, and on the definition of two morphisms that identify which operations can be performed on the base model and on the aspect models. Morphisms partition the base and aspect models into sets that contain model elements to keep unchanged, model elements to remove, model elements to be replaced and model elements to be added in the based model. The GeKo process is as follows :

1. **Weakening the metamodel** of the based model allows the definition of abstract pointcuts which can match a larger number of model elements in the base model. Weakening a metamodel consists in removing constraints, declare all feature as optional and making all abstract model elements concrete.
2. **Identifying join points** is achieved by a Prolog–based pattern–matching mechanism that tries to match pointcut with model elements from the base model. A Prolog query is executed over a knowledge base containing the domain metamodel, the base model, and the abstract pointcuts and results are converted back into a Kermeta [MFJ05 ; JBF10] data-structure.
3. **Applying morphisms** to match pointcuts with the advice model to compute the sets of model elements to add, to replace, or to remove.
4. **Finalizing** the weaving process by constructing a result from the union of the models elements to keep, to add and to replace, and replacing or removing model elements that should not be part of the result. Then a "cleaning" step is performed to delete the relationships to the model elements that have been removed in the the first phase of the finalization.

1.1.4.4 Semantic-based Weaving of Scenarios

In [KHJ06], Klein *et al.* proposed an algorithm to weave aspects into Message Sequence Charts (MSC), taking into account the semantics of the MSCs. Aspects are represented as alternative scenarios that address specific behavior in case the nominal scenario cannot be executed. Providing a matching algorithm that allows the detection of multiple minimal matchings in MSCs, they extend the matching algorithm to detect

4. http ://www.jboss.org/drools/drools-expert.html

matchings into a possibly infinite set of behaviors extracted from hierarchical MSCs (hMSC). The global process consists of phases as follows :

1. **Detect** potential matches from unfolding hierarchical MSC and produce an acyclic hMSC.
2. **Detect** future matches from unfolding hMSCs, starting from end nodes.
3. **Replace** all join points detected from the previous steps with the advice.

This approach suffers from several limitations inherent to the MSC but provide a procedure to weave aspects into MSC at some degree. This approach does not seem generic at a first glance, since the definition of pointcut and advice, and the proposed algorithms are specific to the MSC structure.

1.1.4.5 Discussion

In the light of these four model composition techniques, we intuitively identify three similarities as follows :

- Every technique composes a pair of models.
- Every technique proposes a mechanism for detecting similar or equivalent model elements (*i.e.*, matching).
- Every technique proposes a mechanism that uses matchings for combining models.

Among these very similarities, we observe variability that may depend on *(i)*the result that designers expect, *(ii)*the characteristics of the models, or *(iii)*the degree of genericity that the techniques propose. In Section 1.1.5, we discuss existing approaches and frameworks for comparing model composition techniques.

1.1.5 Comparing Model Composition Techniques

We presented in Section 1.1.4 several model composition techniques that allow composing models with one another in a specific context or for a specific purpose. Inspired by previous works, these techniques were nonetheless mainly developed from scratch. Reusing and adapting existing model composition techniques to address new contexts or purposes is still a challenge and remains a difficult and error-prone activity.

In [JFB08] and [Jea08], Jeanneret *et al.* propose a framework to compare model composition techniques and listed a dozen dedicated model composition techniques. The basics of Jeanneret's comparison framework is the triplet **`what-where-how`** questions identifying respectively which elements should be composed, where elements should be inserted or modified and how the composition process works to get the expected result. This work also proposes naming conventions to distinguish between model composition, model weaving, model merging, and model alignment that are often mixed up in the literature. As Jörg Striegnitz stated, coming to an agreement on model composition terminology needs a "...theory of extensible languages to reason about language ... compositions."[CØV02, 2002, p.13]. Until now, we are not aware of any such unifying theory or any such consensus.

Extending the definition of model composition to a wider ranger of operations on models is usually referred as **model management** in the terminology of Bernstein [Ber03]. In [BCE+06], Brunet *et al.* proposed a framework for comparing existing merging approaches. They propose a set of model management operations and they identify relevant properties for a model merging operator. Similarly, Boucké *et al.* proposed to characterize relations between various views of a system to extract a comparison framework [BWH+08]. The comparison framework stresses the following three dimensions : **Usage** which refers to the purpose of the relationships between views, **Scope** which encompasses the range of the relationships, and **Mechanism** which details the nature of the elements related with one another. In [Ber03], Bernstein proposed several model management operators but still he asks for "[m]ore detailed semantics of model management operators"[Ber03, 2003, p.11] and a definition of "the boundary of useful model management computations"[Ber03, 2003, p.12]. In [ZDD06], Zito *et al.* also propose to see the issues of "package merge and extension...in a even wider context as particular problems in generic model management,...towards a general theory of model manipulation and transformation."[ZDD06, 2006, p.5].

Steel and Jézéquel concludes their proposition on model typing [SJ07] stating that "[t]he lack of proper mechanisms for typing operations on models such as model transformations leads to brittle and overly restrictive reuse characteristics" [SJ07, 2007, p.11]. We think that a common background for comparing and analyzing existing model composition techniques is necessary to define and/or identify such reuse characteristics.

In the light of these attempts to classify or at least organize model composition techniques, the computer science community and especially the model–driven engineering community is eager to have such a comparison tool or theory on model composition to "...help giving first class status to model composition..."[BBDF+06, 2006, p.14] and to leverage research on model composition.

1.2 Key concepts in Model Composition

Among the multitude of model composition techniques and tools available, we consider that organizing the body of knowledge encompassing existing model composition tools and techniques is the first step towards understanding the key concepts in model composition and leveraging the applicability of Domain-Specific Language (DSL) in software developers and designers day-to-day activities.

In the light of Section 1.1.4.5 and previous attempts to classify model composition approaches (see Section 1.1.5), we observe that correspondences and interpretations of these correspondences have a strong influence in the characterization of model composition approaches. We expect that proposing categories of correspondences and categories of interpretations will ease comparing, classifying and detecting reusable modules in existing model composition techniques.

Section 1.2.1 propose the definition of six categories of **correspondence relationships** that are the basis of the identification of the model elements involved in any

model composition process. Since model composition techniques work towards a specific purpose, we propose fifteen kinds of **`interpretation`** for these correspondences divided in three categories (see Section 1.2.2). Interpretation is what we define as the meaning of the correspondence relationships for a given purpose in a specific context. For the purpose of validation, we conducted a systematic literature review (see Section 1.3 to evaluate how precise and relevant each category is, regarding the state of practice in model composition.

1.2.1 Correspondences

In the definition of an operation that manipulates models to achieve a specific goal, we consider correspondences as any kind of implicit or explicit relationships between sets of models or sets of model elements. This section presents an intuitive definition of the six kinds of correspondences relationships (see Figure 1.3) that we propose. The proposed classification is based on an extensive work on identifying correspondence relationships from various model manipulation techniques. Sections 1.2.1.1 to 1.2.1.6 present an intuitive definition of the six categories of relationships.

1.2.1.1 Operator-based correspondence

Operator-based correspondence relationships rely on a set of functions or similar constructs which operational semantics encompasses both the specification of the correspondence between model elements and the interpretation of the correspondence. This category includes techniques which provide predefined operators with fixed semantics for a given purpose (*e.g.*, [LNK+01 ; ASM+10 ; PGP+07 ; Ber03 ; Bar08]).

1.2.1.2 Pattern-based correspondence

Pattern–based correspondences are patterns or constructs very similar to patterns to both find and express similarities between model elements. This category ranges from very straight-forward pattern definitions for term-matching on names or other linguistic terms to complex join-points, pointcuts, signatures or morphisms that provide more control, more expressiveness and more flexibility in the definition of correspondences (*e.g.*, [Cla02 ; Jez08 ; BSM+07 ; ACL+10 ; MKB+08 ; TT08]).

1.2.1.3 Rule-based correspondence

Rule–based correspondences are expressed as specific rules that support identification, selection and filtering of model elements that should be in relation with one another (*e.g.*, [CRR+07 ; JWE+07 ; HKG+10 ; SY10 ; PBB+09 ; BTF05]). Similar to patterns that consist in looking for templates adequateness to identify correspondences, rules allow nevertheless more expressiveness and more complex computations.

1.2.1.4 Constraint-based correspondence

Satisfying constraints such as pre- or post-conditions, invariants, context-free or context-dependent is another way of detecting correspondences. We call this kind of correspondence a constraint-based correspondence (*e.g.*, [PVSG+08 ; BBN+10 ; AT98 ; IK04]).

1.2.1.5 Model-based correspondence

Model–based correspondences are formally defined by a modeling language or a well–typed representation (*i.e.*, a meta–model). A correspondence model is a dedicated tree–based or graph–based representation with its own structure and semantics. The creation of such model is often supported by an (semi–)automatic process that uses other kind of correspondences (*e.g.*, [BHP00 ; MBJ08 ; NB04 ; PR04 ; SE06 ; ZLL09 ; ZC07 ; GW09]).

1.2.1.6 Delta representation-based correspondence

When correspondences are identified by analyzing the difference between two or more versions of the same representation, we call them delta-based correspondences. Deltas may be captured as traces, diffs (similar to version control tools), or in a dedicated model (*e.g.*, [CDRP08 ; CRE+08 ; EPK06]).

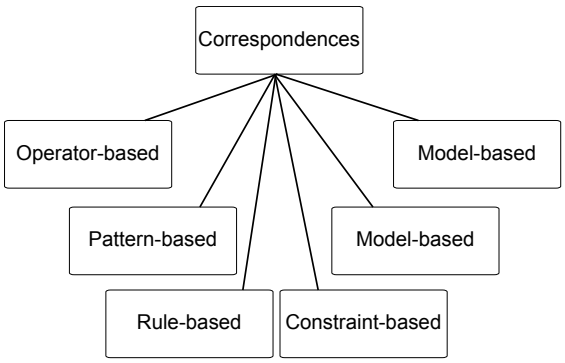


FIGURE 1.3 – Intuitive classification of correspondences

1.2.2 Interpretation

Interpretation is what Jeanneret *et al.* called the "why", that is the "...specification and requirements of the [model] composition"[JFB08, 2008, p.5]. In other words, an interpretation is what relates the correspondence relationships to the global purpose of a model composition technique. Classifying interpretation of correspondences is similar to intentional views proposed by Mens *et al.* proposed in [MMW02]. Intentional views define intention relationships between views and bound a specific meaning to

each intention relationship. Instead of considering the relationship and its interpretation as highly coupled, we rather distinguish the correspondence relationships from their interpretation to allow comparing techniques with each other and to eventually increase the reusability of the correspondence relationships and of the interpretation (see Figure 1.4). We identified three major situations in which model composition needs specific interpretations to handle a wide range of software development activities :

- The creation of multiple models about the same of closely related concepts leads to **`overlap`**. We propose seven interpretations to reconcile these models into a unified representation and to deal with divergences.
- Separating concerns of a base model into reusable fragments leads to **`cross-cutting`** models that describe ways to alter the behavior or structure. We propose three interpretations to reflect the changes on the base model.
- Run-time **`interaction`** between models may allow to fulfill a given goal. We propose three interpretations that specify how models are assembled with one another.

1.2.2.1 Overlapping

Composing a set of overlapping model with the intent to create a unified representation is usually referred as model **`merging`**. Overlapping models contain same or closely related fragments that most model merging techniques use as join point. The seven interpretations presented below are the expression of the variability in the merging process.

Equivalence An equivalence interpretation means that the model elements that are related to each other have the same semantics and structure. A merging process will create a single concept from these model elements and will aggregate their attributes and references.

Similarity Similarity is close to an equivalence interpretation in a sense that the sets of elements have a close but still different structure/semantics. Small changes are thus necessary to align the two set of models with each other to create a unified representation.

Generalization A generalization interpretation implies that one set of model elements acts as a specialization/generalization of another set of elements. In Object–Oriented design, it means that one set of model elements inherits/is–inherited–by another set of model elements. Specialization/generalization is very useful when models participates in model refinement activities.

Aggregation An aggregation interpretation means that one set of model elements "contains" another set of model elements. In Object–Oriented design, it means that one set of model elements has a **`is-part-of`** relationship to the other set of model

elements and respectively the other set of model elements has a **has-a** relationship to the source set of model elements.

Override An override interpretation means that one set of model elements is replaced by another set of model elements. This happens when a specific representation is deprecated and we replace this representation with a more adequate one.

Information Gap An information gap interpretation means that a specific set of model elements (*i.e.*, a fragment of structure) is required to properly merge the set of model elements.

Ad hoc An ad hoc interpretation applies in all other situations when the construction of the unified representation needs more complex computation such as model transformations to achieve a specific manipulation on a set of model elements.

1.2.2.2 Cross-cutting

Composing a set of cross-cutting fragments (*i.e.*, aspects) that describe ways to alter the behavior or structure of a base model is usually referred as model **weaving**. Cross-cutting aspects are associated with pointcuts that help detect the join points in the base model and in aspects to create a meaningful and consistent representation. Model weaving covers three interpretations that allow *(i)*replacing a fragment with another one, *(ii)*augmenting a fragment with the contents of another fragment or *(iii)*removing a specific fragment that should not participate in the definition of the global system.

Replace Replacing an aspect with another aspect implies that one structure or behavior defined in an aspect is not useful for a given purpose. This aspect is thus removed and a more adequate aspect is woven instead.

Augment Aspects often represent different concerns that possibly work together to achieve a specific goal. The intent of augmenting an aspect with another aspect is to mix the structure and behavior of the concerns.

Remove Since aspects are concerns of a system, we may consider that every single concern is not useful for every purpose of a given system. It means that for a specific use a of system, an aspect may be not adequate or not relevant for a specific purpose, or may clash with another aspect. Such a situation requires an operator that allows removing an aspect from the base model.

1.2.2.3 Interaction

Interaction interpretations refer to behavioral models only. Behavioral models describe the expected behavior of a given system. This behavior is represented as a set of

activities (*i.e.*, model elements are activities) which execution has to follow a specific order. The order of execution is usually described on a time basis.

In this context, correspondence relationships allow to define an order of execution between sets of activities.

Sequence A correspondence relationship that is interpreted as a sequence means that model elements from one model are executed before or after the model elements from another model.

Parallel Parallel interpretation is close to the notion of parallelism in General-purpose Programmation Language (GPL) : models elements from different models are executed at the same time.

Co-dependency Co-dependency is the state of two sets of model elements to be mutually dependent. Project planning and workflow management activities propose four relations to represent the relative execution of two sets of models :

- start-to-start means the execution of one set of model elements is allowed to start only if another set model elements has been started.
- start-to-finish means the execution of one set of model elements is not allowed to finish until another set model elements has been started.
- finish-to-start means the execution of one set of model elements is not allowed to start until another set model elements has been finished.
- finish-to-finish means the execution of one set of model elements is not allowed to finish until another set model elements has been finished.

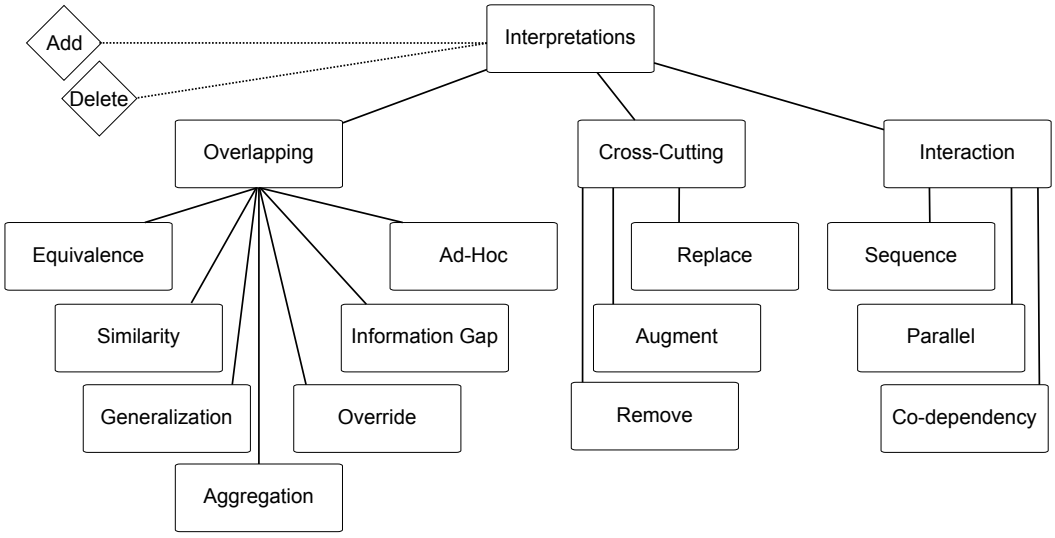


FIGURE 1.4 – Intuitive classification of interpretations. This classification includes the "add" and "delete" interpretations from Section 1.2.2.4, linked with dotted lines.

1.2.2.4 Uncategorized Interpretations

As an extension to the three situations described in the beginning of Section 1.2.2, we assume that knowledge about the internal structure of a model is not necessary to propose correspondence relationships. This situation leads to the proposition of two additional interpretations that allows inserting or removing set of model elements to produce an expected result. We provide the following semantics for these interpretations :

Add An add interpretation allows inserting a set of model elements into another set of model elements. Since the correspondence relationship is drawn between models, we may consider two situations where an add interpretation occurs :

- Experts have a tacit knowledge of the internal structure of a model and decide to add a specific set of model elements from one model into another model.
- The goal of the model composition operation is to include model elements that do not exist in the model to compose. Experts thus provide enough data to build these model elements prior to their composition.

However, if the model composition process has no data about the internal structure of the models, we expect that the set of model elements to add will have no side–effect on the original set of model elements.

Delete A delete relationship allows removing a set of model elements from another set of model elements. Similarly to the add interpretation, the remove interpretation is used in two situations as follows :

- Experts have a tacit knowledge of the internal structure of a model and decide to remove a specific set of model elements owned by one of the model involved in the correspondence relationship.
- A well–known set of model elements should be discarded from the result.

However, if the model composition process has no data about the internal structure of the models, there is no guaranty that the original set of model elements will be changed.

1.3 Validating the key elements of model composition

We propose to validate the categories of correspondence relationships and interpretation proposed in Section 1.2 against model composition techniques from the literature. Since a multitude of model composition techniques are available to support multiple software engineering activities in multiple domains of application, time has come to review and reason about model composition at a higher level of abstraction. We conducted a systematic literature review to extract and collect valuable, analyzable and reusable information that validate the proposed categories. Detecting similarities between multiple model composition techniques developed for specific purposes provide enough insights for proposing efficient ways to define and build model composition tools for specific problems not yet addressed.

In the following sections, we use the protocol template [BMA+05] proposed by Biolchini *et al.* and adapted from the work of Kitchenham [Kit04] to describe the systematic review protocol (see Figure 1.5).

Section 1.3.1 details the systematic literature review protocol and process. For presentation purposes, we regroup data as follows : *(i)*Section 1.3.1.1 presents problems, research objectives and hypotheses that this review should help respectively to answer, to corroborate and to validate ; *(ii)*Section 1.3.1.2 lists keywords and synonyms that literature usually use to refer to model composition ; *(iii)*Section 1.3.1.3 details requirements which characterize the relevance of the information that we gathered from articles ; *(iv)*Sections 1.3.1.4 and 1.3.1.5 presents the data sources, the selection process, the selection review and the initial list of articles selected for the systematic literature review, with regard to selection criteria ; *(v)*Section 1.3.1.6 presents biases and threats to the internal validity of the current systematic literature review ; *(vi)*Section 1.3.1.7 details how we choose to list extracted data.

Sections 1.3.2 to 1.3.4 presents the body of data that we extracted from the application of the systematic literature protocol. Section 1.4 ultimately discusses results and expectations with regard to the research objectives.

1.3.1 Systematic Review Protocol

1.3.1.1 Research Objectives

The systematic literature review should provide answers to the following research questions :

Q1 Do all model composition techniques use correspondences between models or model elements and is there a specific meaning given to these correspondences ?

- **Q1a** Do the correspondences fit into the proposed categories ?
- **Q1b** Do the meanings reflect the category proposed interpretations ?

Q2 Do model composition goals and purposes cover a wide range of software development activities ?

With regard to the intuitive categories of correspondence and interpretation that we proposed in Section 1.2, we expect to validate the following hypotheses named after the questions they refer to :

H1.0 Every model composition technique uses a set of correspondence relationships and a specific interpretation given a specific problem

H1a.1 Correspondence relationships from the literature should correspond to the proposed categories

H1b.1 Interpretations from the literature should correspond to the proposed categories

H1.3 The list of correspondence and interpretation is exhaustive and precise enough to cover all model composition techniques

H1.4 We observe an even distribution of the model composition techniques in the proposed categories

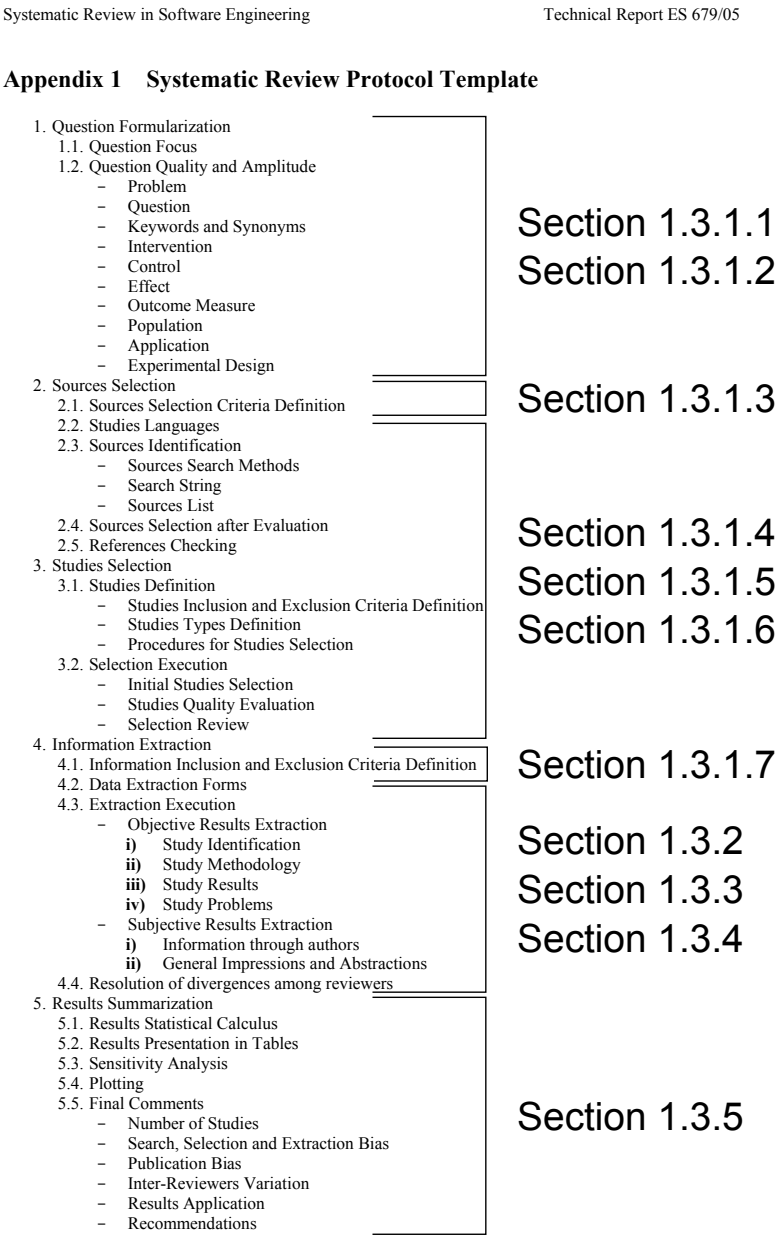


FIGURE 1.5 – Template proposed by Biolchini for the definition and execution of systematic review protocol. For presentation purposes, we regroup the description of the protocol, the extracted data, and the results into nine sections referenced on the figure.

H2.0 A wide range of the software life-cycle activities are supported by model composition techniques

1.3.1.2 Model Composition and Synonyms

The selection of a relevant set of articles relies on a proper definition of keywords and synonyms to look for model composition techniques. The term *model* is not precise enough to produce interesting results by itself and since some model composition techniques often refer the the higher level of abstraction of a model to present their approaches, we need to add metamodel as part of the keywords of interest. Looking for *model composition* brings out a lot of results since the term *composition* is overloaded in the literature. However, this helps to capture a wider range of techniques. In addition to *composition*, authors often use the terms *merge(-ing)*, *fusion*, or *weaving* which corresponds to specific applications of model composition in Software Engineering (SE). Some of them are discussed in Section 1.3.2

1.3.1.3 Valuable Information Characterization

The identification of relevant information in articles must provide answers to the research objectives that we define in Section 1.3.1.1. The characterization of the relevance of the information is represented as a set of requirements as follows :

R1 Authors use an explicit or an implicit representation of correspondences between models and/or metamodels.

R2 A model composition technique has a specific purpose for which it was designed for

R3 A set of correspondences has at least one meaning (interpretation) which is presented in the paper

From these requirements, we should gather enough data to classify existing model composition techniques with regard to the categories of correspondence relationships and interpretations proposed in Section 1.2. Classification should help comparing model composition approaches with one another. Exploring the uses of model composition in various software development activities is the first step in proposing an interpretive lens to identify the core concepts of model composition. This will help software designers, software architects and tool providers to reuse existing techniques for their own needs.

1.3.1.4 Articles Selection Criteria and Methods

Articles of interest for this systematic literature review have been published in software engineering conferences. Available and properly referenced technical reports about model composition techniques are also valid candidates. The articles are mainly written in English but French articles were also allowed. Articles that we discuss in this review were extracted from electronic data sources including the ACM Digital Library[Ass11], the IEEE Xplore Digital Library [IEE11b], the RefDoc

Service [Ini11], the open access archive named HAL [CCS11], the IEEE Computer Society Digital Library, [IEE11a], the CiteSeerX Digital Library [Pen11], the IBM Technical Journals [IBM11], and the book series of Lectures Notes in Computer Science available on SpringerLink [Spr11].

These data sources were crawled using web-based services to retrieve the references of the articles to include in the selection. We use mainly the Google Scholar [Goo11] service but also to some extent the Researchr [Res11], the SciVerse ScienceDirect [Els11], the Oxford Journals [Oxf11], the ArnetMiner [KEG11] and CiteULike [Ove11] websites, and several authors or research team projects homepages.

We perform our selection of articles using the following string that conforms to the explanations of Section 1.3.1.2 :
model composition OR metamodel OR merging OR fusion OR weaving OR combination.

The articles that are selected for this review are only those which discuss model composition operators or methods and techniques leading to the definition of such model composition operator. The selection process starts by running the search on the identified web-based search engines. Relevant articles are identified first by their title and second by reading the abstract. If the articles do not discuss model composition techniques or discuss the specific kind of relation in UML called *composition*, the article is not included in the selection. Once a paper has been selected for review, it is read entirely to capture valuable information. The adequateness of articles regarding selection criteria is checked by a main reviewer and at least by two more reviewers who have an expertise on the domain of model composition.

1.3.1.5 Study selection

Among the articles that match the research criteria, some of them do not propose any model composition technique or approach, but deal instead with industrial requirements for efficient model merging [BE09], with internal properties of a specific composition operator [PB09], or even with identifying compatibility issues with UML2 Package Merge [ZDD06]. The articles that were not proposing any model composition technique or approach explicitly were removed from the set of selected articles for this study. The complete list of articles that we selected for this systematic literature review is presented in Table 1.1

Key	Author(s)	Title	Year
[ACL+10]	Acher, Collet, Lahire et al.	« Managing Variability in Workflow with Feature Model Composition Operators »	2010
[ACL+09]	Acher, Collet, Lahire et al.	« Composing Feature Models »	2009
[AEC+07]	Anwar, Ebersold, Coulette et al.	« Vers une approche à base de règles pour la composition de modèles. Application au profil VUML. »	2007
[AJT+09]	Apel, Janda, Trujillo et al.	« Model Superimposition in Software Product Lines »	2009
[ASM+10]	Alférez, Santos, Moreira et al.	« Multi-view Composition Language for Software Product Line Requirements »	2010
[AT98]	Aksit et Tekinerdogan	Solving the Modeling Problems of Object-Oriented Languages By Composing Multiple Aspects Using Composition Filters	1998
[BA00]	Bergmans et Aksit	« Composing Software from Multiple Concerns : A Model and Composition Anomalies »	2000
[BBN+10]	Bensalem, Bozga, Nguyen et al.	« Compositional verification for component-based systems and application »	2010
[BCR05]	Boronat, Carsí et Ramos	« MOMENT : a formal MOdel manageMENT tool »	2005
[BCR+07]	Boronat, Carsí, Ramos et al.	« Formal Model Merging Applied to Class Diagram Integration »	2007
[BHP00]	Bernstein, Halevy et Pottinger	« A vision for management of complex models »	2000
[BKB+08]	Barais, Klein, Baudry et al.	« Composing Multi-view Aspect Models »	2008
[BLTN10]	Brottier, Le Traon et Nicolas	« Composing Models at Two Modeling Levels to Capture Heterogeneous Concerns in Requirements »	2010
[BSM+07]	Balasubramanian, Schmidt, Molnar et al.	« Component-Based System Integration via (Meta)Model Composition »	2007
[BTF05]	Balaban, Tip et Fuhrer	« Refactoring support for class library migration »	2005

Continued on next page

Key	Author(s)	Title	Year
[BWH10]	Boucké, Weyns et Holvoet	« Composition of architectural models : Empirical analysis and language support »	2010
[Bar08]	Bartelt	« Consistence preserving model merge in collaborative development processes »	2008
[Bel04]	Belapurkar	Use AOP to maintain legacy Java applications	2004
[Ber03]	Bernstein	« Applying Model Management to Classical Meta Data Problems »	2003
[CBJ10]	Clavreul, Barais et Jézéquel	« Integrating Legacy Systems with MDE »	2010
[CDK+07]	Curbera, Duftler, Khalaf et al.	« Bite : Workflow Composition for the Web »	2007
[CDRP08]	Cicchetti, Di Ruscio et Pierantonio	« Managing Model Conflicts in Distributed Development »	2008
[CRE+08]	Cicchetti, Ruscio, Eramo et al.	« Automating Co-evolution in Model-Driven Engineering »	2008
[CRR+07]	Chitchyan, Rashid, Rayson et al.	« Semantics-based composition for aspect-oriented requirements engineering »	2007
[Cla02]	Clarke	« Extending standard UML with model composition semantics »	2002
[CSN08]	Chen, Sztipanovits et Neema	« Compositional Specification of Behavioral Semantics »	2008
[FDV07]	Fabro, Didonet et Valduriez	« Semi-automatic model integration using matching transformations and weaving models »	2007
[DRMM+10]	Di Ruscio, Malavolta, Muccini et al.	« Developing next generation ADLs through MDE techniques »	2010
[EPK06]	Engel, Paige et Kolovos	« Using a Model Merging Language for Reconciling Model Versions »	2006
[ES06]	Emerson et Sztipanovits	« Techniques for metamodel composition »	2006
[FBB+07]	Fleurey, Breton, Baudry et al.	« Model-Driven Engineering for Software Migration in a Large Industrial Context »	2007
[FBF+08]	Fleurey, Baudry, France et al.	« A Generic Approach for Automatic Model Composition »	2008
[DFB+05b]	Didonet, Fabro, Bézivin et al.	« AMW : a generic model weaver »	2005

Continued on next page

Key	Author(s)	Title	Year
[FFR+07]	France, Fleurey, Reddy et al.	« Providing Support for Model Composition in Metamodels »	2007
[FGF+08]	Fritzsche, Gilani, Fritzsche et al.	« Towards Utilizing Model-Driven Engineering of Composite Applications for Business Performance Analysis »	2008
[GJ05]	Groenmo et Jaeger	« Model-driven semantic Web service composition »	2005
[GKR+08]	Grönniger, Krahn, Rumpe et al.	« MontiCore : a framework for the development of textual domain specific languages »	2008
[GS03]	Gössler et Sifakis	« Composition for Component-Based Modeling »	2003
[GW09]	Giese et Wagner	« From model transformation to incremental bidirectional model synchronization »	2009
[HHJ+08]	Henriksson, Heidenreich, Johannes et al.	« Extending grammars and metamodels for reuse : the Reuseware approach »	2008
[HKG+10]	Hemel, Kats, Groenewegen et al.	« Code generation by model transformation : a case study in transformation modularity »	2010
[IK04]	Ivkovic et Kontogiannis	« Tracing evolution changes of software artifacts through model synchronization »	2004
[JKB+06]	Jackson, Klein, Baudry et al.	« Executable Aspect Oriented Models for Improved Model Testing »	2006
[JWE+07]	Jayaraman, Whittle, Elkhodary et al.	« Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis »	2007
[JZF+09]	Johannes, Zschaler, Fernández et al.	« Abstracting Complex Languages through Transformation and Composition »	2009
[Jez08]	Jezequel	« Model driven design and aspect weaving »	2008
[KAAK09]	Kienzle, Al Abed et Klein	« Aspect-oriented multi-view modeling »	2009
[KHJ06]	Klein, Hélouet et Jézéquel	« Semantic-based Weaving of Scenarios »	2006
[KJP05]	Klein, Jézéquel et Plouzeau	« Weaving Behavioural Models »	2005

Continued on next page

Key	Author(s)	Title	Year
[KM10]	Kelsen et Ma	« A Modular Model Composition Technique »	2010
[KPP06]	Kolovos, Paige et Polack	« Merging Models with the Epsilon Merging Language (EML) »	2006
[KUL+10]	Krause, Uhlendorf, Lubitz et al.	« Annotation and merging of SBML models with semanticSBML »	2010
[LNK+01]	Ledeczi, Nordstrom, Karsai et al.	« On metamodel composition »	2001
[LP03]	Liang et Paredis	« A port ontology for automated model composition »	2003
[Let07]	Letkeman	Comparing and merging UML models in IBM Rational Software Architect : Ad-hoc modeling - Fusing two models with diagrams	2007
[MBFF10]	Mosser, Blay-Fornarino et France	« Workflow Design Using Fragment Composition »	2010
[MBJ08]	Morin, Barais et Jézéquel	« Weaving Aspect Configurations for Managing System Variability »	2008
[MBJ+07]	Morin, Barais, Jézéquel et al.	« Towards a Generic Aspect-Oriented Modeling Framework »	2007
[MKB+08]	Morin, Klein, Barais et al.	« A Generic Weaver for Supporting Product Lines »	2008
[MBN+09]	Morin, Barais, Nain et al.	« Taming Dynamically Adaptive Systems using models and aspects »	2009
[MMP+10]	Malavolta, Muccini, Pelliccione et al.	« Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies »	2010
[MPL+09]	Morin, Perrouin, Lahire et al.	« Weaving Variability into Domain Metamodels »	2009
[Mos10]	Mosser	« Behavioral Compositions in Service-Oriented Architecture »	2010
[NB04]	Nahrstedt et Balke	« A taxonomy for multimedia service composition »	2004
[NM00]	Noy et Musen	« PROMPT : Algorithm and Tool for Automated Ontology Merging and Alignment »	2000
[NSC+07]	Nejati, Sabetzadeh, Chechik et al.	« Matching and Merging of Statecharts Specifications »	2007
[OMK09]	Oldevik, Menarini et Krüger	« Model Composition Contracts »	2009

Continued on next page

Key	Author(s)	Title	Year
[OO07]	Oliveira et de Oliveira	« A Guidance for Model Composition »	2007
[PBB+09]	Perrouin, Brottier, Baudry et al.	« Composing Models for Detecting Inconsistencies : A Requirements Engineering Perspective »	2009
[PBC+11]	Parra, Blanc, Cleve et al.	« Unifying Design and Runtime Adaptations Using Aspect Models »	2011
[PDCS+01]	Paredis, Diaz-Calderon, Sinha et al.	« Composable Models for Simulation-Based Design »	2001
[PGP+07]	Pons, Giandini, Perez et al.	« An Algebraic Approach for Composing Model Transformations in QVT »	2007
[PR04]	Park et Ram	« Information systems interoperability : What lies beneath ? »	2004
[PRB+09]	Pedro, Risoldi, Buchs et al.	« Composing Visual Syntax for Domain Specific Languages »	2009
[RCE08]	Rubin, Chechik et Easterbrook	« Declarative approach for model composition »	2008
[SE06]	Sabetzadeh et Easterbrook	« View merging in the presence of incompleteness and inconsistency »	2006
[SFS+08]	Sánchez, Fuentes, Stein et al.	« Aspect-Oriented Model Weaving Beyond Model Composition and Model Transformation »	2008
[SY10]	Shonle et Yuen	« Compose & conquer : modularity for end-users »	2010
[TT08]	Tansey et Tilevich	« Annotation refactoring : inferring upgrade transformations for legacy applications »	2008
[PVSG+08]	von Pilgrim, Vanhooff, Schulz-Gerlach et al.	« Constructing and Visualizing Transformation Chains »	2008
[WJ08]	Whittle et Jayaraman	« MATA : A Tool for Aspect-Oriented Modeling Based on Graph Transformation »	2008
[WS08]	Weisemöller et Schürr	« Formal Definition of MOF 2.0 Metamodel Components and Composition »	2008
[Wac07]	Wachsmuth	« Metamodel Adaptation and Model Co-adaptation »	2007
[Wag08]	Wagelaar	« Composition Techniques for Rule-Based Model Transformation Languages »	2008

Continued on next page

Key	Author(s)	Title	Year
[XLH+07]	Xiong, Liu, Hu et al.	« Towards automatic model synchronization from model transformations »	2007
[ZC07]	Zhang et Cheng	« Towards Re-engineering Legacy Systems for Assured Dynamic Adaptation »	2007
[ZLL09]	Zhang, Li et Liu	« An Approach for Model Composition and Verification »	2009

TABLE 1.1 – Full list of selected articles

1.3.1.6 Study quality assessment

Three kinds of threats to the internal validity of the study have been identified. We avoided selection biases by using the same search string on every session of data sources crawling. However, the identification of articles of interest has been achieved by reading the title and abstract of articles and it is possible that some references were involuntarily not selected. The selection of articles relies on papers published in international conferences on software engineering and thus we cannot avoid publication biases that prevent accessing non-published articles of interest. Information of interest is captured by a manual process of reading the articles entirely. Because of the nature of data, this process is both systematic (search for keywords) and subjective (provide a context and extract data about purpose and interpretation mainly. This process is even more difficult because of the use of wordings specific to each authors. The process of peer reviewing is proposed to tackle this issue. The last threat to validity is related with the application of Biolchini *et al.* guidelines [BMA+05]. We closely followed the guidelines for conducting this experiment to avoid misinterpreting some information.

1.3.1.7 Data Extraction

As a guiding thread for further discussion on model composition, we propose to organize the body of knowledge about model composition techniques regarding the main activities in the software life-cycle. We do not follow any existing development or life-cycle model such as incremental, V, Y or evolutionary life-cycles to properly cover a significant part of the domain and to avoid introducing any bias. We focus on three of four main activities in software engineering which are : *(i)*Design (see Section 1.3.2) as the full range of activities for building software and systems from scratch or by assembling existing reusable software artifacts ; *(ii)*Verification and Validation (see Section 1.3.3) as the set of activities for validating software, ensuring that it behaves as expected and that it answers the end–user expectations ; *(iii)*Evolution (see Section 1.3.4) as the set of activities that deal with software maintenance, fixing errors, providing new functionalities, re-engineering existing systems or versioning software artifacts to ensure traceability and information backup. In the following sections, we

investigate various model techniques under the scope of the software engineering activities that authors claim to address. The following sections present articles of interest. Data presentation first focuses on the definition of terms that represent subactivities of a given software life-cycle activity. Then we propose a detailed presentation of articles of interest if any for each category of correspondence relationship. The selection of articles for which we provide more details is based on *(i)*the relevance of the approach regarding both the category of the correspondence relationships and the category of interpretation and on *(ii)*the number of citations gathered from electronic data sources that we referenced in Section 1.3.1.4.

1.3.2 Model Composition for Systems Design

This section lists the contributions in the literature that deal with the general activity of designing software. We consider the following activities as subtypes of the design activity :

Model Composition  The term "composition" is overloaded in the MDE community, so we choose to focus on all activities that enable to build a system from the union of several independent and dependent software artifacts. This definition includes the usual terms of merging, union, weaving or fusion along with any activity whose intent is to create a software from reusable chunks of other systems.

Derivation  Derivation is borrowed from the domain of SPL, where applications are built from the selection of variants. These variants may be represented in a feature model / diagram for instance where designers choose which ones are part of one product. Derivation is different from the definition of model composition above since "features" are specified in a specific formalism that allows identifying relationships between features such as optionality, coupling, mutual-exclusion, etc.

Orchestration  The term "composition" when related with the domain of services engineering, is connoted with the notion of assembly. For instance, Software as a Service (SaaS) promotes the use of multiple services and their interaction to support user requirements. While some techniques in the literature address the specific problem of merging internal behavior of services to create new services, composition and composite services usually refer to what is called orchestration in the domain of workflow. Orchestration (centralized decision) or choreography (decentralized decision) is the activity of arranging services execution with one another to create a fully running process.

Integration  is another engineering activity focused on building software for a given purpose. The main difference with the three categories above is that it generally manipulates existing systems as wholes. In other words, this activity produces new systems from the "interaction" of several independent and running systems.

5. Each category is assigned a symbol to ease reading and understanding

The following subsections will give details about techniques found in the literature that address these four specific activities. Each description is completed by the description of a selected subset of techniques that support this activities.

1.3.2.1 Composition

Model composition is the activity of manipulating model elements from at least two source models to produce a unified representation that may be serialized (case of merging, union or fusion) or made only available at run-time (weaving for instance). Usually the source models are specified using the same meta–language, *i.e.*, the source models share common concepts such as entities or relationships. Most research in the literature is focused on this specific goal of being able to produce systems from the union of existing models, automatically or semi-automatically. The reason why there are so many approaches is mainly because of the inherent complexity of the domain(s) considered, the language(s) that define the models, and the result of the composition that may vary depending on the user expectations. As an illustration of model composition, we give details about six techniques as follows.

Operator-based Correspondence

"On Metamodel Composition" In [LNK+01], Ledeczi *et al.* proposed three UML-based operators to build new DSML from existing metamodels. These operators support model modifications such as equivalence, implementation inheritance and interface inheritance. The equivalence operator helps to define and execute a full union of two UML class objects. Implementation inheritance captures all parent's attributes and containment relationships and "inject" them into the corresponding class. Interface inheritance only deals with capturing relationships other than containment and with binding them to another class.

Similar model composition approaches are proposed by Boronat *et al.* [BCR05; BCR+07], Kelsen and Ma [KM10], Pedro *et al.* [PRB+09], Henriksson *et al.* [HHJ+08] and Grönniger *et al.* [GKR+08].

Pattern-based Correspondence

"Extending standard UML with model composition semantics" In the broader scope of model integration [Cla02], Clarke proposed a technique for aligning artifacts written in different paradigms. Along with the definition of relationships as first-class entities, she proposed composition patterns to indicate how model elements from one model should be overridden by model elements from another model. These patterns are represented as a list of elements (classes,operations,attributes) from the model.

Similar model composition approaches are proposed by Wagelaar [Wag08], Johannes *et al.* [JZF+09],Fleurey and al. [FBF+08], Letkeman [Let07], Nejati *et al.* [NSC+07], Mosser *et al.* [MBFF10], Rubin *et al.* [RCE08], Noy and Musen [NM00], Emerson and Sztipanovits [ES06], Oliveira and de Oliveira [OO07], France *et al.* [FFR+07], Krause *et*

al. [KUL+10], Boucké *et al.* [BWH10], Kienzle *et al.* [KAAK09], Sánchez *et al.* [SFS+08], Klein *et al.* [KHJ06 ; KJP05], Jézéquel [Jez08], Morin *et al.* [MBJ+07], Weisemöller and Schürr [WS08], Apel *et al.* [AJT+09], and Acher *et al.* [ACL+10].

Rule-based Correspondence

"Semantics-based composition for Aspect-Oriented Requirements Engineering" Chitchyan *et al.* approach for composing requirements ([CRR+07]) is using a set of rules to describe pointcuts and joinpoints in the requirements specifications. Based on a dedicated language called Requirements Description Language, they support requirements composition using a set of predefined operators for composition. From a step of natural language processing, they detect "relationships" and "actions" classes of verbs to which they bind specific composition operations. Semantic queries are then used to select concerns that act as the constraint to be imposed. Finally, intersections between constraints enable to find new requirements that come from the composition of existing requirements.

Similar model composition approaches are proposed by Kolovos *et al.* [KPP06], Whittle and Jayaraman [WJ08], Jayaraman *et al.* [JWE+07], Brottier *et al.* [BLTN10], and Oldevik *et al.* [OMK09].

Model-based Correspondence

"A vision for management of complex models" Bernstein *et al.* model-based approach ([BHP00]) for management for complex models relies on high-level operations on models and models mappings to manage changes and transformations of models. The main part of the paper is focused on providing a formal definition and semantics of correspondences relationships (or mappings) between model elements. Composition has two meaning in this paper. First they propose composition between mappings to provide mappings transitivity (map(A,B) composed with map(B,C) offers map(A,C)). Second, they proposed a definition of a merge operator between models. They propose to use an additional input to the merge operator, using a delta model of the source and target models, to drive the merge activity. Existing implementations of the merge operator are very similar to the definition that they proposed, for instance in [FFR+07] and [ZDD06].

Similar model composition approaches are proposed by Fabro *et al.* [DFB+05b], Wachsmuth [Wac07], Liang and Paredis [LP03], and Paredis *et al.* [PDCS+01].

Delta-based Correspondence

"Managing Model Conflicts in Distributed Development" In [CDRP08], Cicchetti *et al.* proposed a DSL for syntactic and semantics conflicts. Conflicts are relationships between difference models, i.e delta models. Delta models are easy to process

to identify structural conflicts and scenarios are thereafter used to identify semantic-related patterns for conflicts. From conflicts identification they build a difference model and add new model elements to model additions, deletions and changes. From this representation, authors propose to achieve sequential and parallel combination. Sequential composition means merging modification conveyed by the first model with the second one. About parallel composition, they distinguish two kinds : parallel dependent and parallel independent. Parallel independent means that changes do not effect the same model elements, so they apply the merge process they proposed for sequential combination. About parallel dependent, they claim it requires additional analysis to determine an order in the composition process. Automation of the composition process is achieved through the use of a set of Object-Constraint Language (OCL) constraints with post-conditions that return collections of elements

1.3.2.2 Derivation

Derivation is a relatively new domain of research in software engineering. The demand of fast-paced development and production of new software from existing products increases in a lot of domains including but not limited to telephony services or on-board car systems. Derivation is a important activity in the context of software product lines. It extends the traditional body of knowledge about software building from reusable artifact in a sense that all end-products are related to one another with the same global characteristics and expectations but tailored for specific usages. The following sections illustrates derivation examples found in the literature.

Operator-based Correspondence

"Multi-view Composition Language for Software Product Line Requirements" In the context of SPL, Alferez *et al.* propose an approach [ASM+10] for composing elements defined in separated and heterogeneous requirement models using a simple set of operators. Using a DSL called Variability Modeling Language for Requirements Engineering to support the definition of relationships between SPL features from various feature models. They propose also a set of composition operators such as insert, replace or remove that they define as a specific graph-transformation rule. Those operators are then use to build SPL products from feature models.

A similar model composition approach is proposed by Morin *et al.* [MPL+09].

Pattern-based Correspondence

"Model-Driven Design and Aspect Weaving" In [Jez08], Jezequel explores the relationship between modeling and aspect weaving with regard to the large domain of software product line. The author first pinpoints the fact that modeling is not only abstracting but also analyzing a domain in a sense of separating various cross-cutting and non cross-cutting concerns. Those concerns becomes then aspects that designers can weave with one another to build a detailed design model. As the author said,

mixing and dealing with multiple concerns at a time to build a system is something that engineers are able to do but what is really challenging is to quickly, cheaply and safely change variants of aspects to build several products in the context of product lines. Along the line of several papers about aspect weaving, Jezequel describes what aspects are composed of and how they are composed with one another. Concepts of joinpoints, pointcuts and advices are described as general aspect-weaving concepts and then instantiated on the specific example of sequence diagrams where additional issues arise. One of them is about temporality of the joinpoint detection or in other words when a specific pointcut (as q sequence of messages) can be detected. Because of interleaved messages that may occur when we weave multiple aspects, a specific joinpoint may not be replaced by the advice without losing some information. To solve this problem, the author introduces a specific operator called amalgamated sum which helps capturing commonalties between a base model and the advice.

Similar model composition approaches are proposed by Apel *et al.* [AJT+09], Morin *et al.*[MBJ08 ; MKB+08 ; MPL+09], and Parra *et al.* [PBC+11].

Rule-based Correspondence

"Model Composition in Product Lines and Feature Interaction Detection Using Critical-Pair Analysis" Jayaraman *et al.* proposed the MATA approach to compose variant features in the context of SPL [JWE+07]. Variants are modeled as UML models and MATA allows automatic composition of such models. Relationships between variants are captured in the MATA language and the ordering of the features may be describe to provide additional detection of structural conflicts between models. Since MATA defines rules on graphs, they use AGG [Tae04] to execute each rule and produce a composed model that is then converted back in the UML language.

A similar model composition approach is proposed by Alférez *et al.* [ASM+10].

Model-based Correspondence

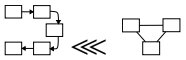
"Weaving Aspect Configurations for Managing System Variability" In [MBJ08], Morin *et al.* propose to tackle the issue of limited reusability of aspects by integrating variability mechanisms into aspects. Using AOM concepts such as joinpoints, pointcuts and advices, they propose a template model that expresses joinpoints in a more generic way and that relaxes existing metamodel constraints to increase flexibility in the definition of join points. The weaving process uses adapters that describe what is going to be composed, where it should be composed (template model) and how it is going to be composed (composition protocol). The composition protocol is described by an adaptation model which contains predefined adaptations operations such as makeUnique, create, clone, setProperty and unsetProperty. Variability is addressed by mixing SPL concepts with the adaptation model. Adapters can support variants, optional features or dependency constraints. Bindings between the adaptation model

and actual model elements is necessary for the composition protocol to properly weave model all together.

A similar model composition approach is proposed by Parra *et al.* [PBC+11].

1.3.2.3 Orchestration

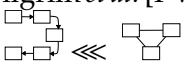
This idea of orchestration has been an underlying activity since the early days of Computer Science and computing. The management of software artifacts in terms of precedence and successiveness in time has been undergone from the very day the first computer programs emerged until now with more complex abstractions but the basic idea is quite the same. The term "orchestration" is part of the domain that deals with modeling workflows, business processes and so on. Conforming to the previous definition, orchestration is the definition of how artifacts or fragments "interact" with one another, considering these fragments as black-boxes. The following sections illustrate orchestration examples found in the literature.



Operator-based Correspondence

"An Algebraic Approach for Composing Model Transformations in QVT" Pons *et al.* proposed an approach [PGP+07] for the orchestration of model transformation in QVT [BBB+]. Their approach is based on using the algebraic theory of problems as a mathematical foundation. The theory of problems defines a problem as a quadruple <D,R,q,I> where D is the data domain, R is the result domain, q is a binary relation on $D \times R$ and I is a set of instances of interest in the scope of the problem. By comparison with QVT, they define QVT relations as problems and QVTOp [OMG07] mappings as solutions. Based on this analogy, they propose a set of composition operators such as union, sequence or fork between QVT relations and QVTOp mappings. Composition operators are included in what they call a composition calculator that automatically compose existing QVT-based transformations.

A similar model composition approach is proposed by von Pilgrim *et al.* [PVSG+08].



Pattern-based Correspondence

"Code Generation by model transformation : a Case Study in Transformation Modularity" In [HKG+10], Hemel *et al.* propose techniques for improving separation of concerns in the implementation of code generators for DSMLs. Using model composition between transformation rules, they support code generation and model-to-model transformation. Composition operators between rules represent composition strategies such as sequential composition, deterministic choices, identifiers composition, etc. In their approach, correspondences between rules has a special meaning that depends on the type of elements involved. When correspondence is achieved through a rule, they composition means orchestration or weaving. When correspondence is a specific operator "#" or a specific transformation, then composition refers to weaving or merging.

A similar model composition approach is proposed by Mosser [Mos10].

Rule-based Correspondence

"Bite : Workflow Composition for the Web" Curbera *et al.* present an approach to deliver composition capabilities in a resource-centric environment [CDK+07], "such that data and behavioral compositions are seamlessly supported by a common workflow-oriented model". Bite is a minimalist choreography language and runtime that relies on the concepts of activity and dependency link between activities. Bite proposes a set of basic constructs that represent actions in Web workflows such as receiving and replying to a message, invoking a service or waiting for a fixed time for a flow to terminate. With regard to this definition, Bite natively supports web data flows in which several processing steps are connected with one another by data dependencies. Their execution is considered as a data flow composition that executes each one of them synchronously. Interactive flows, meaning web applications that ask users to provide data, are widespread and Bite supports their definition and execution by proposing asynchronous execution and supporting seamless integration of Web interaction models and interactions.

Similar model composition approaches are proposed by Chitchyan et a. [CRR+07], Pons *et al.* [PGP+07], and Hemel *et al.* [HKG+10].

Constraint-based Correspondence

"Constructing and Visualizing Transformation Chains" Based on UniTI [VAVB+07], a unified representation of transformations, von Pilgrim *et al.* propose an approach for representing dependencies between transformations and products of these transformations [PVSG+08]. They use traceability links to propose user-friendly visualization of chains of transformations. UniTI framework is based on three principles namely (1) black-boxing by hiding internal behavior of transformations, (2) external specification of provided and required interfaces for each transformation, and (3) composition of transformations that they consider as reusable building blocks. Through the definition and use of well-defined interfaces for transformations, interconnecting them in the way component can be connected with one another is pretty straight-forward.

Model-based Correspondence

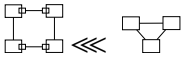
"A Taxonomy for Multimedia Service Composition" In [NB04], Nahrstedt and Balke proposed to bridge the multimedia domain with the Web Services domain to benefit from web service composition techniques. They propose a taxonomy framework that includes metrics about web and multimedia integration. The framework also provide partitioning the composition space into successive composition, concurrent

composition and hybrid composition. This taxonomy of partitions is refined with concepts of performance, content and infrastructural support metrics to allow decomposing multimedia composition problems. To give a few more details about composition, successive composition is a composition of functionally dependent services that are invoked sequentially. Concurrent composition is a composition of independent services that are executed in parallel. Hybrid composition combines both successive and parallel composition of services executions demonstrating functional dependencies and time-based synchronization. However, authors do not give details in the paper about how the composition is achieved.

A similar model composition approach is proposed by Bernstein *et al.* [BHP00].

1.3.2.4 Integration

Integration is the design activity that allows communication between systems that were not design to communicate. Communication is necessary to build a new system from existing software or at least a whole that behaves like a single system. A more usual term that describes interaction between existing system is interoperability. While integration as a goal may be achieved with the use of composition, derivation or orchestration, it is an activity that should take into account the characteristics of dealing with existing systems. Those systems have been developed independently, which means that it is pretty sure that they were not implemented using the same technologies. Here comes the difficult task of making heterogeneous systems interact. Second, building a new system which integrates existing systems does not mean discarding existing ones. Integration should then be conducted as if systems were gray-boxes, where we have knowledge of some of the internal machinery of these systems, but still we do not have full control on their implementation. It means that it is very often forbidden to modify existing systems to implement the integration. We must then build some intermediate and consensual representation that we share between the systems to be integrated. We usually consider two major ways to define the intermediate representation [6] : (1) message-based integration captures the calls and events from one system and adapts these calls into the representation of another system ; (2) Data transformation integration requires to build an intermediate format with serialization and unserialization from the legacy systems own data. The following sections illustrate integration examples found in the literature.



Operator-based Correspondence

"Applying Model Management to Classical Meta Data Problems" In [Ber03], Bernstein propose an extension to previous work [BHP00] composition operators and applies them on integration, evolution and round-trip engineering activities. From the definition of mappings as a representation encompassing two morphisms from the mapping to the input model, he uses these mappings to define an algebra and

6. http ://en.wikipedia.org/wiki/Enterprise_application_integration#Patterns

implement specific model operators for model manipulation. Examples of operators are : (1) Match that creates a mapping between two input models, (2) Diff that computes a difference between a model to a given mapping, (3) Merge that returns a copy of all the model elements of the input model after collapsing equivalent elements into a single element in the output model, (4) Compose that combines two existing mappings. Additional operators such as Apply, Copy, ModelGen or Enumerate allows performing specific operations on models and/or mappings.

Similar model composition approaches are proposed by Boronat *et al.* [BCR05 ; BCR+07], Gössler and Sifakis [GS03], and Shonle and Yuen [SY10].

Pattern-based Correspondence

"Component–Based System Integration via (Meta)Model Composition" Balasubramanian *et al.* propose a model composition approach to integrate heterogeneous systems [BSM+07]. The integration process creates a new DSML from existing DSML and is supported by their System Integration Modeling Language (SIML) tool. They propose to map interfaces from different technologies to achieve integration. Mapping of interfaces involves (1) "mapping datatypes from source to target technology", (2) "mapping exceptions from source to target technology" and (3) "mapping languages", that is "mapping datatypes while accounting for differences in languages at the same time". Differences between technologies should be considered at a low-level of abstraction to allow providing mappings for protocols, to allow discovering components and to allow providing mappings for Quality of Service (QoS) mechanisms. The composite DSML obtained by SIML defines the semantics of the integration process which might include reconciling differences between technologies. Elements from sub-DSMLs are black-boxes which behavior and structure cannot be modified.

Rule-based Correspondence

"Compose & Conquer : Modularity for End-Users" In [SY10], Shonle and Yuen propose an innovative approach for providing end-users with on-the-fly solutions that may possibly involve complex computation. The list of solutions is built from a "creativity engine" that selects existing modules and compose the modules with one another to provide integrated solutions. Selection of modules and libraries and the various ways to compute the expected result are supported by ontology-based computation. The approach is similar to code mashups since it requires adapters for every candidate library or module but they do not need to know the modules names. They presented two examples, one about producing a PDF document from a JPEG document in various ways, and the other example dealt with crawling a website to get the phone number, address and city of a restaurant from data scattered in various places and various content types.

Similar model composition approaches are proposed by Bernstein [Ber03] and Balasubramanian *et al.* [BSM+07].

Model-based Correspondence

"Information Systems Interoperability : What Lies Beneath ?" Park and Ram proposed a framework for helping the integration of databases [PR04]. The framework is called Conflict Resolution Environment for Autonomous Mediation (CREAM) is designed to identify semantically related data from different databases and resolve semantic conflict among them before the integration process. The framework relies on schema for modeling data and shareable data from the databases, ontologies for capturing contextual knowledge, mappings between schema to generate valid local queries, relationships between concepts of knowledge for semantic reconciliation and mediators to detect conflicts between schema and provide data access and transformation.

Similar model composition approaches are proposed by Liang and Paredis [LP03], Paredis *et al.* [PDCS+01], Malavolta *et al.* [MMP+10], Clavreul *et al.* [CBJ10], and Del Fabro and Valduriez [FDV07].

1.3.3 Model Composition for the Verification and the Validation of Systems

This section presents techniques from the literature that address verification and validation of systems.

> Software verification and validation processes determine whether the development products of a given activity conform to the requirements of that activity and whether the software satisfies its intended use and user needs. [IEE05, §1, p.1]

Validation is the process of

> providing evidence that the software and its associated products satisfy system requirements allocated to software at the end of each life cycle activity, solve the right problem (*e.g.*, correctly model physical laws, implement business rules, use the proper system assumptions), and satisfy intended use and user needs. [IEE05, §3.1.35, p.9]

Verification is the process of

> providing objective evidence that the software and its associated products conform to requirements (*e.g.*, for correctness, completeness, consistency, accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance) ; satisfy standards, practices, and conventions during life cycle processes ; and successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities (*e.g.*, building the software correctly). [IEE05, §3.1.36, p.9]

This section lists the articles in the literature that deal with model composition to validate and/or verify software. We focus on the following activities as subtypes of the V&V activity :

Checking Consistency  ensures that a system design is complete and detects definition overlaps or conflicting situations otherwise. Model composition techniques propose methods to identify conflicts, resolve overlaps and ensure completeness.

Checking Correctness  ensures that a system produces data that is valid against the values expected by the same system or another system in interaction. Model composition techniques provide global representations that suit well with satisfying a set of constraints or rules for data correctness.

The following subsections give details about techniques found in the literature that address these two specific activities. Each description of an activity is completed by the description of a selected subset of techniques that support this activity.

1.3.3.1 Model Composition for Checking Consistency 

Checking the consistency of a system is ensuring that the data flow of a system remains consistent between two transactions. This is particularly true for databases systems : each time a data is changed, the integrity of the database must remains. In other words, the transaction should not introduce invalid data or else be canceled. Broadening the definition to general-purpose software, checking consistency implies to be consistent with a set of rules which are either provided by the system's architecture or design, or by the business domain which the system targets. The following sections illustrate consistency checking examples found in the literature.

Operator-based Correspondence

"Consistence Preserving Model Merge in Collaborative Development Processes" In [Bar08], Bartelt proposes mechanisms to synchronize models in a collaborative development environment. The author proposes a merge operator for multiple versions of the same model, this operator ensuring consistency of the merged model. By analogy with traditional document merging tools such as SVN or CVS, Bartelt proposes several atomic operators such as add or delete and a set of mappings such as changedItem, predecessors or successors for revision tasks. In addition to these operators, two functions are provided : (1) changeState that determines the last change operation on an information item and (2) mergeState that determines the information items of a document after processing all changes of a revision task and its predecessors. However the author specifically mentions that all these structures are not enough for building valid model merging tools so the author focuses instead on consistency checking. Consistency checking is dealt with in terms of checking constraints on models but no mechanism for automatically solving inconsistencies is proposed. It still rely on developers meetings for providing solutions.

Pattern-based Correspondence

"Managing Variability in Workflow with Feature Model Composition Operators" Acher *et al.* propose to compose parameterized services into workflows using SPL and AOM techniques in [ACL+10]. Their intent is (1) to capture commonalties and variabilities in parameterized services and (2) to provide support for tailoring and composing services. Using a set of composition operators, they insert a concern into the description of services and they merge models of connected services with one another. The set of operators are based on join points to express where concerns have to be woven in the description. The current approach deals with merging services with the same names. The composition operators offer consistency checking facilities such as verifying that sequential, concurrent or condition-based communication between services is conflict-free when reasoning on the global workflow.

Similar model composition approaches are proposed by Fleurey *et al.* [FBF+08], Noy and Musen [NM00], and Barais *et al.* [BKB+08].

Rule-based Correspondence

"Composing Models for Detecting Inconsistencies : A Requirements Engineering Perspective" In [PBB+09], Perrouin *et al.* propose a generic composition framework which extracts information from heterogeneous models and translate it into a set of fragments. The fragments are composed with one another to check for underspecification and inconsistencies. The fragment composition (or fusion in their terminology) process is driven by a set of fusion rules which are either equivalence rules (ER) or normalization rules (NR). ERs are operations that replace equivalent objects by a composed object and NRs transform fragments to detect inconsistencies. ER are atomic operations on models which computes the union of a source and a target object. NRs are constraints used to detect false positive inconsistencies.

Similar model composition approaches are proposed by Whittle and Jayaraman [WJ08], Jayaraman *et al.* [JWE+07], Perrouin *et al.* [PBB+09], and Brottier *et al.* [BLTN10].

Model-based Correspondence

"An Approach for Model Composition and Verification" Zhang *et al.* propose a model weaving framework [ZLL09] that defines a set of model composition operators that are related with source and target models. These models are transformed into Alloy and a LinkModel captures the composition relationships (LinkPoints) between model elements with precise semantics. The LinkModel is transformed into Alloy and drives the model composition engine. Running the Alloy Analyzer on top of this LinkModel, we obtain valid instances of the composed model and we can check that the composition definition is consistent.

Similar model composition approaches are proposed by Park and Ram [PR04], Zhang *et al.* [ZLL09], and Sabetzadeh and Easterbrook [SE06].

Delta-based Correspondence

The approach proposed by Cicchetti *et al.* [CDRP08] and presented in details in Section 1.3.2.1 also applies for checking model consistency.

1.3.3.2 Model Composition for Checking Correctness

Validating how a system or component behaves regarding the data it reads or produces requires the full picture of the system. Checking data for each component does not implies that the composition of all components produce data that is still correct. The following sections illustrate correctness checking examples found in the literature.

Pattern-based Correspondence

"Annotation and merging of SBML models with semanticSBML" In [KUL+10], Krause *et al.* propose a tool to check and edit MIRIAM annotations and SBO terms in SBML models. SBML (Systems Biology MArkup Language) is the leading exchange format for mathematical models in Systems Biology. Semantic annotations link model elements with external knowledge via unique database identifiers and ontology terms. The proposed model merging process joined duplicate elements of two models. Matching elements relies on a comparison of their MIRIAM annotations to resolve name differences. The composition process is driven by the modeler interpretation of the models and the semanticSBML tool help modelers to detect syntactic and semantic conflicts. The main purpose of using model merging on SBML models is to build a set of validity criteria for SBML models.

A similar model composition approach is proposed by Jackson *et al.* [JKB+06].

Constraint-based Correspondence

"Compositional Verification for component–based systems and applications" Bensalem *et al.* propose an algebraic model [BBN+10] that states how a system composed of atomic components may satisfy a given global invariant. They compute interaction invariants for each component of the system and they compute component invariants to obtain the set of reachable states of the system. This iterative method produces invariants that are progressively stronger and thus satisfying these invariants should lead to better verification of the system. They demonstrate their approach on checking for deadlock situations not considering synchronization-based deadlocks and four other use-cases.

Model-based Correspondence

"View Merging in the presence of incompleteness and inconsistency" In [SE06], Sabetzadeh and Easterbrook propose an approach for merging overlapping views of a single system. Views are used by many people with different skills and their own understanding of the system. It leads to "have discrepancies over the terminology being used" or on the semantics or the structure of the elements being used. The authors propose a framework for merging these incomplete or inconsistent views based on a graph-based formalism. Domain experts annotate these graphs with a degree of knowledge to express their confidence in the design. This degree of knowledge participates in the detection of inconsistencies or incomplete specifications. The process of merging views is based on a disjoint union of the graphs nodes and edges and knowledge about how to resolve inconsistency and incompleteness is captured in both connectors (fragments of graphs that contains overlapping elements) and correspondences between the elements from the connector and the elements from the views. The correspondences are interpreted as equivalences between graph elements or adding/hiding specific graph elements. All correspondences have to provided explicitly since the merging process does not embed any matching mechanism.

1.3.4 Model Composition for the Evolution and the Maintenance of Systems

This section presents techniques from the literature that deal with the evolution of software. Evolution in Computer Science represents a large part of the software life-cycle to cope with environment changes, end-users expectations evolution, technological advances and bug correction. We consider the following activities as subtypes of software evolution and maintenance :

Dynamic Reconfiguration  Adaptation of a system once it has been deployed for being used by end-users is an active field of research. Due to the increasing pace of evolution in software, software have to be adapted constantly and this activity may require relative long time of shutdown. In contexts where service unavailability is not possible or not affordable, systems have to be adapted while running. Compared to static adaptation, dynamic (or runtime) adaptation requires specific mechanisms to both keep the system running and to provide synchronous or asynchronous ways to modify a system. Examples of dynamic adaptation are "fast-switch" of contexts or degradation and requires also specific validation techniques before deployment to avoid loss of service.

Refactoring  The refactoring activity is the application of "a set of ... restructuring operations (refactorings)" [Opd92, p.2] on an existing software design or implementation. William Opdyke give the following definition to refactorings as :

> Refactorings are reorganization plans that support change at an intermediate level. ... While refactorings do not change the behavior of a program, they support software design and evolution by restructuring

> a program in a way that allows other changes to be made more easily. [Opd92, §1.2, p.10–11]

Goals of refactoring are numerous and to give some examples, we may think about increasing maintainability using design patterns or SoC, improve readability by adding comments, or removing dead code using analysis techniques.

Adaptation Adapting a software has several goals. While it may refer to refactoring, the meaning we choose here is adapting a system to be able to communicate with another piece of software. Similarly to integration at design-time, developers have to provide adapters, wrappers, proxies, facades, etc. to achieve systems interaction and inter-communication.

Synchronization Synchronization is another activity involving several independent systems. Contrary to adaptation which refer to build a new software from existing ones, synchronization is focused on ensuring consistency between systems while considering them as different entities.

Reconciliation Reconciliation is an activity that focuses on synchronizing two representations of the same system. These representations have a common ancestor from which they evolved independently. This is a very common use case in versioning systems.

1.3.4.1 Dynamic Reconfiguration

In the literature, we found a single approach that deals with dynamic reconfiguration of systems. This approach proposed by Morin *et al.* is detailled in the following section.

Pattern-based Correspondence

"Taming Dynamically Adaptive Systems using Models and Aspects" In [MKB+08], Morin *et al.* propose an approach to tackle the challenge of reconfiguring Dynamic Adaptive Systems at runtime. These systems rely on modes that capture the current state and configuration of the system. They use Aspect-oriented Modeling techniques and generative approaches to tackle the combinatorial explosion in the computation of modes for these systems. They propose a five-step process as follows : (1) gather data about the current configuration and build an abstract representation of the running system, (2) compute a configuration on-demand with aspect weaving techniques, (3) generate reconfiguration scripts, (4) validate the new configuration and (5) adapt the running system with the new configuration. The construction of a configuration is supported by SmartAdapters [LMV+07], an aspect-weaving tool that uses join points, pointcuts and advices to specify what, where and how aspects are woven with one another. Additional correspondences are implicitly created in step 4 in which the comparison of the new configuration with the original one produces a match model and a diff model. The match model captures which model elements are equivalent

whereas the diff model includes "addition/removal of components/bindings, changes in attributes values, etc.".

1.3.4.2 Refactoring

The following sections illustrate examples of approaches found in the literature regarding refactoring existing systems.

Operator-based Correspondence

The approach proposed by Berstein [Ber03] and presented in Section 1.3.2.4 also applies for system refactoring.

Pattern-based Correspondence

"Annotation Refactoring : Inferring Upgrade Transformations for Legacy Applications" Tansey and Tilevich proposed an approach for performing legacy applications refactoring [TT08]. The paper is intended to solve the problem of upgrading text- and name-based frameworks into metadata-based frameworks, specifically with annotations. Using metadata from these annotations, they are able to infer general transformation rules to perform the refactoring. From a set of representative examples, users guide the inference engine by defining upgrade patterns and parameterize the transformation engine using provided generated rules. Representative examples are classes or interfaces that uses framework features that differ between versions or vendors. Differences may occur in types, method signatures, fields, annotations, imports or statements. Upgrade patterns capture the refactoring for many upgrade scenarios. Patterns are defined within what we may call scopes such as : (1) bottom-up when changes in the code depend of the enclosing class, (2) top-down when a specific class migration imposes a special structure, or (3) identity when correspondences are found between elements of the same type. From the analysis of these representative examples, authors build transformation rules from a specific DSL that eases further changes by end-users. To help the definition of transformation rules, representative examples are analyzed by an inference algorithm that generates a set of rules for a given refactoring.

A similar model composition approach is proposed by Belapurkar [Bel04].

Rule-based Correspondence

"Refactoring Support for Class Library Migration" In [BTF05], Balaban *et al.* propose a technique for automatic migration of applications that use legacy library classes. Based on mappings between legacy classes and their replacement classes, they use type-based constraints to determine where it is possible to migrate code without effecting the program's type-correctness or behavior. Type-based constraints are a formalism for expressing relationships between the types of declarations and expressions

that are used for type-checking, type inference or refactoring. These constraints are generated from the Abstract Syntax Tree (AST) of a program. Solving these constraints is then achieved by a specific algorithm. A solution encompasses the maximum number of legacy classes that can be transformed to their replacement classes.

Similar model composition approaches are proposed by Bernstein [Ber03] and Fleurey *et al.* [FBB+07].

Model-based Correspondence

"Metamodel Adaptation and Model Co-Adaptation"
Different from previous approaches on a more "classic" definition of refactoring, the work of Wachsmuth [Wac07] on the adaptation of meta–models and models is another illustration of refactoring in a more general meaning. We intentionally classified this paper in the Refactoring Section because of the definition that we give of Refactoring versus Adaptation. In this paper, the author propose to propagate the changes of a meta–model to the models that conform to this meta–model. This allows keeping consistency and validity of existing models regarding changes. Wachsmuth proposes eight types of relationships between meta–models to capture changes and evolutions. Relationships are used to map meta–models together, considering equivalence, submodel or enrichment, variation or extension, instance or semantic preservation, etc. Based on this theoretical basis, they achieve meta–model evolution through the development of a set of Query/View/Transformation (QVT) relations. Once the meta–models have been related with each other, specific transformations are available for refactoring models. Refactoring may or may not be necessary, depending the type of relations. For instance, adding new instances should have no impact on old instances whereas modifying relationships cardinality (and by the time, restricting relations) may need refactoring for some instances that do not meet the new requirements.

Similar model composition approaches are proposed by Bernstein *et al.* [BHP00], Fleurey *et al.* [FBB+07], and Zhang and Chen [ZC07].

1.3.4.3 Adaptation

The following sections illustrate examples of approaches found in the literature regarding adaptation of existing systems.

Operator-based Correspondence
The approach proposed by Berstein [Ber03] and presented in Section 1.3.2.4 also applies for system adaptation.

Pattern-based Correspondence
The approach proposed by Tansey and Tilevich [TT08] and presented in Section 1.3.4.2 also applies for system adaptation.

Rule-based Correspondence
The approach proposed by Berstein [Ber03] and presented in Section 1.3.2.4 also applies for system adaptation using rule-based correspondences.

Constraint-based Correspondence

"Solving the modeling problems of object-oriented languages by composing multiple aspects using composition filters" Aksit and Tekinerdogan proposed Composition–Filters [AT98] to validate or invalidate a message that is received by an object in an object-oriented based application. Filters are conditions or constraints with a special "action" part which is either reject a message or dispatch a message. I selected this approach as the illustration of an alternative way to weave aspects in an aspect-oriented approach for objects that were not designed to be reusable in the first hand. Every filter is an extension of an existing class and is defined using the class elements such as the operations. Allowing or rejecting messages are one way to adapt existing elements and provide adaptation mechanisms without modifying existing objects specifications.

Model-based Correspondence

"Towards Re-Engineering Legacy Systems for Assured Dynamic Adaptation" In [ZC07], Zhang and Cheng propose a model-driven approach for adapting legacy systems that were not meant to be adapted. Adaptation must also ensure that the systems still fulfill their intended purpose and that the systems correctness is preserved. Using UML models to bridge the gap between software verification and software implementation, they propose Petri nets based representations to produce an adaptation model. Based on previous work about formalizing a subset of UML diagrams using mappings between meta–models, they integrate a reverse-engineering step to tackle legacy and newly developed code adaptation. They propose thus a four-step process that (1) selects a set of legacy programs to be adapted from requirements analysis, (2) translates these programs into UML Statecharts diagrams for verification, (3) helps developers to create an adaptation model for verification, and (4) integrates and translates adaptation models into executable programs. This technique is invasive which means that legacy programs have to be modified to use the adaptive programs that have been generated along the adaptation process.

A similar model composition approach is proposed by Bernstein *et al.* [BHP00].

1.3.4.4 Synchronization

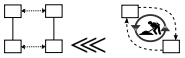
The following sections illustrate examples of approaches found in the literature regarding synchronization of existing systems.

Operator-based Correspondence

"Towards Automatic Model Synchronization from Model Transformations" Based on existing model transformations, Xiong *et al.* propose an approach for synchronizing the models involved in the model transformations [XLH+07]. Given a specific transformation, they have implemented a set of ATL rules to build a synchronization system. Since ATL rules only provide a one-way transformation, they calculate the backward transformation by extending the ATL Virtual Machine with push-back functions. The synchronization step relies on two common operators in model-driven engineering. The first one deals with identifying differences between two versions of the same model. The second one called Merging achieves the merging of two models by unifying the sets of their model elements. To keep track of differences and to guide the merging process, differencing two models produces metadata that are concretely bound to the original model using annotations. These annotations indicate if a model element has not been modified, or if it has been replaced, or if it has been inserted, or if it has been deleted. Using these tags, they are able to produce different models that includes modifications from the target model or the source model and that we should propagate. Propagation if performed by the merging operator and according to a set of rules provided by the authors for conflicts resolution, both the target model and the source model are updated.

Rule-based Correspondence

The approach proposed by Xiong *et al.* [XLH+07] and presented in Section 1.3.4.4 is an hybrid approach that uses both operator–based and rule–based correspondences.



Constraint-based Correspondence

"Tracing Evolution Changes of Software Artifacts through Model Synchronization" In [IK04], Ivkovic and Kontogiannis use model transformations to achieve model synchronization. Their approach rely on a set of rules and relationships to capture dependencies or equivalence between models and to evaluate if these models are synchronized or not. They propose a new metamodel for the definition of models with synchronization capabilities. This generic metamodel is called Graph Metamodel for Synchronization. Based on the information captured in this language and a set of mappings, they propose another language to synchronize existing models. Models that conform to the transformation language contain the concrete algorithm for propagating changes from one model to the other and vice-versa.

Model-based Correspondence

"From Model Transformation to Incremental Bidirectional Model Synchronization" [GW09] is an extension of [GW06] where Giese and Warner use Triple Graph

Grammar (TGG) to synchronize models. In TGG, transformation designers manipulates three models that correspond to the source model, the target model and an additional correspondence model that defines mappings using predefined classes and associations. Information gathered in the correspondence model is very similar to the information that we may capture to build transformation traces. Using TGG, designers also manipulate "stereotypes" on classes and relations between the source model, the correspondence model and the target model. These "stereotypes" carry information about the possible creation or deletion of an element during the model transformation process. Based on the correspondence model, Giese and Warner propose an algorithm to navigate through this model and synchronize the two models.

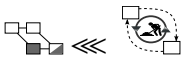
A similar model composition approach is proposed by Wachsmuth [Wac07].

Delta-based Correspondence

"Automating Co-Evolution in Model-Driven Engineering" This work of Cicchetti *et al.* [CRE+08] propose an approach for vertical co-evolution, *i.e.*, synchronize a metamodel and its conforming models in the case of modifications. Based on a model-based representation of differences between models, they automatically generate a transformation that performs the necessary synchronization steps. The difference metamodel that captures the differences between models is automatically generated from the source and target models. They end up with a new language that is dedicated to represent differences for a given language. Differences are characterized within three categories : additions, deletions or changes. Once the difference model is computed, two transformations are generated, one from the source to the target and one from the target to the source. These two transformations may then be used to synchronize source and target models with each other.

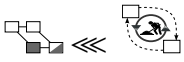
1.3.4.5 Reconciliation

The following sections illustrate reconciliation with examples from the literature.



Operator-based Correspondence

The approach proposed by Bartelt. [Bar08] and presented in Section 1.3.4.4 also applies for system reconciliation.



Rule-based Correspondence

"Using a Model Merging Language for Reconciling Model Versions" Engel *et al.* propose an approach for reconciling different versions of a model and to provide automatic versioning [EPK06]. Their tool uses a differencing and merging process with possible interaction with developers, to identify how to compose versions of a model. Using Epsilon Merging Language (EML) [KPP06], they define matching and merging rules for reconciling versions of models. Matching rules compares model

elements from the input models to identify equivalent model elements. The merging process is realized by merging rules which state how to elements have to be composed with each other and how to compute the merged element. In the specific example of models reconciliation, they use matching rules to detect which elements have been added or deleted between two versions of a model. Detecting changes is also part of the matching process however no step in the merging process refers to it so this interpretation is discarded. Once model elements are marked to be added or deleted, the merging model follows a straight-forward algorithm for deleting/adding model elements from/to the resulting model. Attributes and relationships between model elements are handled in a fourth and fifth steps.

Model-based Correspondence
The approaches proposed by Wachsmuth [Wac07] and Bernstein *et al.* [BHP00] and presented respectively in Section 1.3.4.2 and in Model–based correspondence in Section 1.3.2.1 also applies for system reconciliation.

Delta-based Correspondence
The approach proposed by Engel *et al.* [EPK06] and presented in Section 1.3.4.5 also uses delta-based correspondences. The approach proposed by Cicchetti *et al.* [CDRP08] and presented in Section 1.3.2.1 also applies for system reconciliation.

1.3.5 Systematic Literature Review Summary

The systematic literature review provides examples that supports the categories of correspondence and interpretation that we presented in Section 1.2. This section presents results from the systematic literature review to summarize the distribution of articles regarding *(i)*the kind of correspondence that they use (see Section 1.3.5.1), *(ii)*the kind of correspondence that they use against the interpretation of these correspondences (see Section 1.3.5.2, and *(iii)*the kind of correspondence that they use against the software development activity (see Section 1.3.5.3). We provide additional discussion and validation with regard to the research objectives in Section 1.4.

Raw results from the systematic literature are presented in Section 1.3.5.1, 1.3.5.2, and 1.3.5.3. Section 1.4 discusses the research objectives proposed in Section 1.3.1.1.

1.3.5.1 Kind of Correspondences and Distribution of Articles

From the total number of 88 articles, 19 use operator-based correspondences, 32 use patterns to define correspondences, 15 use rule-based correspondences, 4 use constraint-based correspondences, 24 use models to represent correspondences and 3 use delta-based correspondences. The sum of the distributed articles is greater than the total number of articles since some approaches are hybrid (see Figure 1.6) such as operators with patterns, operators with models, patterns with models.

Results corroborate the **`H1a.1`** hypothesis from Section 1.3.1.1 : every model composition technique has successfully been assigned a kind of correspondence from the

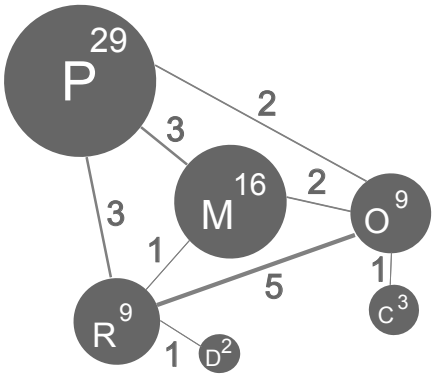


FIGURE 1.6 – Distribution of articles with respect to the type of correspondences. Circles represents the number of articles for each kind of correspondence and lines represents the number of hybrid approaches

category. Let RA_i the number of articles that are assigned to only one kind of correspondence i and HA_i the number of articles that are assigned to at least two kinds of correspondence. We compute the precision of the n kinds of correspondence such that :

$$Precision = \frac{\sum_{i=1}^{n} \frac{RA_i}{RA_i + HA_i}}{n} \tag{1.1}$$

Values of precision for each kind of correspondence and the global precision of the proposed categories is shown in Table 1.2 with TA (RA_i+HA_i) the total number of articles for one kind of correspondence i.

	Operator	Pattern	Rule	Constraint	Model	Delta	Total
RA	9	29	9	3	16	2	
HA	10	8	10	1	6	1	
TA	19	37	19	4	22	3	
Precision	0.47	0.78	0.47	0.75	0.73	0.67	0.65

TABLE 1.2 – Precision for each kind of correspondence and global precision for the category

As an additional criteria, we compute the ratio of hybrid approaches HA against the number SA of approaches that use a single kind of correspondence such that :

$$Ratio = \frac{HA}{SA} = 0.27 \tag{1.2}$$

1.3.5.2 Interpretation and Distribution of Articles

Table 1.4 presents results with regard to the class of overlapping interpretations of correspondences. Since the interpretations of correspondences are rarely provided

by the authors explicitly, the classification reflects the global feeling of how correspondences may be interpreted in a given context for a specific problem. Moreover, approaches usually supports several interpretations with regard to a given context. However, among a total number of 78 articles that deal with overlapping models, a large number of model composition techniques propose equivalence and similarity as the main interpretations. The results agree with the idea that usually, overlapping models contains model elements which are semantically equivalent or very close to one another, these interpretation have respectively 51 and 23 candidates. The Ad Hoc interpretation has 28 candidates since this interpretation includes all articles which interpretations do not fit in the proposed classification. Other interpretations have far less candidates in the set of articles since they correspond to more specific processing of the correspondences for a given problem : 11 occurrences of the add interpretation, 11 occurrences of the delete interpretation, 5 occurrences of the change interpretation, 7 occurrences of the generalization interpretation, 5 occurrences of the aggregation interpretation, 7 occurrences of the overriding interpretation, and 1 occurrence of the information gap interpretation.

Table 1.5 shows results with regard to the classes of cross-cutting and interaction interpretations of the correspondences. The distribution of articles in these interpretation is almost even except for the co-dependency interpretation which only collect 4 occurrences in the set of articles. Co-dependency is specific to the representation of processes and few papers dealing with model composition in this domain go beyond the sequentiality and parallelism of activities. The distribution for each interpretation is as follows : 11 occurrences are about replacing model elements, 18 occurrences are about augmenting, 8 occurrences are about removing, 15 occurrences are about sequencing, and 12 occurrences are about paralleling.

Results corroborate the **`H1b.1`** hypothesis from Section 1.3.1.1 : every model composition technique has successfully been assigned a kind of correspondence from the category. However articles within the Ad Hoc category have either an undefined interpretation or a too large interpretation to fit in the categories. Thus, we compare the number of articles that are only filed in the Ad Hoc kind of interpretation against the total number of articles to strengthen the relevance of the proposed categories of interpretations. Let AI be the number of articles that file the Ad Hoc category only and TI the total number of articles. We compute the rate of articles P that do not use the Ad Hoc interpretation such that :

$$P = \frac{TI-AI}{TI} = \frac{88-7}{88} = \frac{81}{88} = 0.92 \tag{1.3}$$

Let AI' the number of articles that file the Ad Hoc category in addition to other categories, we the rate of articles P' that use the Ad Hoc interpretation among others such that :

$$P' = \frac{TI-AI}{TI} = \frac{88-21}{88} = \frac{67}{88} = 0.76 \tag{1.4}$$

Most of the model composition approaches propose more than one interpretation for correspondences. We explain this result by the fact that some interpretations are related with each other in a specific context. For instance, a model composition technique

that handles cross-cutting models and provides an "augment" interpretation without proposing "remove" or "replace" would look incomplete. Computing the number of articles with only one interpretation against the total number of articles using at least this interpretation is not a relevant criterion in this situation. We propose to calculate the **accuracy** of the categories of interpretation using the following scale :

- Classification of an approach in exactly one kind of interpretation is a very relevant classification and is valued 1.
- Classification of an approach in exactly two kinds of interpretations is a somewhat relevant classification and is valued 0.8.
- Classification of an approach in exactly three kinds of interpretations is a somewhat irrelevant classification and is valued 0.3.
- Classification of an approach in at least four kinds of interpretations is an irrelevant classification and is valued 0.

Using this scale, we compute a value of accuracy for the categories of interpretations. Results are shown in Table 1.3 with NI the number of interpretations, R the scale of relevance, NA the number of articles for a given number of interpretations, $TA = 83$ the total number of articles and $Accuracy = \frac{\sum(R*NA)}{TA}$ the value of accuracy for this category.

NI	R	NA	NA/TA	Accuracy
1	1	23	0.28	
2	0.8	28	0.34	
3	0.5	17	0.21	
≥ 4	0	15	0.18	
				0.65

TABLE 1.3 – Accuracy of the categories of interpretations

Correspondence \ Interpretation			Overlapping						
	Add +	Delete −	Equ ≜	Sim ≈	Gen →	Aggr ◆	Overriding :=	Info Gap ⇜⇝	Ad Hoc (...)
Operators			[Ber03]	[Ber03]					[Ber03]
		[DRMM+10]	[DRMM+10]	[DRMM+10]		[DRMM+10]			
			[GS03]						
			[BCR05]	[BCR05]					
			[BCR+07]	[BCR+07]					
			[KM10]						
			[MPL+09]						
			[LNK+01]		[LNK+01]				
			[MBN+09]						
			[PRB+09]		[PRB+09]	[PRB+09]			[PRB+09]
			[HHJ+08]						[HHJ+08]
			[HKG+10]						
	[XLH+07]	[XLH+07]							[XLH+07]
									[PGP+07]
				[CSN08]					
	[Bar08]	[Bar08]							
									[PVSG+08]
	[ASM+10]	[ASM+10]							[ASM+10]
Patterns			[BKB+08]			[BKB+08]			
			[FBF+08]						
	[Let07]	[Let07]	[Let07]				[Let07]		
			[NM00]						
	[ACL+09]		[ACL+09]						
			[Cla02]				[Cla02]		
			[ES06]		[ES06]	[ES06]	[ES06]	[ES06]	
			[OO07]						
			[FFR+07]						

Correspondence \ Interpretation			Overlapping						
	Add +	**Delete** −	**Equ** ≜	**Sim** ≈	**Gen** ⇾	**Aggr** ◆	**Overriding** :=	**Info Gap** ⇜⇝	**Ad Hoc** (...)
	[MPL+09]		[KUL+10] [BWH10] [MPL+09] [SFS+08] [Jez08] [TT08] [WS08] [AJT+09] [ACL+10] [JZF+09] [NSC+07] [CSN08]	[KUL+10] [AJT+09]			[MPL+09] [SFS+08] [Wag08]		[BWH10] [TT08] [FBB+07] [MBJ08]
Rules	[EPK06] [XLH+07]	[EPK06] [XLH+07]	[EPK06] [BSM+07] [RCE08] [ACL+09] [KPP06] [AEC+07] [BTF05] [BLTN10] [PBB+09] [HKG+10]	[EPK06] [BSM+07] [CRR+07]					[EPK06] [BSM+07] [RCE08] [BTF05] [BLTN10] [XLH+07]

Correspondence \ Interpretation			Overlapping						
	Add +	Delete −	Equ ≜	Sim ≈	Gen ⇾	Aggr ◆	Overriding :=	Info Gap ⇜⇝	Ad Hoc (...)
									[WJ08] [JWE+07]
Constraints			[IK04]						[IK04] [PVSG+08]
Models	[Wac07] [SE06]	[Wac07] [SE06]	[ZLL09] [GW06] [GW09] [FDV07] [CBJ10] [MMP+10] [PR04] [DRMM+10] [AEC+07] [DFB+05b] [Wac07] [SE06]	[GW06] [GW09] [FDV07] [CBJ10] [MMP+10] [PR04] [AEC+07] [DFB+05b] [Wac07] [ZC07]	[FDV07] [MMP+10] [DFB+05b] [Wac07]	[BA00]	[ZLL09]		[FDV07] [CBJ10] [MMP+10] [AEC+07] [DFB+05b] [Wac07] [BHP00] [FBB+07] [LP03] [PDCS+01] [MBJ08]
Deltas	[EPK06] [CRE+08]	[EPK06] [CRE+08]							

TABLE 1.4 – Distribution of articles with regard to the types of correspondence and the overlapping interpretations. The Add and Delete interpretations are included in this table for convenience purposes.

Correspondence \ Interpretation	Cross-Cutting			Interaction		
	Replace :=	Augment +=	Remove \	Sequence ;	Parallel ||	Co-Dependency
Operators		[DRMM+10] [GKR+08]		[GS03] [PGP+07] [SY10] [PVSG+08]	[GS03] [PGP+07] [BBN+10]	[SY10] [Bar08]
Patterns	[JKB+06] [BKB+08] [MKB+08] [MBN+09] [Cla02] [KAAK09] [SFS+08] [Jez08] [Bel04] [MBJ+07]	[JKB+06] [BKB+08] [MKB+08] [MBN+09] [KAAK09] [Jez08] [Bel04] [MBJ+07] [KHJ06; KJP05] [PBC+11] [HKG+10] [MBJ08]	[JKB+06] [BKB+08] [MKB+08] [MBN+09] [Bel04] [PBC+11] [HKG+10]	[BKB+08] [KHJ06; KJP05] [MBFF10] [Mos10]	[HKG+10] [MBFF10] [Mos10]	[MBFF10]
Rules	[OMK09]	[OMK09]	[OMK09]			

Continued on next page

Correspondence \ Interpretation	Cross-Cutting			Interaction		
	Replace	Augment	Remove	Sequence	Parallel	Co-Dependency
	:=	+=	\	;	||	
		[CRR+07]		[CRR+07]	[CRR+07]	[CRR+07]
		[HKG+10]	[HKG+10]		[HKG+10]	
				[CDK+07]	[CDK+07]	
				[PGP+07]	[PGP+07]	
				[SY10]		[SY10]
Constraints		[AT98]				
				[PVSG+08]		
Models		[DRMM+10]				
		[ZLL09]				
		[PBC+11]	[PBC+11]			
		[MBJ08]				
				[BA00]	[BA00]	
				[NB04]	[NB04]	
				[GJ05]	[GJ05]	
Deltas				[CDRP08]	[CDRP08]	

TABLE 1.5 – Distribution of articles with regard to the type of correspondence and the cross-cutting and interaction interpretations

1.3.5.3 Software Activities and Distribution of Articles

Table 1.6 presents results with regard to the intention of designing software. As we expected, 47 of the articles refer to the activity of model composition as their first intention. However the use of model composition for other activities is significant, respectively 9 articles about derivation, 12 about orchestration and 12 about integration. Table 1.7 presents results with regard to the intention of verifying and validating software. Among 17 articles related to verification and validation, 13 articles propose approaches for checking software consistency and 4 are about checking software correctness. Table 1.8 shows results with regard to the intention of making existing software evolve. On a total of 24 articles, 1 propose dynamic reconfiguration facilities, 7 are about refactoring, 4 deals with software adaptation, 7 are about software synchronization and 5 propose approaches for software reconciliation.

Results corroborate the **H2.0** hypothesis from Section 1.3.1.1 : a large range of software life-cycle activities are actually supported by model composition techniques to some extent. The model composition activity plays a significant role in these software life-cycle activities although not always being emphasized.

Intention / Correspondence	Design			
	Composition	**Derivation**	**Orchestration**	**Integration**
Operators	[PGP+07],[Ber03] [KM10],[LNK+01] [PRB+09] [HHJ+08],[GKR+08] [CSN08],[HKG+10] [DRMM+10] [BCR05 ; BCR+07]	[ASM+10] [MPL+09]	[PGP+07] [FGF+08] [PVSG+08] [HKG+10]	[Ber03],[SY10] [GS03] [BCR05 ; BCR+07]
Patterns	[Wag08],[Let07] [NSC+07],[MBFF10] [Cla02],[ES06] [NM00],[OO07] [FFR+07],[KUL+10] [BWH10],[CSN08] [KAAK09],[SFS+08] [PBC+11],[ACL+10] [Jez08],[MBJ+07] [WS08],[AJT+09] [KHJ06 ; KJP05]	[MPL+09] [MBJ08] [JZF+09] [Jez08] [AJT+09] [PBC+11] [MKB+08]	[Mos10]	
Rules	[CRR+07],[RCE08] [ACL+09],[KPP06] [AEC+07],[WJ08] [JWE+07],[BLTN10] [HKG+10],[OMK09]	[JWE+07]	[CRR+07] [HKG+10] [CDK+07]	[BSM+07]
Constraints	[AT98]		[PVSG+08]	
Models	[BHP00],[DFB+05b] [DRMM+10],[AEC+07] [PBC+11],[SE06] [Wac07] [LP03 ; PDCS+01]	[PBC+11] [MBJ08]	[BHP00],[NB04] [GJ05],[FGF+08]	[FDV07],[CBJ10] [MMP+10],[PR04] [LP03 ; PDCS+01]
Deltas	[CDRP08]			

TABLE 1.6 – Distribution of articles with respect to the type of correspondences and the software design intention

Correspondence \ Intention	Verification	
	Consistency	Correctness
Operators	[Bar08]	[BA00]
Patterns	[FBF+08],[NM00],[ACL+10],[BKB+08]	[KUL+10],[JKB+06]
Rules	[WJ08],[JWE+07],[PBB+09],[BLTN10]	
Constraints		[BBN+10]
Models	[PR04],[ZLL09],[SE06]	[SE06]
Deltas	[CDRP08]	

TABLE 1.7 – Distribution of articles with respect to the type of correspondences and the software verification and validation intentions

Correspondence \ Intention	Evolution				
	Reconfiguration	Refactoring	Adaptation	Synchronization	Reconciliation
Operators	[MBN+09]	[Ber03]		[Ber03],[XLH+07]	[Bar08]
Patterns	[MBN+09]	[FBB+07] [TT08]	[TT08]		
Rules		[BTF05]		[XLH+07]	[EPK06]
Constraints				[IK04]	
Models		[BHP00] [ZC07] [Wac07]	[BHP00], [ZC07] [FBB+07]	[GW06 ; GW09] [Wac07]	[BHP00] [Wac07]
Deltas				[CRE+08]	[EPK06] [CDRP08]

TABLE 1.8 – Distribution of articles with respect to the type of correspondences and the software evolution intention

1.4 Discussion

With respect to the possible review biases that we discussed in Section 1.3.1.6 and in Section 1.3.1.7, this section proposes to go further on interpreting the systematic literature review results regarding the research objectives (see Section 1.3.1.1).

1.4.1 Are Correspondences and Interpretations Pervasive ?

The systematic literature review validates the precision and relevance of the classification of the key concepts of model composition techniques. Observations back up the intuitive classification of the kinds of correspondence and of the kinds of interpretations and increase the confidence that we have in the proposed classifications as a basis for characterizing and comparing model composition techniques. With a global precision of 65% (see Table 1.2) for the classification of the kinds of correspondence and a precision of 92% (see Equation 1.3) for the classification of the kinds of interpretation, we consider the hypotheses **`H1a.1`**, **`H1b.1`** and **`H1.3`** as valid.

Validity of the **`H1.3`** hypothesis implies that the **`H1.0`** hypothesis is also valid. Since the list of correspondences and interpretations is precise enough to cover all model composition techniques, we deduce that every model composition technique uses a set of correspondence relationships and a specific interpretation to these correspondences. The proposed classification and decomposition into a set of correspondences and interpretations to these correspondences (Question **`Q1`**) is thus relevant to characterize model composition approaches.

We must however moderate the validity of the **`H1.3`** hypothesis since *(i)*26% (see Equation 1.2) of the studied model composition approaches use hybrid kinds of correspondences, *(ii)*76% (see Equation 1.4) of the model composition approaches use an Ad Hoc interpretation in conjunction with other interpretations, and *(iii)*the global accuracy of the categories of interpretations is about 65% (see Table 1.3). These figures pinpoint the difficulty to extract valuable information due to the lack of formalization and precise definition of the model composition techniques. Model composition techniques also often propose several interpretations to achieve the same goal in a given context, these interpretations being logically related with one another. Ultimately, the wording of model composition approaches may lead to some misinterpretation of their description and definition.

Results from the systematic literature review do not satisfy the **`H1.4`** hypothesis. The distribution of model composition techniques among each category (correspondences and interpretations) is far from even. We details our explanation as follows :

Correspondences

- Pattern is the most represented kind of correspondences in the studied articles. A pattern is a general word that covers a lot of techniques to compare elements with one another and this probably explains why almost half of the hybrid techniques uses patterns in conjunction with another kind of correspondence.
- The choice of a kind of correspondence may probably results from cultural background or designers preference in addition to a trade-off between ex-

pressiveness, degree of automation and further considerations related to the context and purpose of a specific model composition technique.

Interpretations

- Equivalence, similarity are the most represented kinds of interpretation in the studied articles. Cultural background has probably an impact on this result since the definition of model composition is often narrowed to merging model with close structures at design–time.
- Ad Hoc interpretations are also well represented in the studied articles. The lack of formalization or classification for interpretation probably prevents slicing very expressive operators that perform complex computations into more manageable and reusable modules of computation.

1.4.2 Is Model Composition a Common Operation in Software Development ?

The validity of hypothesis **H2.0** has been briefly discussed in Section 1.3.5.3 and answers positively to Question **Q2** from the research objectives. In the context of this thesis we will not further analyze the relationship between correspondences and software activities nor propose any further explanation on the distribution of correspondences and interpretations among software activities. We use software activities mainly for presentation purpose only and our intent was to rely on examples to confirm that model composition tackles far more software activities than merging structural models at design-time.

1.4.3 Summary of the Contribution

The validation of the two questions **Q1** and **Q2** from the research objectives is the first step in the detection of commonalties among various operations on models and among existing model composition techniques. We demonstrate that categories proposed in Section 1.2 are valid with regard to the current state of practice in model composition. Based on these categories, the first contribution of this thesis is the **definition of an interpretive lens to analyze and compare existing model composition techniques**.

The main contribution of this thesis is to propose **novel definition of model composition as a pair of a mapping and a set of interpretations**. This novel definition allows capitalizing the commonalties of the various model composition techniques. Capitalization motivates the second contribution of this thesis that is the **definition of a unified theoretical framework**. The definition of a unified theoretical framework should *(i)*allow the redefinition of the various model composition techniques and of the model composition purposes ; and *(ii)*allow the definition of specific model composition frameworks for a given purpose.

In Section 1.4.4, we present an overview of existing generic model composition frameworks (GCF). We propose a set of objectives that we consider as important to achieve capitalization and we stress the specific challenges that this thesis address.

1.4.4 Overview of Existing Generic Composition Frameworks

We consider that the following objectives are important towards capitalizing commonalties across various model composition approaches that adress various model composition goals :

- We should propose semantics for mappings relationships as a set of predefined interpretations. A list of predefined interpretations should allow designers to reuse them to address specific model composition purposes.
- We should keep the coupling between a mapping and its interpretation as low as possible. Low coupling allows both *(i)*reusing a mapping with different interpretations to tackle various purposes ; and *(ii)*reusing a given set of interpretations that tackle a specific purpose with different mappings.
- We should support various model composition operations to cover the whole range of activities that are supported by model composition approaches.
- We should support the customization of the model composition process to decrease the effort in building specific model composition frameworks.

Sections 1.4.4.1 to 1.4.4.3 discuss existing GCFs with regard to the contributions of this thesis. Section 1.4.4.4 summarizes how existing GCFs support the proposed objectives and emphasizes the challenges that the main contribution addresses.

1.4.4.1 Relationship–based Approach

Chechik *et al.* propose a relationship–based approach to ease model integration [CNM11]. They state that "...relationships [which hold between a set of models] should be treated as first–class artifacts..." [CNM11, §6, p.14] to reduce the global complexity of the definition of model composition frameworks. The categories of interpretations from Section 1.2.2 are inspired from this work : we keep the list of overlapping relationships with the same semantics and detail the types of relationships for the model interaction and model cross-cutting categories. Extending the types of relationships available for model composition and validating the categories by conducting a systematic literature review, we propose a formalization for each type of relationship and a process for building model composition frameworks.

Chechik *et al.* claim that future research should "...develop a more thorough classification of the purposes for which merge is applied applied, and subsequently study the applicability and tradeoffs between different merge operators for a given purpose." [CNM11, §6, p.14]. Proposing a novel definition of model composition as a pair of a mapping and an interpretation allows exploring how the purpose of a specific model composition operation influences the meaning of relationships which hold between a set of models. While providing an extensive list of purposes for model composition is out of the scope of this thesis, results from the systematic literature review may provide a starting point for future research in this direction.

1.4.4.2 ATLAS Model Weaver and Virtual EMF

The novel definition of model composition as a pair of a mapping and an interpretation is close to the experiments that Didonet *et al.* and Fabro *et al.* have undergone proposing the generic AMW [DFB+05b ; DFB+05a ; FDV07]. Claiming that no unified meta–model for mapping is enough to handle every model composition situation, there is a need to build a "variety of small dedicated mapping languages" [DFB+05b, §1, p.2]. They subsequently propose a minimal meta–model for identifying correspondences that can be extended to support specific requirements. The generic meta–model provides no meaning to the correspondences links but designers may provide extensions of these links to define special semantics.

Our approach is similar in a sense that we propose to distinguish a mapping from its interpretation to allow reusing correspondences for various model composition purposes. Our contribution differs such that we propose a set of predefined interpretations to handle situations that often occur. Providing predefined interpretations is twofold : *(i)*it helps proposing precise semantics to a set of given correspondence links ; *(ii)*it allows reusing and capitalizing mappings and their semantics among various model composition operations. While predefined operators may not cope with specific requirements or situations, the extension mechanism that we propose – using the Ad Hoc interpretation – supports the same expressiveness as developing an extension to a weaving link in AMW.

Using AMW, Clasen *et al.* proposed a new model composition approach based on the use of a virtual model [CJC11]. A virtual model is a seamless model that contains elements that are proxies to elements contained in other models : it allows manipulating model elements from contributing models in a single workspace, without taking into account the meta–models that these model elements conform to. While this approach seems to target issues that are different from ours, proposing specific model composition operations for specific correspondence links is still one of the central goals of the approach [7].

1.4.4.3 Object–Relational Mapping

ORM has originally being designed to synchronize OO representations of data with data structures in relational databases. Based on a set of mappings (*i.e.*, annotations) that propose equivalences between *objects* and data in tables, the synchronization mechanism is embedded in the interpreter of the mapping language. Since persistence techniques evolved to support various formats and various kinds of databases, multiple mappings should have been necessary to handle this heterogeneity. ObjectSpaces from Microsoft and Java Data Objects (JDO) handle this heterogeneity making annotations independent from the target persistence storage. Using a single language for annotations, developers can access/store data from/to various data storage.

The goal of proposing a novel definition for model composition in which mappings and their interpretation are separated and loosely–coupled is similar : with a unique

7. Section Correspondence Model in http ://code.google.com/a/eclipselabs.org/p/virtual-emf/wiki/userguide

set of mappings between sets of models, different sets of interpretations should lead to the definition of different model composition techniques.

1.4.4.4 Contribution Challenges

Table 1.9 summarizes how GCFs from the literature meet the proposed objectives for capitalization.

We observe that most GCFs either let designers define their own interpretations or propose a single interpretation for mappings. While some GCFs propose several interpretations for mappings, coupling between mappings and interpretations is still high, thus hindering generalization and reuse of mappings and / or interpretations.

Most GCFs support very few model composition operations. Designed for a specific purpose in a given context, effort and time for adapting these approaches to support different model composition operations is significant.

Capturing a specific composition process is still challenging and surely explain why even the adaptation of GCFs that propose a large number of model composition operations requires manual customization. Techniques for manual customization varies but mostly require developing new application code, rewriting mappings interpreters, modifying existing model composition algorithms or even providing behavior in another kind of representation (*i.e.*, state machines).

The contribution of this thesis focuses on improving the state of practice in building generic model composition frameworks. We propose a theoretical framework that supports the definition of model composition as a pair of a mapping and a set of interpretations. Formalization of the concepts of mapping and interpretation is done separately to keep coupling as low as possible. We propose **a list of predefined mappings and a list of predefined interpretations** as a basis for the construction of specific model composition frameworks.

A pair of a mapping and a set of interpretations define a specific model composition framework for a given purpose. The theoretical framework thus **support various model composition operations** through the construction of **different pairs of mappings and interpretations**.

The model composition process supported by a specific model composition framework depends on domain– or problem– specific characteristics. The contribution of this thesis partially tackles the customization challenge. The **selection of a set of interpretations provides default semantics** to the model composition process and captures characteristics related to the problem (*i.e.*, the purpose of the model composition process). Additional semantics that refer to domain–specific concerns are still to be captured manually.

Characteristics / **GCF**	Predefined Interpretations for mappings	Coupling Mapping / Interpretation	Supports various model composition operations	Composition Process Customization
AMW [DFB+05b]	manual	medium / high	*	manual (Java Methods)
ORM [GG10]	*	medium	1	manual (Interpreter)
Relation–based Approach [CNM11]	*	high	*	manual
Canonical Scheme [BBDF+06]	manual	high	1	manual
Model Management [BHP00]	*	high	*	manual (Operators)
Kompose [FFR+07 ; FBF+08]	1	high	1	manual (Algorithm)
GeKo and SmartAdapter [MKB+08 ; MPL+09]	1	high	2	manual
ReuseWare [HHJ+08]	1	high	*	manual (Combine two atomic operations)
Generic Aspect–Oriented Modeling Framework [MBJ+07]	manual	medium	1	manual
DUALLy [MMP+10]	manual	medium	1	manual (State Machines)

TABLE 1.9 – Comparison of existing generic model composition frameworks (GCFs)

Chapitre 2

A Theoretical Framework for Model Composition

This chapter presents the formalization of main contribution of the thesis, that is a theoretical framework for model composition. Section 2.1 motivates the decomposition of the definition of model composition into a pair of a mapping and a set of interpretations using parallels with structures in mathematical logic and with signs in linguistics. These parallels allows characterizing the relations that exist between a mapping and an interpretation. Section 2.2 proposes mathematical definitions for each kind of mapping and each kind of interpretation proposed in Section 1.2. Section 2.3 discusses how the theoretical framework for model composition allows building model composition frameworks that target specific model composition purposes.

2.1 Decomposing Model Composition

As we observed in the systematic literature review (see Chapter 1), most of the model composition techniques use *(i)*correspondences between models or model elements and *(ii)*a specific interpretation of these correspondences to fulfill the ultimate goal of the model composition operator. The nature of correspondences and their interpretations is still to be discussed and formalized.

We propose to shift from a monolithic definition of model composition (a single definition for a single use – whatever large the spectrum of composable models could be) to a modular definition of model composition (a customizable definition and a choice of existing operators to achieve a possibly large range of goals).

The big picture of the proposition is shown in Figure 2.1. Sections 2.1.1 and 2.1.2 draw parallels with structures in mathematical logic and with signs in linguistics respectively to *(i)*motivate the **separation of correspondences from their interpretation** and to *(ii)***define the relationships that exist between a correspondence and its interpretation**.

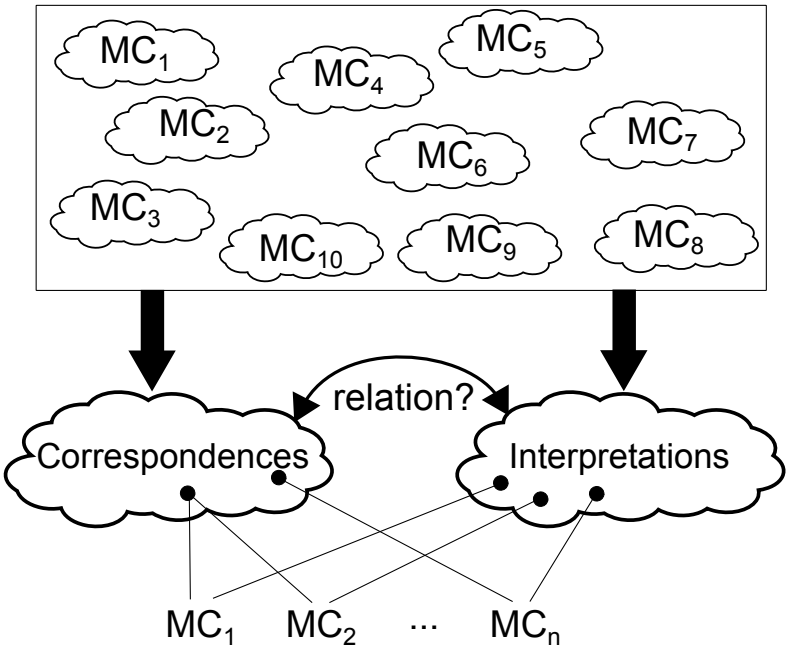


FIGURE 2.1 – Model composition : Moving from monolithic techniques to techniques on-demand

2.1.1 Model Composition is a Structure

Towards demonstrating that model composition is a pair of correspondences and interpretations, we draw a parallel with Structures in mathematical logic.

A Structure in mathematical logic is a triple of a domain, a signature and an interpretation [BS81, §V.1, p.217]. To avoid any misunderstanding in this section, we use the term "goal" to refer to an interpretation in the context of mathematical logic structures.

Let S be a mathematical structure, $|S|$ be the domain of the structure, σ be a signature, and G be a goal such that :

$$S = \langle \mid S \mid, \sigma, G \rangle \tag{2.1}$$

The domain of S is an arbitrary set called the underlying set of the structure, its carrier or its universe. Signature of S is a set of function symbols and relation symbols along with a function that ascribes to each symbol s a natural number ($n = ar(s)$) which is called the arity of s because it is the arity of the "goal" of s. A "goal" function G of S assigns functions and relations to the symbols of the signature. Each function symbol f of arity n is assigned an n–ary function $f^S = G(f)$ on the domain.

From the definition of a structure in mathematical logic, we propose to **decompose model composition as a correspondence language and a specific set of interpretations**. The infinite variability of model composition purposes is an obstacle to the extensive formalization of the "goal" part. Thus we voluntarily discard this part from

our definition of model composition while we still take into account the influence that the purpose has on the definition of a new model composition operator.

We use **`mapping`** in the following explanations to refer to "a correspondence language". A mapping relates model elements – from a single or multiple models – with one another and interpretations provide semantics to the mapping regarding a specific model composition goal.

Let *MC* be the definition of a model composition operator as a mathematical operation, with *MM* a mapping over one or more models and *I* a set of interpretations such that :

$$MC = \langle MM, I \rangle \tag{2.2}$$

Categories proposed in Sections 1.2.1 and 1.2.2 provide background for the formalization of mapping and interpretation. Formal definitions of mapping and interpretation are detailed respectively in Section 2.2.2 and in Section 2.2.3.

2.1.2 Model Composition is a Linguistic Sign

This section draws a parallel between the key concepts in model composition and both linguistics and semiotics *(i)*to strengthen the proposed decomposition of model composition as a pair of a mapping and of an interpretation, and *(ii)*to explore the relationship that exists between a mapping and its interpretation.

Linguistics is a branch of the general science of semiotics that "...investigates the nature of signs and the laws governing them" [Cha08, Introduction, p.9]. In other words, semiotics is the scientific study of human language and includes the work of philosophers, theorists, anthropologists, psychoanalysts, etc., which participate in "...seeking to explore the use of signs in specific social situations" [Cha08, Introduction, p.9].

The definition of a *sign* is closely related with the definition of a linguistic structure : "Linguistic structures are pairing of a meaning and a form. Any particular pairing of meaning and form is a Saussurean sign." [1].

A "Saussurean sign" is defined as follows :

> [Ferdinand de] Saussure offered a 'dyadic' or two-part model of the sign. He defined a sign as being composed of :
> – a 'signifier' (*signifiant*) - the *form* which the sign takes ; and
> – the 'signified' (*signifié*) - the *concept* it represents.
> [Cha08, Signs, p.19]

Similarly to Equation 2.2, our thesis is that a model composition operation (*i.e.*, a *sign*) can be defined as a pairing of a mapping (*i.e.*, a *signifier*) and an interpretation (i.e a *signified*). In other words, a mapping is the definition of the model composition operation and the interpretation gives a precise meaning to the mapping relationships in a specific context.

1. http ://en.wikipedia.org/wiki/Linguistics#Divisions_based_on_linguistic_structures_studied

2.1.2.1 Variability of a Sign

We observed that mapping and interpretation are multiple, depending on the context and the problem that a specific model composition operator handles. The following statement on the definition of a *sign* in semiotics applies equally on the definition of a model composition operator :

> A sign is a recognizable combination of a signifier with a particular signified. The same signifier ... could stand for a different signified (and thus be a different sign).... Similarly, many signifiers could stand for the [signified]... – again, with each unique pairing constituting a different sign. [Cha08, Signs, p.19]

This allows the definition of model composition operators that are flexible, meaning that *(i)*with a given mapping, we can build multiple model composition operators and *(ii)*we can build multiple model composition operators that tackle the same purpose using various mappings. The choice of a specific set of interpretations depends on human–related or problem–related criteria, similarly to what Chandler tells us about the choice of a signifier :

> The use of one signifier rather than another from the same paradigm is based on factors such as technical constraints, code (*e.g.*, genre), convention, connotation, style, rhetorical purpose and the limitations of the individual's own repertoire. [Cha08, Paradigmatic Analysis, p.72]

Quoting that, the general feeling when we observe existing model composition techniques is that mapping and interpretation are rarely distinguished : model composition approaches propose solutions to specific problems, and seldom formalize the underlying purpose of the model composition operator.

Chandler tells us that in semiotics, the signifier and the signified are highly coupled, however, "...the signifier and the signified can be distinguish for analytical purposes." [Cha08, Signs, p.21].

This statement closely matches our reasoning approach : we want to **separate mapping from its interpretation to enhance the inherent reusability of these two concepts**.

2.1.2.2 Mapping and Interpretation Coupling

Identifying the relationship between a mapping and its interpretation is necessary to identify highly reusable model composition techniques and to propose pairs of mapping/interpretation that are adequate and relevant.

Since interpretation is the meaning of a mapping, we propose that *default* meaning is such that mapped elements are in relation with one another. This definition is not sufficient to capture the goal of a specific model composition technique. Thus, we need to use concepts of denotation and connotation to further characterize the goal of the model composition operator.

In semiotics, "...denotation and connotation are terms describing the relationship between the signifier and its signified..." [Cha08, Denotation, Connotation and Myth,

p.90] (see Figure 2.2). A *signified* is bound to a meaning that "...includes both denotation and connotation." [Cha08, Denotation, Connotation and Myth, p.90].

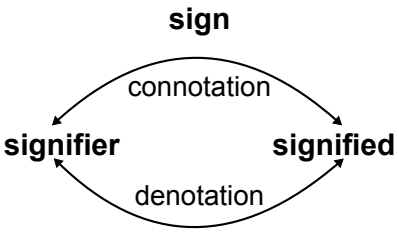


FIGURE 2.2 – A simplified representation of a sign and the relationships between the signifier and the signified.

Chandler gives a more precise definition of denotation and connotation that is :

> 'Denotation' tends to be described as the definitional, 'literal', 'obvious' or 'commonsense' meaning of a sign. In the case of linguistic signs, the denotative meaning is what the dictionary attempts to provide....
> ...
> The term 'connotation' is used to refer to the socio-cultural and 'personal' associations (ideological, emotional etc.) of the sign.
> ...
> Connotations are not purely 'personal' meanings - they are determined by the [cultural] codes to which the interpreter has access. [Cha08, Denotation, Connotation and Myth, p.90]

Denotation and connotation influence the definition of a model composition operator. Selecting a specific mapping and specific interpretations to build a new model composition operator is not enough. The goal of the model composition operator is still to be captured since "[c]hanging the form of the signifier while keeping the same signified can generate different connotations" [Cha08, Denotation, Connotation and Myth, p.92] and "...connotation is very much a question of how [the] language [that includes the signs] is used." [Cha08, Denotation, Connotation and Myth, p.93].

The meaning of a pair of a mapping and an interpretation is given by a specific connotation which represents the concept of goal in the definition of a structure in mathematical logic. Definition of connotation and denotation is still too extensive to propose a formal definition of the goal. However the variability of denotation/connotation illustrates how the goal of a model composition operation may vary.

In the scope of the formalization of a model composition operator, we propose to derive the definitions of connotation and denotation such that :

Definition 1 *Denotation*
Denotation is the generic meaning of a pairing of a mapping representation and an interpretation of this representation : model elements from one model **relate** *with model elements from another model.*

Definition 2 Connotation
Connotation refines the **relate** *meaning into a meaning that takes into account a specific context and a given goal to achieve.*

Since the definition of connotations involves capturing a context that depends on the designers using the model composition operator and depends on the model composition expectations (see Figure 2.3), we propose to use a General Purpose Language (GPL) to realize the purpose of a model composition operator defined by a given mapping and a given interpretation for a specific intention.

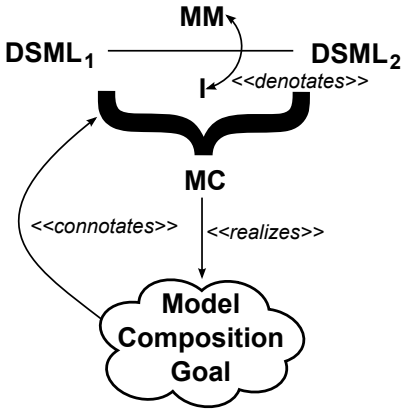


FIGURE 2.3 – The goal of a specific model composition operator influences the relationship between a mapping between two meta–modeling languages and the interpretation of the mapping.

2.1.2.3 From Linguistics to Model–Driven Engineering

Barthes tells us that "language is a pure abstract entity, ..., a set of basic types that speech makes concrete in infinite variable ways" [Bar64, §I.1.4, p.22]. This definition is sound regarding the definition of a meta–modeling language : a meta–modeling language defines a set of basic types that are captured into a meta–model ; models that conform to this meta–model instantiate the meta–model types in infinite variable ways.

This statement reinforces even further the relevance of the parallel between model composition and semiotics (see Section 2.1.2) in supporting the formal definition of a model composition operator in the context of MDE.

The definition of the form of a signifier and of a signified proposed by Chandler makes a smooth transition between semiotics and the concepts of mapping and interpretation since :

- Signifiers :...[are expressed as a] language, formal syntactic structure...
- Signifieds :...[can take the form of a] 'semantic structure' (Baggaley & Duck[Dynamics of Television, 1976])...

[Cha08, Signs, p.39]

We may conclude that a mapping *MM* is a "formal syntactic structure" (*i.e.*, a modeling language) and an interpretation of *MM* is the intended meaning of *MM*. Section 2.2 proposes a formalization of *MM* and *I*.

2.2 Towards a Unified Theory for Model Composition

This section presents the formal definition of a model composition operator and the basic constructs of the theoretical framework. We propose a formalization of the key concepts observed in the systematic literature review (see Section 1.4) and we discuss their usage to build new model composition operators from the reuse of existing techniques. The intent of this section is to propose an interpretive lens for analyzing model composition techniques not covered in the systematic literature review and to provide a framework for building new model composition operators.

Definitions from this section are based on the following assumptions :

1. A mapping exists between a set of meta–modeling languages.
2. A model composition operator based on this mapping allows composing the meta–modeling languages.
3. Execution of the model composition operator is allowed on the models that conform to these meta–modeling languages.

For the sake of simplicity, definitions for mapping, interpretation, and model composition involve only two meta–modeling languages. Nevertheless, the unified framework for model composition is applicable to an arbitrary number of meta–modeling languages.

Section 2.2.1 presents the mathematical symbols and definitions that we use in the formalization. Section 2.2.2 proposes a formalization for each kind of mapping and Section 2.2.3 proposes a formalization of each kind of interpretation. Section 2.2.4 revisits the definition of model composition in the context of domain–specific languages.

2.2.1 Mathematical Symbols and Definitions

This section lists the concepts, sets, function and symbols that we use to propose formal definition of model composition, mapping and interpretation.

2.2.1.1 Domain–Specific Modeling Language

Let *DSML* represents a meta–modeling language.
Let AS_M represents the abstract syntax of $DSLM_M$ (*i.e.*, the set of types that $DSML_M$ defines).
Let CS_M represents the concrete syntax of $DSLM_M$ (*i.e.*, the textual or graphical representation of concepts from $DSML_M$).
Let $M_{AS_M \rightarrow CS_M}$ represents the mapping from the abstract syntax to the concrete syntax.
Let SD_M represents the semantic domain of $DSLM_M$.

Let $M_{AS_M \rightarrow SD_M}$ represents the mapping from the abstract syntax to the semantic domain [HR04].

A meta–modeling language M is a five–part tuple such that :

$$DSML_M = \langle AS_M, CS_M, M_{AS_M \rightarrow CS_M}, SD_M, M_{AS_M \rightarrow SD_M} \rangle \quad (2.3)$$

2.2.1.2 Sets

Let $Boolean = \{TRUE, FALSE\}$ the set of boolean values.
Let $\mathbb{E}$ represents the set of all possible model elements.
Let $\mathbb{M}$ represents the set of meta–types (*i.e.*, the types of the meta-classes) of all possible model elements.
Let $\mathbb{E}_M \subset \mathbb{E}$ be the set of model elements of a model M.
Let $\mathbb{M}_M \subset \mathbb{M}$ be the set of meta–types of $\mathbb{E}_M$.
Let $\mathbb{P}_M X$ represents the set of all properties of a meta–type MX.
Let $\mathbb{C}$ a set of constraint expressions.
Let $\mathbb{E}_M^{\boxplus} \subset \mathbb{E}_M$ be the set of model elements of a model M labeled to be added.
Let $\mathbb{E}_M^{\boxminus}$ $\mathbb{E}_M$ be the set of model elements of a model M labeled to be removed.
Let $\mathbb{E}_M^{\boxdot} \subset \mathbb{E}_M$ be the set of model elements of a model M labeled to be modified.
Let $Type \subset \mathbb{M}$ be a meta–type.
Let $DSML_C$ be the model that results from the execution of the model composition operation.
Let $A \subset \mathbb{E}_A$ be a set of model elements from $DSML_A$ such that : $A = \{x_1, x_2, \ldots, x_i\}$.
Let $B \subset \mathbb{E}_B$ be a set of model elements from $DSML_B$ such that : $B = \{y_1, y_2, \ldots, y_j\}$.
Let $C \subset \mathbb{E}_C$ be a set of model elements from $DSML_C$ such that : $C = \{z_1, z_2, \ldots, z_m\}$.
Let $D \subset \mathbb{E}$ be a set of model elements such that : $D = \{w_1, w_2, \ldots, w_n\}$.

2.2.1.3 Functions and Relations

Let $\mathfrak{R} : \mathbb{E}_M, \mathbb{E}_M$ be a relation of mappings between two set of model elements.
Let $eval : \mathbb{C}, Type, Type \rightarrow Boolean$ be the function that evaluates a given constraint on a pair of model elements.
Let $occ : Type, \mathbb{E}_M \rightarrow \mathbb{E}_M$ be the function that retrieves an occurrence of a model element of type $Type$ in a set of model elements.
Let $super : Type \rightarrow \mathbb{M}_{M_X} \cup \emptyset$ be a function which returns the set of super types for the type X.
Let $ref : \mathbb{E}_M \rightarrow \mathbb{E}_M$ be a function that captures a reference between two sets of model elements.
Let $containment : \mathbb{E}_M \rightarrow \mathbb{E}_M$ be a function that captures the container of a set of model elements. A container owns a *containment* reference that targets contents.
Let $match : \mathbb{E}_M, \mathbb{E}_M \rightarrow Boolean$ be a function that detects whether two sets of model elements overlaps (see Section 1.2.2).
Let $joinpoint : \mathbb{E}_M, \mathbb{E}_M \rightarrow Boolean$ be a relation that detects if two sets of model elements are cross–cutting (see Section 1.2.2).

Let *invoke* : $\mathbb{E}_M$ be a procedure that executes the behavior a set of behavioral model elements (see Section 1.2.2).
Let *align* : $\mathbb{E}_M, \mathbb{P}_M X \rightarrow \mathbb{E}_M$ be a function that modifies a property X on a model element of type M.
Let *prop* : $\mathbb{E}_M \rightarrow \mathbb{P}_M X$ be a function that retrieves the properties of a model element X of type M.

2.2.1.4 Symbols

Let $\prec$ the precedence operator from partial ordering that means in this context "is invoked before".
Let $\bigcirc\!\!\rightarrow_X$ be the time at which the execution of a model element X starts.
Let $\leftarrow\!\!\odot_X$ be the time at which the execution of a model element X ends.

2.2.2 Mapping Definition

A mapping MM is a DSML that captures the key concepts of relationships over the elements of others DSML. MM is formally defined by its abstract syntax (AS_{MM}), its semantic domain (SD_{MM}) and the mapping from the abstract syntax to the semantic domain ($M_{AS_{MM} \rightarrow SD_{MM}}$) such that :

$$MM = \langle\, AS_{MM}, SD_{MM}, M_{AS_{MM} \rightarrow SD_{MM}} \,\rangle \tag{2.4}$$

We voluntarily discard the representation of the concrete syntax (CS_{MM}) and the mapping from the abstract syntax to the concrete syntax ($M_{AS_{MM} \rightarrow CS_{MM}}$) since the choice of a concrete syntax has ultimately no impact on the definition of relations over the elements of several DSML.

Since MM is a set of n–ary relationships over the elements of the abstract syntaxes of n DSML ($\{AS_{DSML_1}, \ldots, AS_{DSML_2}\}$), we consider that MM is equal by definition to a set of alignment rules (R_a) such that :

$$MM \in \mathfrak{R}(AS_{DSML_1}, \ldots, AS_{DSML_n}) \triangleq \sum_{i=1}^{n} R_a(DSML_i, DSML_{i+1}),\ s.t.\ n \geq 1 \tag{2.5}$$

The definition of an alignment rule varies in function of (1) the concrete representation of AS_{MM} and (2) the concrete mapping formalism. For instance, if a DSML is a graph-structure, an alignment rule would be a n–ary relationship between graph patterns (*i.e.*, a graph morphism).

From Section 1.2.1, we proposed six kinds of mapping representation. These representations are concrete examples for the definition of n–ary alignment relationships that work on the abstract syntaxes of the DSML that we want to map with one another. The following equations formalize the various kinds of mappings proposed in the category in Section 1.2.1.

2.2.2.1 Operator-based Mapping

An operator-based mapping $\mathfrak{R}$ is an application between the abstract syntaxes of the two DSML. Since the behavior of the operator cannot be defined extensively, we define $\mathfrak{R}$ using the f and g applications that respectively produce the image of the elements from $DSML_A$ and $DSML_B$, such that :

$$\begin{array}{l} \mathfrak{R}_{f,g} : Type^n \times Type^n \\ A\ \mathfrak{R}_{f,g}\ B \Leftrightarrow \left| \begin{array}{l} (f(A) \rightarrow\ B) \wedge \\ (g(B) \rightarrow\ A) \end{array} \right. \end{array} \tag{2.6}$$

2.2.2.2 Pattern- or Rule- based Mapping

A pattern– or rule– based mapping $\mathfrak{R}$ is the identification of elements with similar characteristics such that :

$$\begin{array}{l} \mathfrak{R}_{occ} : Type^n \times Type^n \\ A\ \mathfrak{R}_{occ}\ B \Leftrightarrow \left| \begin{array}{l} x \in A, y \in B\ s.t. \\ occ(y, A) \neq \emptyset\ \wedge\ occ(x, B) \neq \emptyset \end{array} \right. \end{array} \tag{2.7}$$

2.2.2.3 Constraint-based Mapping

A constraint-based mapping $\mathfrak{R}$ rely on the evaluation of a constraint c to detect the presence of a model element such that :

$$\begin{array}{l} \mathfrak{R}_{c} : Type^n \times Type^n \\ A\ \mathfrak{R}_{c}\ B \Leftrightarrow \left| \begin{array}{l} c \in \mathbb{C}, x \in A, y \in B\ s.t. \\ eval(c, x, y) = TRUE \end{array} \right. \end{array} \tag{2.8}$$

2.2.2.4 Model-based Mapping

A model-based mapping $\mathfrak{R}$ is itself a DSML whose abstract syntax (AS_M) includes model elements from the abstract syntaxes of the two DSML (AS_{DSML_A}, AS_{DSML_B}) such that :

$$\begin{array}{c} \mathfrak{R} : DSML_M = \langle\ AS_M, SD_M, M_{AS_M \rightarrow SD_M}\ \rangle\ s.t. \\ AS_M \triangleq AS_{DSML_A} \cup AS_{DSML_B} \end{array} \tag{2.9}$$

The new DSML built from the abstract syntaxes of the DSML is a language that manipulates and relates model elements with one another, providing enough information for identifying equivalent, similar or different model elements.

2.2.2.5 Delta-based Mapping

A delta-based mapping $\mathfrak{R}$ provides information about at least the two situations as follows :

- A set of model elements is found in one of the DSML but not in the other. If model elements are found in $DSML_A$ but not in $DSML_B$, we consider these elements as added ($\mathbb{E}_A^{\boxplus}$). If model elements are found in $DSML_B$ but not in $DSML_A$ we consider them as $\mathrm{deleted}$ ($\mathbb{E}_A^{\boxminus}$).
- A set of model elements is found in both DSML but the characteristics of these model elements are not identical. We consider them as $\mathrm{modified}$ ($\mathbb{E}_A^{\boxdot}$).

$$\begin{array}{l} \mathfrak{R}_\Delta : Type^n \times Type^n \\ A\ \mathfrak{R}_\Delta\ B \Leftrightarrow \left| \begin{array}{l} x \in A, y \in B\ s.t. \\ \mathbb{E}_A^{\boxplus} = \mathbb{E}_A^{\boxplus} \cup \{x\} \Rightarrow x \in (A \setminus (A \cap B)) \\ \mathbb{E}_A^{\boxminus} = \mathbb{E}_A^{\boxminus} \cup \{y\} \Rightarrow y \in (B \setminus (A \cap B)) \\ \mathbb{E}_A^{\boxdot} = \mathbb{E}_A^{\boxdot} \cup \{x\} \Rightarrow x \in (A \cap B)\ \wedge\ x \neq y \end{array} \right. \end{array} \tag{2.10}$$

2.2.3 Interpretation Definition

The interpretation of a mapping participates in providing the semantics (SD_{MM}) of the mapping language. However, keeping mapping and interpretation loosely coupled implies that an interpretation I participates in the definition of SD_{MM} but is not sufficient for expressing the whole semantics of the mapping language regarding the final purpose of the mapping. Formalizing "human content" (the substance of the signified [Chandler, 2008, p.39]) or expectations is difficult and is out the scope of this work. However, we propose to build a DSML to capture some of these "expectations" such that :

$$I = \langle\ AS_I, SD_I, M_{AS_I \to SD_I}\ \rangle \tag{2.11}$$

where AS_I is the set of interpretations that we propose in the category in Section 1.2.2 and SD_I and $M_{AS_I \to CS_I}$ being the set of definitions for each interpretation that we propose in Section 1.3.5.2.

Sections 2.2.3.1 to 2.2.3.5 present a formal definition of each category of interpretations.

2.2.3.1 "Add" Interpretation

A *Add* interpretation allows us to add a set of model elements into another model such that :

$$\begin{array}{l} + : Type^n \times Type^n \\ A + B \Leftrightarrow \left| \ C = C \cup (A \cup B) \right. \end{array} \tag{2.12}$$

2.2.3.2 "Delete" Interpretation

A *Delete* interpretation allows us to remove a set of model elements from another model such that :

$$\begin{array}{l} - : Type^n \times Type^n \\ A - B \Leftrightarrow \left| \ C = C \cup (A \setminus (A \cap B)) \right. \end{array} \tag{2.13}$$

2.2.3.3 Overlapping

"Equivalence" Interpretation An *Equivalence* interpretation allows us to identify two sets of model elements from two DSML that we consider to have the same semantics. Let $\triangleq$ the operator meaning *is equal by definition* such that :

$$\begin{array}{l} \triangleq : Type^n \times Type^n \\ A \triangleq B \Leftrightarrow \left| \begin{array}{l} x \in A, y \in B \; s.t. \\ x = y \Rightarrow C = (C \cup \{x\}) \; \vee \; (C \cup \{y\}) \end{array} \right. \end{array} \tag{2.14}$$

"Similarity" Interpretation A *Similarity* interpretation allows us to identify two sets of model elements from two DSML that have a close but different structure. *Similarity* means that we need to align the two structures with each other such that :

$$\begin{array}{l} \approx : Type^n \times Type^n \\ A \approx B \Leftrightarrow \left| \begin{array}{l} x \in A, y \in B, p \in prop(x), p' \in prop(y) \; s.t. \\ x \approx y \Rightarrow ((align(x,p) = p') \; \wedge \; (C = C \cup \{x\})) \vee \\ ((align(y,p') = p) \; \wedge \; (C = C \cup \{y\})) \end{array} \right. \end{array} \tag{2.15}$$

This interpretation often requires additional inputs to correctly handle changes and to produce the expected result. Additional inputs are encompassed in the *align* function.

"Generalization" Interpretation A *Generalization* interpretation is applicable on meta–models only since it modifies the inherent structure of the target modeling language. *Generalization* allows us to transform one set of model elements into a specialization of another set of model elements such that :

$$\begin{array}{l} \multimap : Type^n \times Type^n \\ A \multimap B \Leftrightarrow \left| \begin{array}{l} x \in A, y \in B \; s.t. \\ super(x) \cup \{y\} \end{array} \right. \end{array} \tag{2.16}$$

"Aggregation" Interpretation An *Aggregation* interpretation is applicable on meta–models only since it modifies the inherent structure of the target modeling language. *Aggregation* allows us to create a *containment* relationship between two sets of model elements such that :

$$\begin{array}{l} \blacklozenge : Type^n \times Type^n \\ A \blacklozenge B \Leftrightarrow \left| \begin{array}{l} x \in A, y \in B \; s.t. \\ C = C \cup \{x\} \; \wedge \; C = C \cup \{y\} \; \wedge \; containment(x) = y \end{array} \right. \end{array} \tag{2.17}$$

"Overriding" Interpretation An *Overriding* interpretation allows us to replace an existing set of model elements with another set of model elements at places provided by the function *match* such that :

$$\begin{array}{l} :=: Type^n \times Type^n \\ A := B \Leftrightarrow \left| \begin{array}{l} x \in A, y \in B \; s.t. \\ match(x,y) \Rightarrow (C = (C \setminus \{x\}) \; \wedge \; (C = C \cup \{y\})) \end{array} \right. \end{array} \tag{2.18}$$

"Information–Gap" Interpretation An *Information–Gap* interpretation allows us to define a specific structure (*i.e.*, a set of model elements) to relate the sets of model elements that originate from the two DSML. Let q and r be two properties owned by a specific structure D that helps binding the two sets of model elements such that :

$$\begin{array}{l|l} \leftrightsquigarrow : Type^n \times Type^n & \\ A \leftrightsquigarrow B \Leftrightarrow & x \in A, y \in B, w_i \in D, w_j \in D \; s.t. \\ & \exists q \in prop(w_i), \exists r \in prop(w_j) \; s.t. \\ & C = C \cup A \;\wedge\; C = C \cup B \;\wedge \\ & ((ref(x) = q \;\wedge\; ref(y) = r) \vee \\ & (ref(q) = x \;\wedge\; ref(r) = y)) \wedge \\ & C = C \cup D \end{array} \tag{2.19}$$

"Ad Hoc" Interpretation An *Ad Hoc* interpretation allows us to define an arbitrary-complex computation on one set of model elements of the DSML or the other. Let op1 and op2 be two operations that defines this computation such that :

$$\begin{array}{l|l} op1 : Type^n \longrightarrow Type^n & op2 : Type^n \longrightarrow Type^n \\ op(A) = B & op(B) = A \end{array} \tag{2.20}$$

2.2.3.4 Cross–Cutting

"Replace" Interpretation A *Replace* interpretation allows us to switch one set of model elements from one DSML with another set of model elements from another DSML. Places to change are identified by the *joinpoint* function which is provided by users. We define *Replace* such that :

$$\begin{array}{l|l} :=: Type^n \times Type^n & \\ A := B \Leftrightarrow & x \in A \,, y \in B \; s.t. \\ & joinpoint(x, y) \Rightarrow (C = (C \setminus \{x\}) \;\wedge\; (C = C \cup \{y\})) \end{array} \tag{2.21}$$

"Augment" Interpretation An *Augment* interpretation allows us to mix two sets of elements together such that :

$$\begin{array}{l|l} += : Type^n \times Type^n & \\ A \mathrel{+}= B \Leftrightarrow & x \in A, y \in B, z \in C \; s.t. \\ & (C = C \cup \{z\}) \Rightarrow prop(z) = prop(x) \cup prop(y) \end{array} \tag{2.22}$$

"Remove" Interpretation A *Remove* interpretation allows us to remove parts of two sets of elements that these sets have in common (*i.e.*, a mathematical projection) such that :

$$\begin{array}{l|l} \setminus \; : Type^n \times Type^n & \\ A \setminus B \Leftrightarrow & x \in A, y \in B, z \in C \; s.t. \\ & (C = C \cup \{z\}) \Rightarrow prop(z) = prop(x) \setminus prop(y) \end{array} \tag{2.23}$$

2.2.3.5 Interaction

"Sequence" Interpretation A *Sequence* interpretation allows us to sequence the communication of two sets of model elements from two DSML such that :

$$\begin{array}{l} ; : Type^n \times Type^n \\ A\ ;\ B \Leftrightarrow \mid invoke(A) \prec invoke(B) \end{array} \tag{2.24}$$

"Parallel" Interpretation A *Parallel* interpretation allows us to invoke the communication of two sets of model elements from two DSML at the same time. Let $\|$ the parallel operator such that :

$$\begin{array}{l} // \ : Type^n \times Type^n \\ A\ //\ B \Leftrightarrow \mid invoke(A) \parallel invoke(B) \end{array} \tag{2.25}$$

"Codependency" Interpretation A *Codependency* interpretation is a set of causal relationships between sets of model elements from DSML. Let $\circ\!\!-\!\!\circ$, $\circ\!\!-\!\!\bullet$, $\bullet\!\!-\!\!\circ$ and $\bullet\!\!-\!\!\bullet$ the kinds of causal relationships respectively meaning *start-to-start*, *start-to-finish*, *finish-to-start* and *finish-to-finish* such that :

$$\begin{array}{l} \overset{\bullet}{\underset{\bullet}{\mid}} : Type^n \times Type^n \\ \quad x \in A, y \in B\ s.t. \\ x \overset{\bullet}{\underset{\bullet}{\mid}} y \Leftrightarrow \left| \begin{array}{ll} x \circ\!\!-\!\!\circ\ y & \circ\!\!\rightarrow_X \prec \circ\!\!\rightarrow_Y \\ x \circ\!\!-\!\!\bullet\ y \Leftrightarrow & \circ\!\!\rightarrow_X \prec \leftarrow\!\!\odot_Y \\ x \bullet\!\!-\!\!\circ\ y \Leftrightarrow & \leftarrow\!\!\odot_X \prec \circ\!\!\rightarrow_Y \\ x \bullet\!\!-\!\!\bullet\ y \Leftrightarrow & \leftarrow\!\!\odot_X \prec \leftarrow\!\!\odot_Y \end{array} \right. \end{array} \tag{2.26}$$

2.2.4 Model Composition is a DSML

The definition of a model composition operator as a pair of a mapping *MM* and interpretations of *MM* given in Equation 2.2 allows the definition of operators which semantics is given by both *I* and the global purpose of the model composition process. Going one step further, we propose to define a specific DSML to formalize model composition such that :

$$MC = \langle\ AS_{MC}, SD_{MC}, M_{AS_{MC} \rightarrow SD_{MC}}\ \rangle \tag{2.27}$$

AS_{MC} is defined on top of the abstract syntax (AS_{MM}) of the mapping DSML *MM* (see Equation 2.4) and consequently contains all the concepts of the abstract syntax of the mapping modeling language. The semantic domain SD_{MC} of the modeling language for model composition *MC* is defined on top of the abstract syntax of the interpretation DSML (AS_I) (see Equation 2.11). Taking into account AS_{MM} and AS_I in the definition of the modeling language for model composition, we ultimately rewrite the mapping $M_{AS_{MC} \rightarrow SD_{MC}}$ with $M_{AS_{MM} \rightarrow AS_I}$ such that :

$$MC = \langle\ AS_{MM}, AS_I, M_{AS_{MM} \rightarrow AS_I}\ \rangle \tag{2.28}$$

This allows us to *bind* specific interpretations to a specific mapping that subsequently allows us to build a specific model composition operator. This *binding* ($M_{AS_{MM} \rightarrow AS_I}$) holds information about the context and the purpose of the model composition operator.

Since AS_{MM} is the union of the elements of interest in the abstract syntaxes of the n DSML that are part of the mapping DSML (see Equation 2.9), we rewrite MC such that :

$$MC = \langle \bigcup_{j=1}^{n} AS_{DSML_1} \times AS_{DSML_j}, AS_I, M_{AS_{MM} \rightarrow AS_I} \rangle \tag{2.29}$$

A model composition DSML is thus capable of *(i)*providing mapping explicitly for a set of DSML, of *(ii)*proposing a set of interpretations to capture the semantics of the mapping , and of *(iii)*considering the ultimate purpose of the model composition operation.

Note that the global purpose of the model composition language needs additional input (connotation) to properly capture the purpose and context of the execution of the model composition operation.

2.3 Conclusion

Building a model composition operator that addresses a new model composition situation is a difficult and time-consuming task. The development of such effective model composition operators implies the definition of mappings or the adaptation of existing ones, and the development of the tool that supports such model composition operator.

Our contribution focuses on the activity of designing model composition operators. With the help of the categorization of mappings and interpretations and the formalization (see Section 2.2), we propose a two–step process to build specific model composition operators :

1. Designers select a kind of mapping that fits their needs and a set of interpretations that are in adequateness with the purpose of the model composition operator. This pair of mapping and interpretations is the specification of a model composition framework that supports a specific model composition operation.
2. The model composition framework should be customized to execute the specific model composition operation in a given context. The context refers both to the kind of modeling languages that the model composition operation should support and specific characteristics of the model composition process that are not captured by the set of interpretations.

We illustrate this two–step process of building a specific model composition operator on model merging in Figure 2.4. A given pair of mappings and interpretations is selected to support the very general purpose of model merging. Designers customize the general purpose of the model merging operator by proposing specific connotations

(algorithms) to support merging models that conform to various formalisms. For instance, Figure 2.4 illustrates the customization process for proposing a model merging operator that supports merging UML and ECore models. The customization process allows the definition of homogeneous or heterogeneous model merging operators, according to the designers expectations.

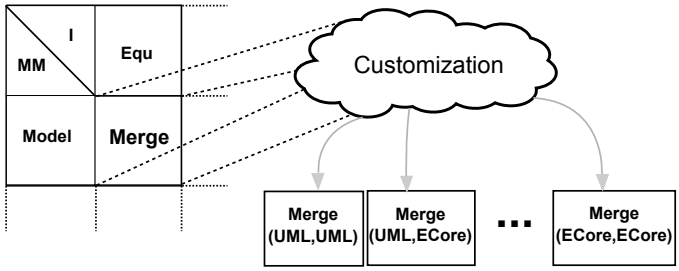


FIGURE 2.4 – A given pair of a mapping and an interpretation that supports model merging is refined into specific model composition operators

Proposing this two–step process, we limit the effort in designing a new model composition operation to the customization of the model composition process. We expect that proposing pairs of mapping and interpretations should become a sporadic activity over time as more and more *generic* frameworks would be available.

In Chapter 3, we leverage the formalization of model composition as a pair of a mapping and a set of correspondences to propose a framework for *(i)*unifying the definition of model composition activities and for *(ii)*for building concrete model composition operators. The proposition of the framework is described using the running example of building a model composition operator for model merging homogeneous models.

Chapitre 3

ModMap : A Framework for Unifying Model Composition Activities

According to the formalization of model composition proposed in Chapter 2, we present in this chapter a tool called MODel MAPping (ModMap) that supports the definition of a mapping and its interpretation for producing an effective model composition language and concrete model composition operators (see Section 3.2.2).

Section 3.1 presents an intuitive process for the definition of a model composition framework that supports model merging. The intuitive process is generalized at the end of Section 3.1 for the definition of new model composition frameworks. Section 3.2 presents the ModMap framework that supports the definition of mappings and that proposes default semantics for interpretations.

3.1 An Intuitive Process for Building Model Composition Frameworks

3.1.1 A Running Example

The Bank model (see Figure 3.1) illustrates a simple bank data model. The BLP model (see Figure 3.2) describes a Bell-LaPadula (BLP) access control feature [BL73] where users and objects under access control are each associated with a security level. User access control is driven by the domination relationship that ensures for instance that a bank user can access his account only if the user's security level dominates the security level of the account. The expected composed model should merge the two models into a consistent model that allows a *Controller* to call operations on *BankUser* or *Account*, with respect to the security levels defined.

Intuitively, designers identify that *Account* and *BankUser* from the Bank model are equivalent to *Account* and *BankUser* from the BLP model, respectively. The identification of overlapping model elements is thus based on names : if a model element from

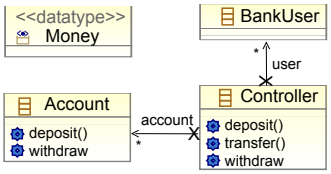


FIGURE 3.1 – The Bank model.

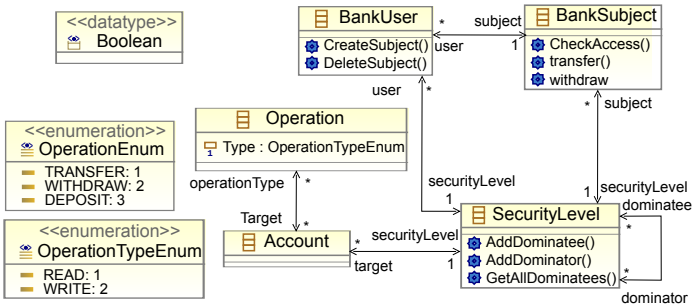


FIGURE 3.2 – The BLP model.

the Bank model has a name that equals the name of a model element in the BLP model, designers would like to create a single model element from these two concepts.

Using a graphical representation to ease the specification of overlapping elements between two models, designers relate *Account* and *BankUser* elements from the Bank model with *Account* and *BankUser* elements from the BLP model. Figure 3.3 is an example of graphical representation of overlapping relationships between the Bank model and the BLP model : dotted lines are mappings that state which elements overlap and *equals* indicates the semantics of the mapping.

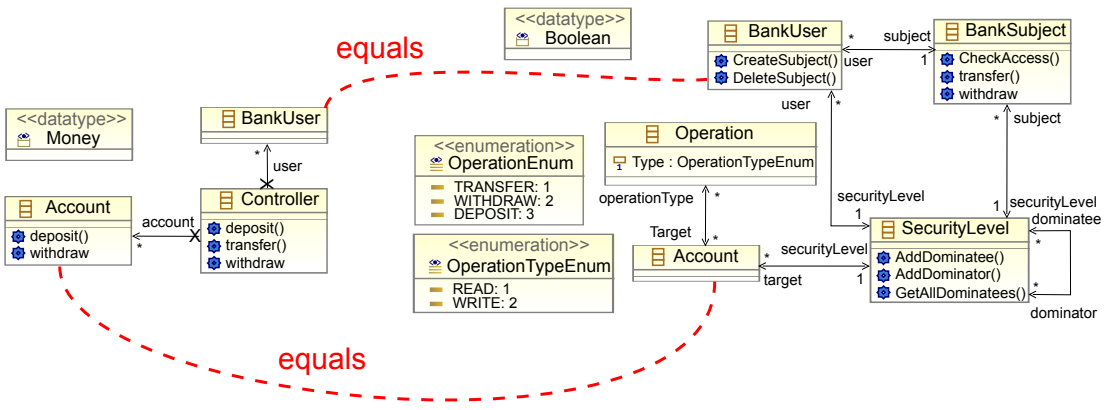


FIGURE 3.3 – Intuitive relationships of overlap between the Bank and BLP models

3.1.2 A Framework for Model Merging

This section presents the intuitive process of building a model composition framework for model merging on the running example and illustrated on Figure 3.4. A designer who had an objective of merging models selects a pair of a mapping and a set of interpretations. This selection is the definition of a generic model composition framework for model merging. The model merging operation executes in a specific context and for a specific purpose : this information is provided by designers that parameterize the framework for model merging to ultimately build a specific model composition operator.

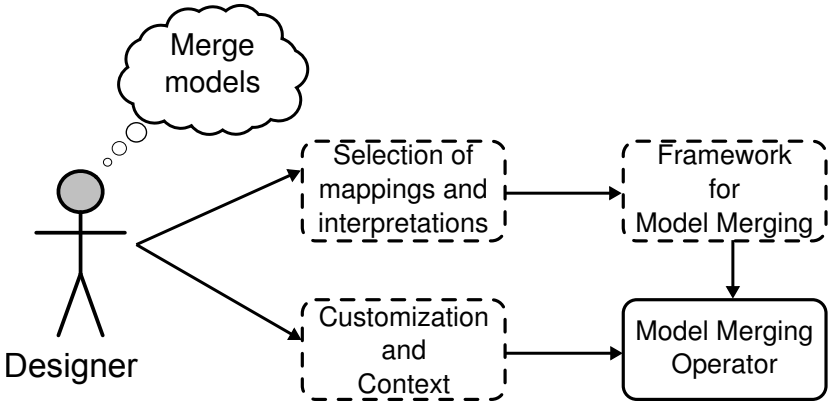


FIGURE 3.4 – Intuitive process for building a model composition framework for model merging.

3.1.2.1 Selection of a pair of Mapping and a set of Interpretations

On the example of merging the Bank model with the BLP model, designers intuitively use an equivalence interpretation (see Section 2.2.3) that allows them to identify a set of equivalent model elements. In the definition of a model composition operation, the choice of a kind of mappings is equally important. Designers choose to use a model representation of mappings. This means that they use a modeling language to create a model of correspondences between the Bank model and the BLP model.

3.1.2.2 Customization of the Framework

The selection of a pair of a mapping and a set of interpretations captures the scope of the model composition operation. The model composition framework for model merging is based on a model representation of mappings and an equivalence interpretation. Thus, the framework allows designers to identify overlapping model elements. Towards building a tool that supports merging two overlapping models, designers should capture the context in which the model merging operation runs. It means that designers should propose specific processing to take into account the

nature of the models to merge and the specific characteristics of the merge operation that are not captured by the interpretations.

We illustrate the process of providing a specific processing for model merging as a *merge()* function presented in Listing 3.1. The *merge()* function takes two model elements as parameters and performs the union of their properties to create a single model element that is the union of the two originating model elements.

```
function merge(A : Object, B : Object) : Object is
  pre type(A) = type(B)
  // creates a new object
  C = new object from type(A)
  // union of the properties
  foreach property p1 from A
    foreach property p2 from B
      if p1 equals p2 then
        // merge values and resolve conflicts if any
        new_value = resolveConflicts(value(p1),value2(p2))
      end
    // add property to C
    end foreach
  end foreach
end
```

Listing 3.1 – A simplified algorithm for model merging.

The execution of the *merge()* function within the model composition framework for model merging produces a new model that contains elements from the Bank model and the BLP model (see Figure 3.5). Model elements that designers identified as overlapping are "combined" into a single element that contains all the properties of the originating model elements.

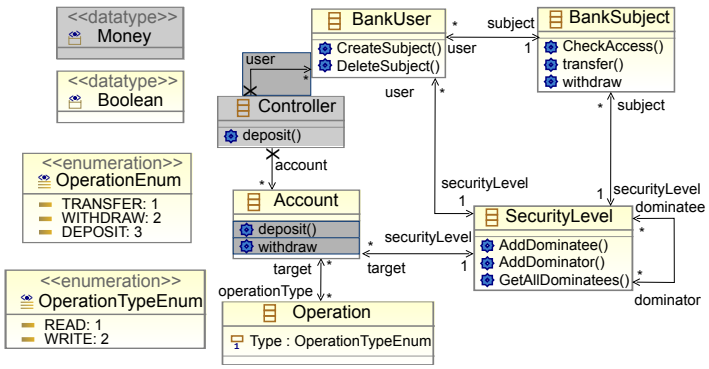


FIGURE 3.5 – Result of merging Bank and BLP. Grayed elements were owned by Bank and have been merged within the set of model elements from BLP.

3.1.3 Generalization of the Intuitive Process

Based on the intuitive process of building a model composition framework for model merging, we propose to generalize the approach to support the definition of

model composition frameworks that target various purposes.

We presented categories of mappings and interpretations in Chapter 1 and we proposed a formalization of these categories in Chapter 2. Towards the definition of model composition frameworks, the ModMap tool supports every kind of interpretations to ease the definition of the meaning of a specific mapping relationship. Among the kinds of mappings, we consider that operators, patterns, rules, constraints and deltas properly support detection and/or expression of mappings at the model level. As stated in Section 1.2.1.5, these kinds of mappings are often part of a (semi–) automatic process that builds model–based representations. Subsequently, we choose to use a model of mapping to successfully represent a wider range of mapping definitions and to address a wider range of model composition activities.

Since a model of mapping is specific to a given context and for a specific purpose, we propose a modular model-alignment modeling language that designers customize for their own purpose. The model-alignment modeling language supports the definition of mappings between models or model elements and allows the selection of interpretations for these mappings.

Customization of the model composition framework is capitalized in the selection of interpretations for a set of mappings. However this set of interpretations is not enough to capture the specific characteristics of the model composition operation. Context–specific and purpose–specific data has to be provided by the designers.

3.2 The ModMap Framework

The ModMap framework proposes a model–alignment modeling language and customization capabilities to build specific model composition frameworks in a given context and for a given purpose. A specific model composition framework supports the definition of a model–alignment modeling language that results from the definition of mappings between models or sets of models elements and from the selection of adequate interpretations for these mappings.

3.2.1 Architecture Overview

This section presents the global architecture of the ModMap framework and the process of customization for building specific model composition frameworks (see Figure 3.6).

The model–alignment language (see Section 3.2.2) is decomposed into four concerns :

- The *Mapping* concern deals with the representation of mappings between model elements (see Section 3.2.2.1).
- The *Filter* concern allows designers to propose filtering capabilities to mappings (see Section 3.2.2.1).
- The *Strategy* concern is the representation of the categories of interpretations formalized in Chapter 2 (see Section 3.2.2.2) and allows designers to associate a specific interpretation to a mapping.

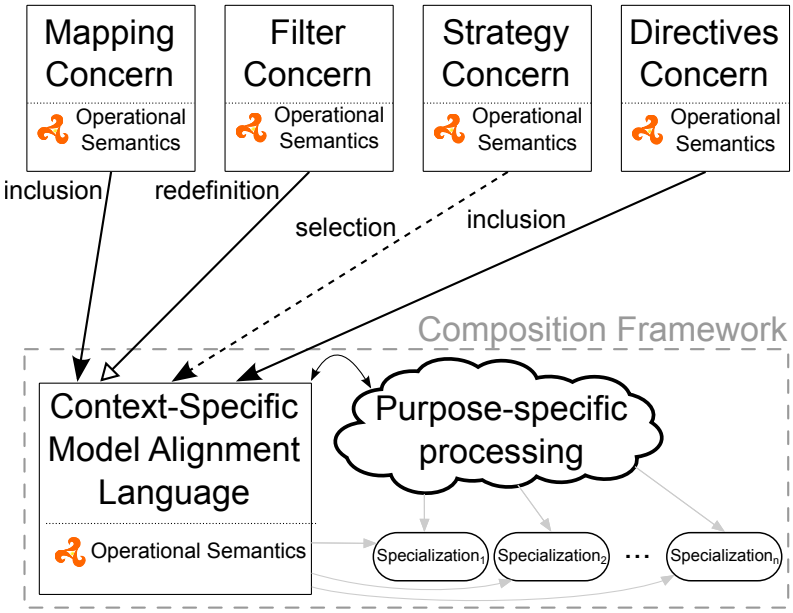


FIGURE 3.6 – Overview of the process of building a problem–specific model composition framework.

– The *Directives* concern represents a set of atomic operations that are applicable on model elements (see Section 3.2.2.2). Directives are parameters for strategies that allow designers to customize the model composition operation for their own needs.

These concerns contribute to the definition of the model–alignment modeling language as shown in Figure 3.6. We consider the Mapping concern and the Directives concern generic enough to be included in a model composition framework with no changes. Mappings between sets of model elements and parameters for interpretations are ready to use for any model composition framework.

Since every model composition expectations vary depending on the objective of the model composition operation, the model–alignment language proposed in Section 3.2.2 requires customization. Figure 3.6 illustrates the process of customization that building a specific model composition framework requires. Building a model alignment language is the first step of the definition of a new model composition framework. Customization allows designers to compose models within the context of a specific model composition intent. Since global intention is difficult to capture, we consider that designers should provide a purpose–specific processing (*i.e.*, algorithm).

The ModMap framework proposes operational semantics for every concept of the four concerns. Proposed operational semantics allows us to provide a default implementation of the various concepts. In the context of the Strategy concern, semantics provide interpretations with a default implementation that corresponds to the formalization proposed in Chapter 2. Each concern is weaved with its operational semantics (see Section 3.2.3) using the Kermeta language. Section 3.2.4 presents the abstract syn-

tax of the model-alignment language. Section 3.2.2 details the constructs of the four concerns of the model-alignment modeling language and Section 3.2.3 presents the operational semantics that are weaved to the various kinds of strategies.

3.2.2 A Language for (meta–)Model Alignment

This section details the constructs of the four concerns of the model alignment language and propose a concrete syntax to ease the definition of mappings between models or model elements. Figure 3.7 illustrates the ModMap meta–model. The ModMap meta–model describes a modeling language for the alignment of (meta–)models that is decomposed of the two main concepts of **`Mapping`** and **`Strategy`**. More details about these two concepts are provided in Section 3.2.2.1 and Section 3.2.2.2 respectively.

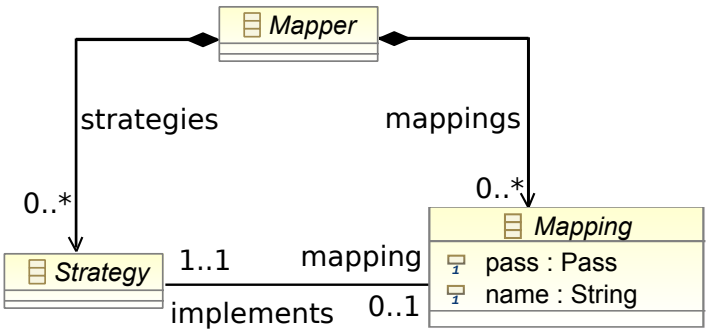


FIGURE 3.7 – The ModMap meta–model is composed of a *Mapper* root and the two main concepts of *Mapping* for creating relationships between model elements and *Strategy* for giving semantics to these relationships.

3.2.2.1 Mapping Concern

The mapping concern (see Figure 3.8) of the model composition language includes the concepts of **`Mapping`**, **`Role`** and **`Filter`**.

Mappings and Role

We propose four types of explicit mappings. Each type of mapping depends on the multiplicity of sources and targets model elements that we map with one another :

- **One2OneMapping** is a relationship between one source model element and one target element.
- **One2ManyMapping** is a relationship between one source model element and at least two target model elements.
- **Many2OneMapping** is a relationship between at least two source model elements and one target model elements.
- **Many2ManyMapping** is a relationship between at least two source model elements and at least two target model elements.

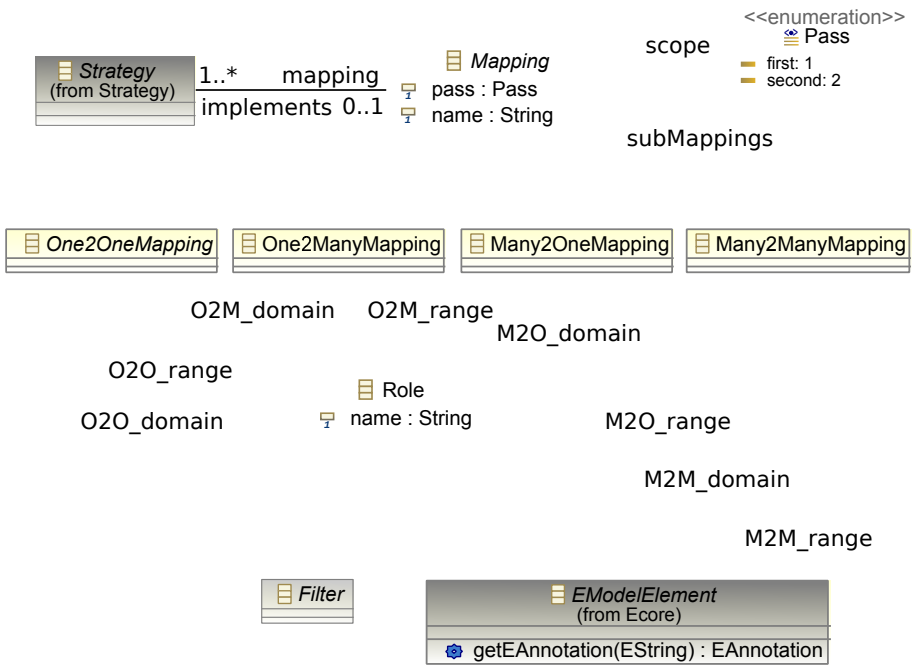


FIGURE 3.8 – Mapping Concern of the ModMap language. Dark–grayed element is a shortcut to the ECore meta–model and the light–grayed model element is the filter concern.

Mappings own relationships with model elements. Using the de-facto standard to design meta–models called Eclipse Modeling Framework (EMF) [1], mapping relationships are bound to the ECore modeling language. In ECore, the most general model element in the hierarchy of model elements is **`EModelElement`**. It allows to define a mapping between every model element from a ECore model.

The concept of **`Role`** acts as a proxy that references **`EModelElement`** and avoid binding a mapping to a **`EModelElement`** directly. This allows to reuse **`Roles`** for different mappings.

A mapping has a reflexive relationship which allows for declaring *sub mappings* that specialize the behavior of a parent mapping : this means that the semantics of the sub mapping encompasses the semantics of its parent mapping. When we traverse a mapping model for generating code or for analysis purposes, we visit the tree-based structure using for instance, a Visitor Design Pattern [GHJ+95]. Visiting a structure is decomposed into two passes, each one dealing with a specific concern. The first pass is dedicated to create the structure of the expected result and the second pass is invoked to correctly set non-containment references between objects. The *pass* attribute of a mapping indicates which pass a mapping refers to.

Default navigation strategy for the Visitor implementation is depth-first : we traverse the model by following the containment relationships. Each model element is

1. http ://www.eclipse.org/modeling/emf

responsible of calling the **`visit()`** method for its own children.

Filter Concern
In Figure 3.8, we propose the concept of **`Filter`** related to a **`Role`**. The **`Filter`** concept is abstract and allows the designer to propose filtering capabilities on the selection of model elements to consider in the mapping relation. In other words, although the definition of mappings between meta–types encompasses all instances of any meta–types involved, we allow experts to select instances of interest for a given situation and respectively discard instances that should not participate in a given mapping relationship. Experts may extend the **`Filter`** abstract concept to provide additional filtering capabilities.

3.2.2.2 Strategy Concern

Representing interpretations for mappings is decomposed in two sets of model elements : strategies and directives adapted from [FBF+08].

Strategies and Parameter
The strategy concern is the first set of model elements that deal with representing the interpretation of the mapping (see Figure 3.9). A strategy is the specification of a specific and predefined algorithm that corresponds to common model manipulation. Regarding the classification of interpretations presented in Chapter 1, we propose a strategy for each kind of interpretation. Some of the strategies need more details to capture a consistent definition of the interpretation. Additional details are provided by a **`Parameter`** which is linked to a set of directives that customize the application of the interpretation. The parameter of the InfoGapStrategy and of the AdHocStrategy is represented as a **`ModelingUnit`** element from Kermeta. A *ModelingUnit* is the root element of a Kermeta model. This means that a binding a **`ModelingUnit`** with a **`InfoGapStrategy`** or **`AdHocStrategy`**, we can provide respectively structural data and behavioral data as parameter to the interpretation.

Most strategies are applicable to any kind of elements at both the model and the meta–model levels of abstraction. The **`GenStrategy`** and the **`AggrStrategy`** involve changes on the hierarchy of types or on the hierarchy of containers respectively. These changes are only applicable on meta–models since they would break the conformance relationship that a model has with its meta–model.

The meaning of parameters for each strategy is such that :

- **GenStrategy** is about including an generalization relationship between a set of model elements. Each model element of the set identified by the parameter becomes a parent of each model element of the other set of models elements.
- **AggrStrategy** is about creating a *containment* relationship between model elements. Similarly to generalization, the parameter provides information about which model element becomes the container of the other set of model elements.
- **SimStrategy** means that we need to rename one set of elements to allow aligning models with one another. The parameter provide renaming directives.

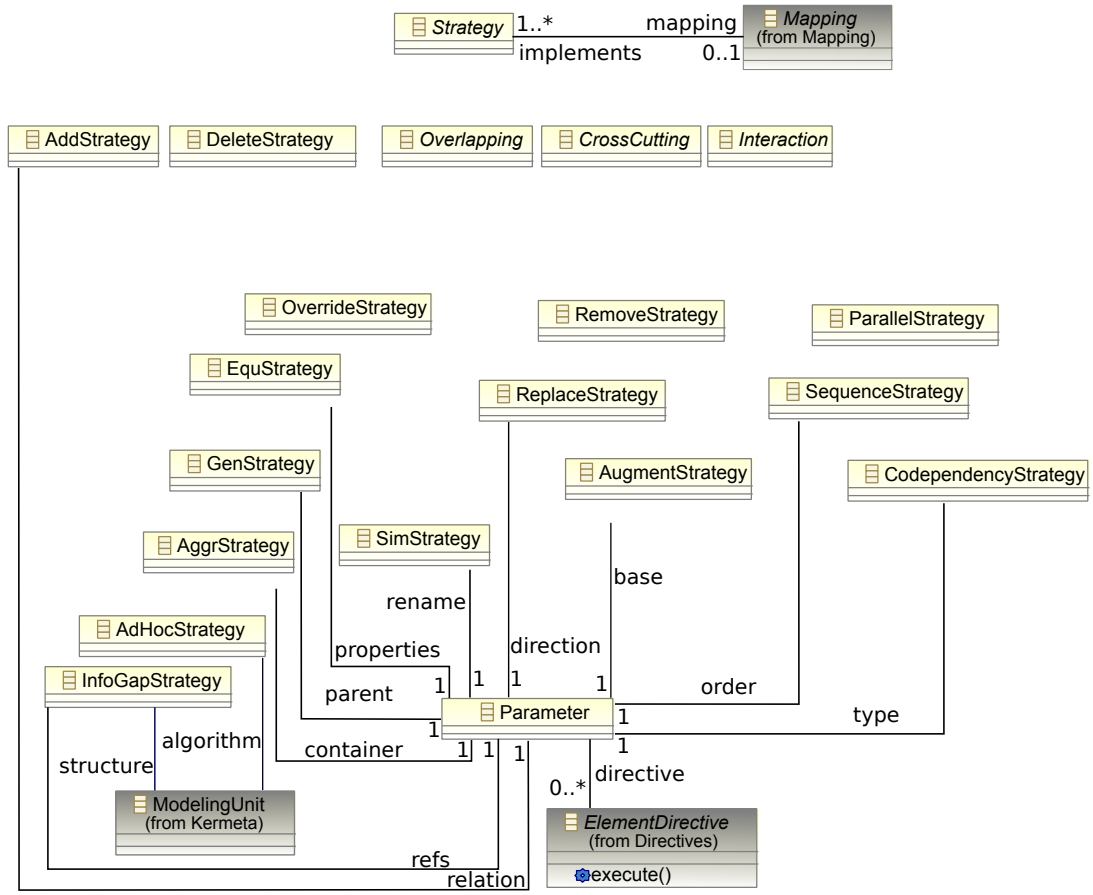


FIGURE 3.9 – The strategy concern contains interpretations proposed in Chapter 1. Most interpretations need additional parameters or structures to allow designers to capture the initial intention of the mapping. Dark–grayed concepts are shortcuts to Kermeta or concepts from the Mapping and Directives concerns.

- **ReplaceStrategy** indicates that one set of model elements is kept whereas other sets of model elements are discarded. The parameter indicates which set of model elements should be kept.
- **AugmentStrategy** is about keeping a set of model elements and adding data from other sets of model elements. The parameter identify which set of model elements should be considered as the *base* in the terminology of AOM.
- **SequenceStrategy** is intended to order the execution of a set of model elements. The parameters indicates how to order this execution.
- **CodependencyStrategy** is about ordering the execution of a set of model elements regarding to precise and somehow complex relationships. The parameter indicates which type of co-dependency should be implemented.

About including a well-formed structure, we consider the two following cases :

- **InfoGapStrategy** means that a partial structure is needed to relate the sets of

model elements. This structure is provided by a relationship to a ModelingUnit.
– **AdHocStrategy** provides an escape mechanism using a Turing-complete language to handle specific complex computations or transformations. The binding with a ModelingUnit allows for providing a specific algorithm provided by users. This algorithm is encompassed into a ModelingUnit that contains both structure and behavior.

Directives

Directives are a set of model elements that correspond to atomic operations on models (see Figure 3.10). The set of directives is voluntarily kept small to increase understanding and reusing.

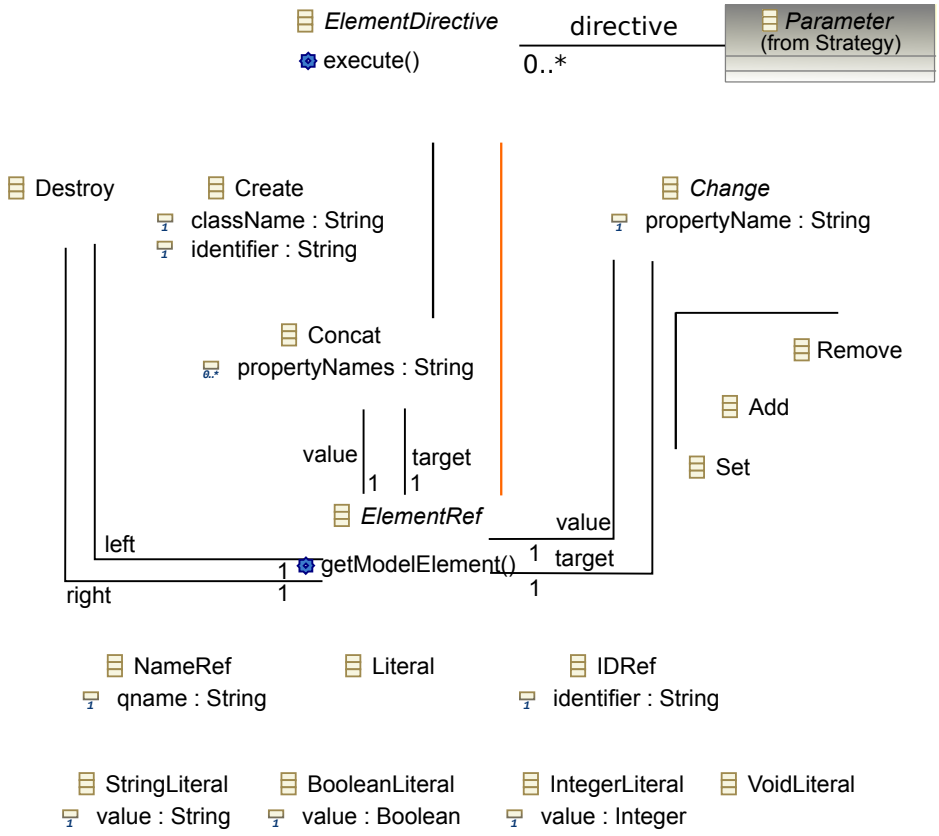


FIGURE 3.10 – Extension of Reddy *et al.*'s directives [RGF+06]. The **`Destroy`** and **`Concat`** concepts have been added to the original implementation. The **`Parameter`** concept is a shortcut to the Strategy concern.

Adapted from the directives [FBF+08 ; RGF+06] of the Kompose Tool [2], we reuse the following concepts with no changes :

2. http ://www.kermeta.org/mdk/kompose

- **ElementDirective** is the top-most model element from the directive concern. This abstract concepts encompasses the **`execute()`** methods which allows running a given directive.
- **Create** allows for creating new model elements with an **`identifier`** (*i.e.*, name of the variable to access to the model element) and a **`className`** (*i.e.*, the type of the new model element).
- **Change** is the abstract concept for any directives that induce modifying a model element. The **`propertyName`** property represents the name of the model element property to change. **`value`** owns the new value to set to the property **`propertyName`**. **`target`** indicates which model element the **`propertyName`** refers to.
  - **Remove** allows removing a property/value.
  - **Add** allows adding a property/value.
  - **Set** allows modifying an existing property/value.
- **ElementRef** is the abstract concept for referring to model elements or value.
- **NameRef** allows referring to a model element by its qualified name (**`qname`**).
- **Literal** allows referring to a literal value.
  - **StringLiteral** is a string value.
  - **BooleanLiteral** is a boolean value.
  - **IntegerLiteral** is an integer value.
  - **VoidLiteral** is the void value.
- **IDRef** allows referring to a specific identifier.

In addition to the original set of concepts, we propose two new concepts as follows :

- **Destroy** allows removing a model element from a model.
- **Concat** allows concatenating multiple strings and/or model elements properties of type String. **`propertyNames`** contains the list of strings to concatenate. **`value`** and **`target`** have the same semantics than **`value`** and **`target`** from the **`Change`** concept.

We added a new inheritance link between **`ElementDirective`** and **`ElementRef`** to allow reference a given model element within a larger scope.

3.2.3 A Tool for Building Model Composition Frameworks

This section proposes operational semantics for each construct of the concerns presented in Section 3.2.2. Operational semantics allow us to provide additional meaning for the constructs of the model-alignment modeling language and propose a default implementation for the set of interpretations.

Section 3.2.3.1 is a remainder on the methodology and techniques that we use to propose the operational semantics to the ModMap concerns. Following this methodology, Sections 3.2.3.2 to 3.2.3.4 proposes operational semantics for each ModMap concern.

3.2.3.1 Methodology and Techniques in Kermeta for Providing Operational Semantics

Operational semantics for the ModMap modeling language are provided using the Kermeta [MFJ05 ; JBF10] language. This section is a reminder about the implementation of the Visitor Design Pattern with aspects in Kermeta to "weave" operational semantics within the ModMap meta–model.

Aspects and Static Introduction

The ModMap concerns are embedded in an independent modeling language. We use Kermeta's open class mechanism to statically introduce new properties or new operations within the model elements of the meta–model of modeling language. This means that new properties and operations are statically type–checked against the complete underlying Kermeta model. We provide operational semantics for every ModMap model element using the static introduction supported by Kermeta, as illustrated in Listing 3.2.

```
// Reopening an existing class A
aspect class A{
  // Introducing new properties
  attribute name : String
  reference b : B
  // Introducing new operations
  operation myOperation(...) is do
    ...
  end
  // Overriding an existing operation
  method oldMethod(...) is do
    super
    ...
  end
}
```

Listing 3.2 – Static Introduction in Kermeta

The **aspect** keyword allows reopening an existing Kermeta class. In the scope of the aspect, attributes, references and operations are easily added using the Kermeta standard concrete syntax.

Visitor Design Pattern in Kermeta

As any GPL–like programming language, Kermeta allows implementing the Visitor Design Pattern [GHJ+95] to scan a tree–based representation such as a model. Using aspects and static introduction, developers define their abstract and concrete visitors with no changes to the (meta–)model on which it applies.

The implementation of the Visitor Design Pattern is however cumbersome : it requires to create an **accept()** method in every model element ; it requires an **AbstractVisitor** to declare **visit()** methods for every model element ; it requires to implement **visit()** methods for any concrete **Visitor**.

Inspired from David Lorenz approach [Lor98] for using aspects for the implementation of the the Visitor Design Pattern, we use aspects to capture traceability information. Listing 3.3 shows an example of Visitor using aspects.

```
// Integrating Visitor into an existing class
aspect class A {
  /* Adding a reference for the purpose of traceability
   * and references the result of the Visitor
   */
  reference output: Object
  // The visit operation
  operation visit(...): Void is do
    ...
  end
}

// Using the Visitor to scan a structure
class Visitor{
  operation main{...}: Void is do
    var a : A init A.new
    ...
    // starts visiting
    a.visit()
    // retrieves result
    a.output
    ...
  end
}
```

Listing 3.3 – Implementation of a Visitor using aspects in Kermeta

Using aspects, we reopen an existing class in which we add a new reference and a new operation [3]. The operation performs a specific computation and stores the result into the reference. In any Kermeta class, we can start visiting a structure using the **`visit`** method and access the result by a call to the **`output`** reference of an object. **`output`** plays two roles : the reference stores the result of the **`visit()`** operation, and the reference is a traceability link between the current object and the result object.

In the following sections, we propose default operational semantics for the ModMap model elements using aspects and the Visitor Design Pattern. The purpose of proposing default operational semantics is to provide behavior to the structural concepts of the ModMap meta–model. Providing behavior allows us to capitalize part of the purpose of various model composition operators. Semantics allow designers to focus on proposing purpose–specific processing that covers what remains to build a specific model composition operation.

3.2.3.2 Operational Semantics for the Mapping Concern

Operational semantics for the mapping concern refers to the way the mapping model structure is to be scanned. The default method of scanning is naive : we scan every mapping one by one, in the order of creation. Since scanning may have an

3. The reference name and the operation name have no impact on the execution of the visitor.

impact on the ultimate purpose of the model composition, the naive method may be specialized if necessary. Scan is available through the implementation of the Visitor Design Pattern (see Listing 3.4 and Listing 3.5), using the methodology that we detailed in Section 3.2.3.1.

```
/* Mapper is the root object of the ModMap model
 * Method visit scans mappings in the
 * order of their creation.
 * Output is a ModelingUnit
 */
aspect class Mapper {
  reference output : ModelingUnit

  operation visit() is do
    output := ModelingUnit.new
    // scans mappings and calls Visitor on each
    self.mappings.each{ m |
      m.visit()
      //missing output assignment
    }
  end
}
```

Listing 3.4 – Operational Semantics for the Mapper model element from the mapping concern.

The operational semantics of the **Mapper** model element creates a Kermeta **ModelingUnit** that represent the model of composition. The **visit()** method scans mappings and subsequently modify the **ModelingUnit**.

Scanning mappings is a three steps process (see Listings 3.5 and 3.6) : *(i)*we scan nested mappings using the **subMappings** property of a **Mapping** ; *(ii)*we scan roles associated to a mapping ; *(iii)*we build an algorithm from the operational semantics of associated strategies.

About roles, we expect to build a specific Kermeta Class for each role to represent the related **EModelElement**. This Kermeta class is used to define a specific composition method. The creation of a class (**buildClassFromEModelElement** method) manipulates Kermeta concepts and uses a filter if any to allow selecting relevant model elements only. Inputs about filtering are provided by end-users.

```
/* Scanning nested mappings of a mapping */
aspect class Mapping {
  reference output : Set<Class>
  operation visit() is do
    self.subMappings.each{ m |
      m.visit()
      output.add(m.output)
    }
  end
}
/* Scanning domain and range roles */
aspect class One2OneMapping {
  method visit() is do
    super
    // Creates output classes
    var domain_class : Class
    var range_class : Class
    // Scans associated roles
    O2O_domain.visit()
    O2O_range.visit()
    var domain_block : Block
    var range_block : Block
    // Scans related strategies and builds blocks of statements
    self.implements.each{ s |
      s.visit(domain,range)
      domain_block := s.output_domain()
      range_block := s.output_range()
    }
    domain_class := O2O_domain.output
    range_class := O2O_range.output
    // Adds block of statements to the composition method
    var domain_op : Operation init domain_class.getOperation("myOp")
    var range_op : Operation init range_class.getOperation("myOp")
    domain_op.statement.add(domain_block)
    range_op.statement.add(range_block)
    output.add(domain_class)
    output.add(range_class)
  end
}
```

Listing 3.5 – Operational Semantics for the Mapping model elements from the mapping concern.

```
/* Scanning a Role and build a Kermeta representation of the target
 * EModelElement
 */
aspect class Role {
  reference output : Class
  operation visit() is do
    output := buildClassFromEModelElement(self.element, filter)
  end
}
```

Listing 3.6 – Operational Semantics for the Role model element from the mapping concern.

3.2.3.3 Operational Semantics for the Strategies Concern

This section presents the operational semantics of the strategy concern of the ModMap meta–model. These operational semantics relies on the directives concern that is presented in Section 3.2.3.4.

Listing 3.7 illustrates the definition of the **`visit()`** method for strategies. Every model element that inherits from Strategy provides an implementation of the **`visit()`** method.

The default semantics of the **`visit()`** method calls an operation called **`compose()`** that embeds the specific processing of the model composition operation. The **`compose()`** operation produces a set of statements that achieve the specific model composition operation on the objects *A* and *B*. This set of statements is encapsulated into a Kermeta **`Block`** of statements.

```
aspect class Strategy {
  reference output : Set<Block>
  operation visit(A : Set<Object>, B : Set<Object>) is do
    /* compose() is an operation that includes the process specific to the
       purpose of the model composition */
    var b : Block init compose(A,B)
    output.add(b)
  end
}
```

Listing 3.7 – Operational Semantics for the Strategy model element from the strategy concern.

The **`Parameter`** model element allows providing parameters for a strategy if necessary. These parameters are written using directives. Scanning a parameter thus subsequently scans the set of directives associated with the parameter (see Listing 3.8).

```
aspect class Parameter {
  reference output : Set<Object>
  operation visit(A : Set<Object>, B : Set<Object>) is do
    directive.each{d|
      d.visit(A,B)
      output.add(d.output)
    }
  end
}
```

Listing 3.8 – Operational Semantics for the Parameter model element from the strategy concern.

Add and Delete Strategies

Add Strategy The **`visit()`** method of the Add strategy (see Listing 3.9) calls the **`visit()`** method of its parameter. If a parameter has been provided, we create a new relation (**`Property`** object in Kermeta) and add this new relation to the set of properties of the object *A* before calling the **`super`** operation from Strategy.

```
aspect class AddStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.relation.visit(A,B)
    var p : Property init self.relation.output
    if p != void then
      A.ownedAttribute.add(p)
    end
    super
  end
}
```

Listing 3.9 – Operational Semantics for the Add Strategy model element from the strategy concern.

Delete Strategy The Delete Strategy (see Listing 3.10) has no additional parameter. It calls the **`remove()`** operation on the collection of objects from A with objects from B as a parameter before calling the **`super`** operation from Strategy.

```
aspect class DeleteStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    A.removeAll(B)
    super
  end
}
```

Listing 3.10 – Operational Semantics for the Delete Strategy model element from the strategy concern.

Overlapping Strategies

Equivalence Strategy The Equivalence Strategy (see Listing 3.11) relies on a set of parameters (*properties*). The **`visit()`** method calls the Visitor on the properties parameter and then call the **`equals()`** operation. The **`equals()`** operation stores which properties proposed by the user should be considered as equivalent for each model element. This information supports the composition process to identify equivalent model elements.

```
aspect class EquStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.properties.visit(A,B)
    equals(A,B,self.properties.output)
    super
  end
}
```

Listing 3.11 – Operational Semantics for the Equivalence Strategy model element from the strategy concern.

Similarity Strategy The Similarity Strategy (see Listing 3.12) relies on a set of parameters (*rename* property). The **`visit()`** method calls the Visitor on its parameter and then call the **`align()`** operation. The **`align()`** operation align objects A and B with regard to the set of directives provided by the user.

```
aspect class SimStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.rename.visit(A,B)
    align(A,B,self.rename.output)
    super
  end
}
```

Listing 3.12 – Operational Semantics for the Similarity Strategy model element from the strategy concern.

Generalization Strategy The Generalization Strategy (see Listing 3.13) only applies on the meta–level of abstraction (*i.e.*, the meta–model) of a model, since it modifies the tree of hierarchy of a set of model elements. The **`parent`** parameter indicates which set of model elements should become the parent of the other. If A or B are equals to the parameter, the set of supertypes of A or B is respectively augmented with B or A.

```
aspect class GenStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.parent.visit(A,B)
    if A.equals(self.parent.output) then
      B.superType.add(A)
    else
      A.superType.add(B)
    end
    super
  end
}
```

Listing 3.13 – Operational Semantics for the Generalization Strategy model element from the strategy concern.

Aggregration Strategy The Aggregation Strategy (see Listing 3.14) only applies on meta–models since it modifies the inherent structure of a model by adding a containment relationship. The **`container`** parameter indicates which set of model elements should become the container of the other set. From this information, we create a new relationship with its *containment* property equals to *true*. This new property is then added to object A or B, depending on the container parameter.

```
aspect class AggrStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.container.visit(A,B)
    var p : Property init self.container.output
    if A.equals(self.container.target) then
      A.ownedAttribute.add(p)
    else
      B.ownedAttribute.add(p)
    end
    super
  end
}
```

Listing 3.14 – Operational Semantics for the Aggregation Strategy model element from the strategy concern.

Override Strategy The Override Strategy (see Listing 3.15) traverse the structure of A to detect model elements to override. Detected model elements are added to a new set of objects **new_a** that contains the original model elements of A without those replaced by the objects from B.

```
aspect class OverrideStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    var new_A : Set<Object> init Set<Object>.new
    A.each{o|
      if not B.contains(o) then
        new_A.add(o)
      else
        new_A.add(B.get(o))
      end
    }
    A.clear
    A.addAll(new_A)
    super
  end
}
```

Listing 3.15 – Operational Semantics for the Override Strategy model element from the strategy concern.

Information Gap Strategy The Information Gap Strategy (see Listing 3.16) expects to get a set of relationships from the parameter. Visiting the parameter produces a set of properties identified by their expected containing model element. We browse this map of properties and subsequently add them to either A or B, depending on the parameters. The reason why we do not manipulate the *structure* relationship and consequently the **ModelingUnit** attached to it is that we expect the **refs** parameter to provide relationships that already point to the structure to include.

Ad Hoc Strategy The Ad Hoc Strategy (see Listing 3.17) propose an escape mechanism that allows arbitrary complex computation on the model elements that are mapped with one another. We propose to encapsulate this complex computation in a Kermeta **ModelingUnit** which represent the algorithm of the process. This algorithm is

```
aspect class InfoGapStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    var m : ModelingUnit init self.structure
    self.refs.visit(A,B)
    var properties : Hashtable<Object,Property> init self.refs.output
    properties.keySet.each{o|
      if o.equals(A) then
        A.add(properties.get(o))
      else
        if o.equals(B) then
          B.ownedAttribute.add(properties.get(o))
        else
          o.ownedAttribute.add(properties.get(o))
        end
      end
    }
    super
  end
}
```

Listing 3.16 – Operational Semantics for the Information Gap Strategy model element from the strategy concern.

added to the current **`Block`** of statements of the strategy as preliminary process before the execution of the **`compose()`** operation.

```
aspect class AdHocStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    //complex user-defined computation
    var m : ModelingUnit init self.algorithm
    output.add(ModelingUnit)
    super
  end
}
```

Listing 3.17 – Operational Semantics for the Ad Hoc Strategy model element from the strategy concern.

Cross–Cutting Strategies

Replace Strategy The Replace Strategy (see Listing 3.18) replaces all model models from one set with model elements from another set. According to the **`direction`** parameter that states which set of model elements should be replaced, we traverse the sets of model elements and create a new set which contains replaced model elements.

```
aspect class ReplaceStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    var new_set : Set<Object> init Set<Object>.new
    self.direction.visit(A,B)
    if self.direction.output.equals(A) then
      A.each{o|
        if not B.contains(o) then
        new_set.add(o)
      else
        new_set.add(B.get(o))
      end
      }
      A.clear()
      A.addAll(new_set)
    else
      B.each{o|
        if not A.contains(o) then
        new_set.add(o)
      else
        new_set.add(A.get(o))
      end
      }
      B.clear()
      B.addAll(new_set)
    end
    super
  end
}
```

Listing 3.18 – Operational Semantics for the Replace Strategy model element from the strategy concern.

Augment Strategy The Augment Strategy (see Listing 3.19) rely on a *base* set of model elements which is identified by the **base** parameter. For each model element of the base, if two elements are found equal, we make the union of the sets of their properties to ultimately add it to the model elements of the base.

Remove Strategy The Remove Strategy (see Listing 3.20) is the opposite of the augment strategy. For each pair of equal model elements, we check if they own a common property. If found, we remove the common property.

```
aspect class AugmentStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.base.visit(A,B)
    if A.equals(self.base.output) then
      B.each{o|
        A.each{o2|
          if o.equals(o2) then
            o.ownedAttribute.each{aA|
              o2.ownedAttribute.add(aA)
            }
          end
        }
      }
    else
      A.each{o|
        B.each{o2|
          if o.equals(o2) then
            o.ownedAttribute.each{aA|
              o2.ownedAttribute.add(aA)
            }
          end
        }
      }
    end
    super
  end
}
```

Listing 3.19 – Operational Semantics for the Augment Strategy model element from the strategy concern.

```
aspect class RemoveStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    B.each{o|
      A.each{o2|
        if o.equals(o2) then
          o.ownedAttribute.each{aA|
            o2.ownedAttribute.remove(aA)
          }
        end
      }
    }
    super
  end
}
```

Listing 3.20 – Operational Semantics for the Remove Strategy model element from the strategy concern.

Interaction Strategies

Sequence Strategy The Sequence Strategy (see Listing 3.21) orders the execution of A and B. The call of the **`visit()`** operation on the **`order`** parameter identifies which model element should be executed first. The **`execute()`** operation is then called depending on this **`order`** parameter.

```
aspect class SequenceStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.order.visit(A,B)
    if self.order.output.equals(A) then
      execute(A)
      execute(B)
    else
      execute(B)
      execute(A)
    end
    super
  end
}
```

Listing 3.21 – Operational Semantics for the Sequence Strategy model element from the strategy concern.

Parallel Strategy The Parallel Strategy (see Listing 3.22) rely on the **`concurr_exec()`** operation which implements the concurrent execution of the model elements.

```
aspect class ParallelStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    concurr_exec(A,B)
    super
  end
}
```

Listing 3.22 – Operational Semantics for the Parallel Strategy model element from the strategy concern.

Codependency Strategy The Codependency Strategy (see Listing 3.23) proposes a naive implementation of the *start–to–start*, *start–to–finish*, *finish–to–start*, and *finish–to–finish* codependency relationships. According to the **`type`** parameter which indicates which kind of codependency relationships to take into account, we use a couple of boolean values to indicate if the execution of a model element is **`started`** or **`finished`** and to allow an execution to start or to end with respectively **`allowedToStart`** and **`allowedToEnd`** boolean values.

```
aspect class CodependencyStrategy {
  method visit(A : Set<Object>, B : Set<Object>) is do
    self.type.visit(A,B)
    // Start-to-Start
    if self.type.output.equals(s2s) then
      if A.started then B.allowedToStart := true end
    else
      // Start-to-Finish
      if self.type.output.equals(s2f) then
        if A.started then B.allowedToEnd := true end
      else
        // Finish-to-Start
        if self.type.output.equals(f2s) then
          if A.ended then B.allowedToStart := true end
        else
          // Finish-to-Finish
          if A.ended then B.allowedToEnd := true end
        end
      end
    end
    super
  end
}
```

Listing 3.23 – Operational Semantics for the Codependency Strategy model element from the strategy concern.

3.2.3.4 Operational Semantics for Directives

In this section, we distinguish operation directives and reference directives. We presents the operational semantics for **`ElementDirective`** and its specializations that expect changes (**`ElementDirective`**)and detail **`ElementRef`** directives that allow referencing existing model elements or literals.

Directives Operational semantics of the ElementDirective directive declare an abstract method **`visit()`** which is overridden by the ElementDirective children. Listing 3.24 shows the operational semantics for the ElementDirective object and for **`Change`**, one of its children. The **`visit()`** method of Change looks for the target model element provided by the *propertyName* property and the target *value* property in both A and B. Both objects are added to the directive output.

References to Model Elements References to model elements are provided by the abstract concept **`ElementRef`**. Listing 3.25 illustrates the operational semantics for the **`NameRef`** concept and the **`Literal`** concept which respectively references an object or a constant value. The qualified name of the object to find (*qname* property) is looked up in both A and B and added to the output if found. If the referenced element is a constant value, output is assigned to its value.

```
aspect class ElementDirective {
  operation visit(A : Set<Object>, B : Set<Object>) is abstract
}

aspect class Change {
  method visit(A : Set<Object>, B : Set<Object>) is do
    var o : Object init lookupObject(self.propertyName,A)
    if o == void then
      o := lookupObject(self.propertyName,B)
    end
    self.value.visit(A,B)
    output.add(o)
    output.add(self.value.output)
  end
}
```

Listing 3.24 – Operational Semantics for ElementDirective and its children from directives.

```
aspect class NameRef {
  method visit(A : Set<Object>, B : Set<Object>) is do
    var o : Object init lookupObject(self.qname,A)
    if o == void then
      o := lookupObject(self.qname,B)
    end
    output.add(o)
  end
}

aspect class Literal {
  method visit(A : Set<Object>, B : Set<Object>) is do
    output := self.value
  end
}
```

Listing 3.25 – Operational Semantics for ElementRef and its children from directives.

3.2.4 ModMap Concrete Syntax

Providing a framework for building model composition operators may be difficult to manipulate for designers. To this purpose, we propose a concrete syntax for ModMap to ease the definition of mapping specifications, providing designers with a graphical language. The constructs of the concrete syntax reflect the constructs of the model-alignment modeling language presented in Section 3.2.2. The graphical representation is an adaptation and an extension of the formalism proposed by Hausmann *et al.* [HK03]. Hausmann *et al.*'s formalism includes elements to express "consistency between models" or in other words "conditions under which two models are compatible".

We kept the following definition as a basis for the concrete syntax :

- A white diamond indicates a mapping definition and has a name.
- A given model element or a set of model elements is linked to a mapping definition with a dotted line.

– If a mapping definition involves multiple sources or multiple targets, a black circle is put between the mapping description and the sets of model elements involved.

We propose to extend Hausmann et al.'s formalism as follows :

– We explicitly separated mapping descriptions into four types. Each type depends on the multiplicity of sources models elements and targets model elements that we want to map with one another.
– We allow the creation of a mapping description between any model element such as classes, attributes, and associations between model elements.
– We propose that each mapping description is bound to a set of strategies (see Section 3.2.2.2).
– Each strategy is represented by a colored circle and included into the white diamond of a mapping description.
– If necessary, parameters for strategies are provided in orange boxes and linked to their strategy by a straight gray line.
– For **`Ad Hoc`** and **`Information Gap`** strategies, a red box contains respectively the user–provided structure or algorithm necessary for the model composition processing.

Figure 3.11 illustrates the concrete syntax of the model-alignment language on the running example. We use a **`One2OneMapping`** to define an equivalence relationship between **`Account`** from the Bank model and **`Account`** from the BLP model. The other **`One2OneMapping`** defines an equivalence relationship between **`BankUser`** from the Bank model and **`BankUser`** from the BLP model.

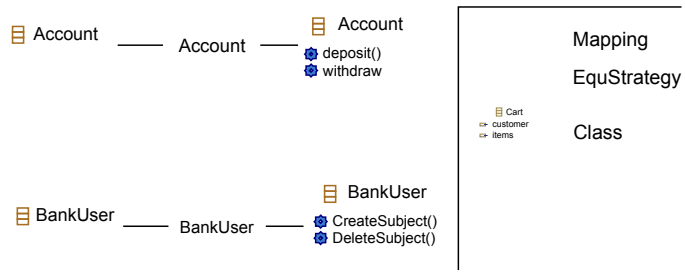


FIGURE 3.11 – Model of mappings between the Bank model and the BLP model

Additional examples of the concrete syntax in action can be found in Chapter 4.

3.3 Conclusion

We have presented in this chapter the ModMap framework that allows designers to build model composition frameworks that are specific to a given purpose. We proposed a model-alignment language that covers the definition of mappings between a set of models or a set of models elements. This model-alignment language can be customized for a given purpose by selecting a set of interpretation that are relevant for a specific purpose. Additional purpose– or domain–specific characteristics should be proposed

by designers to finalize the definition of a model composition framework for a specific operation on models.

In the next chapter, we demonstrate the application of the ModMap framework on three case studies that target three different operations on a set of models.

Chapitre 4

Validation and Application

This section presents three use cases that act as a validation of the theoretical framework for unifying model composition approaches, presented in Chapter 2, and of the ModMap language presented in Chapter 3.

Section 4.1 demonstrates the use of the theoretical framework to unify four existing model merging techniques. With this experiment, we expect to validate the adequateness of the theoretical framework *(i)*to **`compare`** existing model composition approaches and *(ii)*to **`propose a unique kernel`** for model composition that is independent from the context.

Section 4.2 demonstrates the use of the theoretical framework on an industrial case study from the MOPCOM–I project to integrate legacy APIs. We collaborate with industrial partners from Technicolor [1] to automate the integration of legacy API for the configuration and management of heterogeneous video and broadcasting equipments. This validates both the **applicability of the theoretical framework to other management activities** and the **efficiency of using the ModMap framework** in action.

Section 4.3 extends the use of ModMap and the unified theoretical framework to the challenges of model synchronization in a context of Service–Oriented Architecture (SOA). This allows **`extending the scope of the theoretical framework to encompass various model management activities`** beyond model merging and model integration.

4.1 Generalizing Model Merging

Model merging is used to compose different concerns in a large model. Even though the purpose of model merging usually remains constant, most model merging operators are custom-built for specific languages. In this case study, we propose a unified framework of model merging that distills the body of knowledge embedded in four existing model merging operations. The unified framework helps both comparing existing operations on a common basis and improving code reuse to build new model merging operations.

1. http ://www.technicolor.com/en/hi/technology/research-and-innovation-centers/rennes

4.1.1 Existing Tools for Model Merging

We choose four model merging techniques that handle a pair of homogeneous models to illustrate the current approach. We briefly recap the main concepts for each technique in Sections 4.1.1.1 to 4.1.1.4.

Description of the four techniques is organized as follows :

- **Inputs** details if a technique is model or meta–model specific and how many models are involved in a model merging operation.
- **Match** presents the implicit or explicit matching mechanism proposed by a technique to detect model elements to merge.
- **Merge** gives information about the merging process for elements that match and element that do not.
- **Outputs** gives details about the output model, its type and any specific characteristic.
- **Other** lists any other specific processing that supports or enhances the merging process to produce a model that meets the user expectations.

Using this structure, we capture the internal mechanisms of each technique and expect to identify commonalties that would ease the definition of a generic unified core for model merging.

4.1.1.1 UML Package Merge

The Unified Modeling Language (UML) is a standard proposed by the Object Management Group (OMG). The UML2 modeling language supports modular construction of models through the notions of package and ppackage merge. As a tool for improving modularity, UML2 supports a new directed relationship between packages named "Package Merge" [OMG10b, §11.9.3]. Package Merge allows for extending an existing package by merging the contents of another package.

- **Inputs** are two UML Class Diagrams that are either physically independent or defined under two different namespaces. The technique is applicable to the whole UML meta–model if adequate extensions are provided to handle the various kinds of UML types.
- **Match** is an implicit mechanism that checks names and types (*i.e.*, instance of the same element in the meta-model).
- **Merge** applies on a predefined set of types and recursively among packages and their nested children. The set of predefined types may be extended for supporting merging among specific types.
  - Matchings elements are merged w.r.t. a set of transformation rules [OMG10b, §11.9.3, p.165–168] that details the navigation across the structure of elements and constraints to satisfy.
  - Non matching elements are deep copied.
- **Outputs** is a new UML Class Diagram that contains the output of the merge operation : matching elements are merged with one another and non matching elements are copied from the source UML models.

Fig. 4.1 illustrates an example of UML PackageMerge in which the contents of the package *BasicEmployees* should be merged with the content of package *EmployeeLocation*. On this specific example, only one match is found : The UML class *Employee* in *BasicEmployees* has a name that equals the name of the UML class *Employee* in *EmployeeLocation*. The resulting UML class diagram thus contains a UML class *Employee* that contains the properties (*i.e.*, attributes and references) of both source classes. No other match is detected between these packages, thus other UML classes are just copied into the resulting UML class diagram.

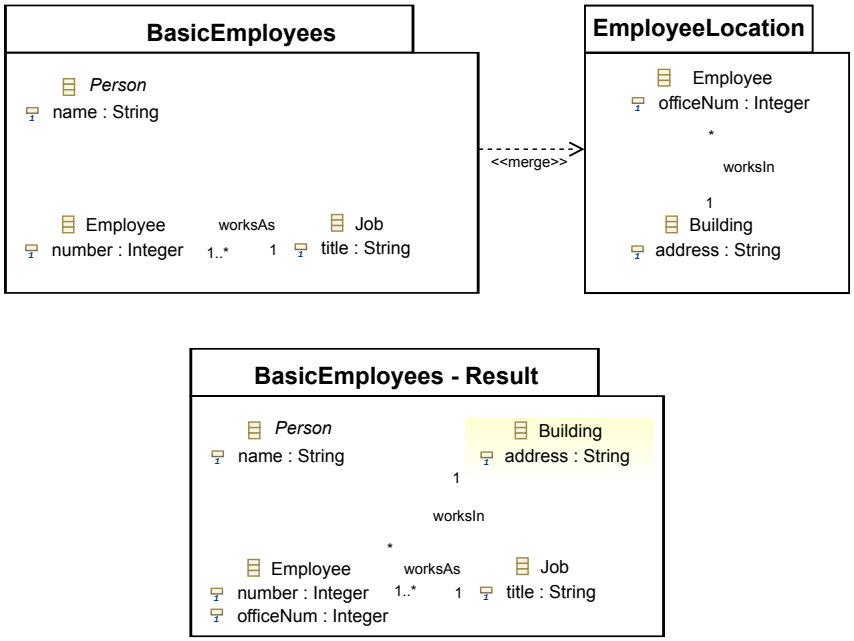


FIGURE 4.1 – A package merge example adapted from [ZDD06]

4.1.1.2 Kompose : A Generic Model Composition Tool

A detailed description of Kompose is available in Section 1.1.4.1. As a reminder, Kompose [2] is a merging tool for multi–view design [FFR+07] (related to Aspect–Oriented Modeling). Kompose merges homogeneous pairs of models that conform to various kinds of meta–models.

- **Inputs** are two homogeneous models that conform to the same meta–model that itself conforms to the Meta Object Facility (MOF) meta–model. One of the two models is considered as the base model and the other as an aspect, similarly to what exists in Aspect–Oriented Modeling.
- **Match** is an explicit mechanism based on signatures. Signatures define a set of properties which are used to detect the equivalence between pairs of model

2. http ://www.kermeta.org/mdk/kompose

elements that have the same type (*e.g.*, pairs of classes or pairs of operations). Default signature checks name.
- **Merge** applies on all types and recursively among containers and their nested children. Using reflexivity and introspection to access properties allows defining a generic merge algorithm to support merging of different models. The model elements to merge are all referenced in a global set of model elements which is sequentially browsed to detect matches and merge model elements that match.
  - Model elements which signatures match are merged into a single model element
  - Model elements with no match are deep copied in the resulting model
- **Outputs** is the modification of the base model that contains the output of the merge operation as described above.
- **Other** facilities are proposed by Kompose to change input and output models prior to and post to the match and the merge mechanisms. In addition, a conflict resolver is provided to meet specific user expectations regarding the merge of values.

Fig. 4.2 and Fig. 4.3 are respectively the base model (BLP) and the aspect model (Bank) from [FFR+07]. In this example, the considered models are class diagrams that conform to the UML meta-model. Listing 4.1 presents the pseudocode of the default signature. Using the default signature, we detect that the classes *BankUser* and *Account* from the *Bank* model matches classes *BankUser* and *Account* from the *BLP* model respectively. Fig. 4.4 shows the new version of the *BLP* model after merging. Matching classes have been merged into a single class and properties from these classes have been merged into a single property. Non matching classes and properties from the input models have been deep copied. As described in [FFR+07], the two operations *transfer()* and *withdraw()* from the *Controller* class of the *Bank* model have been removed from the merged model with post–directives to meet users requirements.

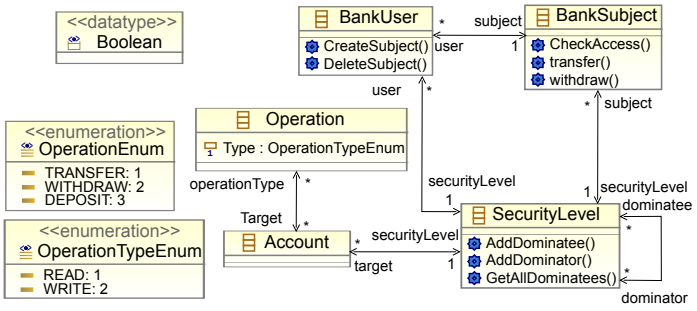


FIGURE 4.2 – An example of the blp model adapted from [FFR+07]

4.1.1.3 Match and Merge of Statechart Specifications

Nejati *et al.* [NSC+07] have defined a merge operator for statecharts. This merge operator produces a new statechart that contains states, transitions, actions, events and guards of the input statecharts. The merge operator preserves the behavioral properties

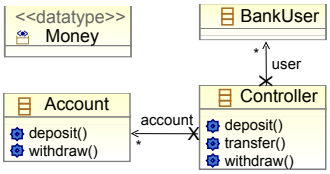


FIGURE 4.3 – An example of the bank model adapted from [FFR+07]

```
function equals(obj1: Object,obj2 : Object) : Boolean is
    return obj1.name == obj2.name
end
```

Listing 4.1 – Pseudo code of the Kompose default signature.

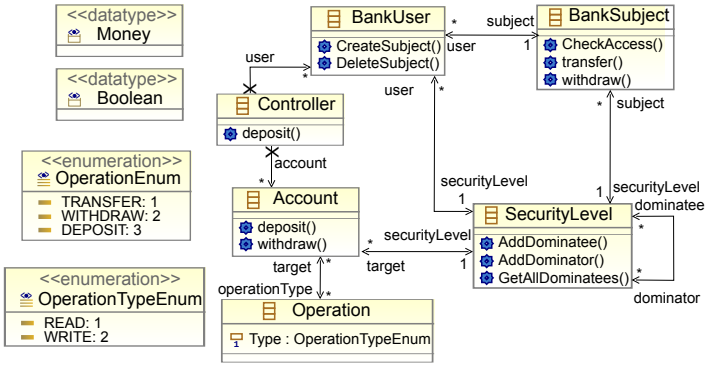


FIGURE 4.4 – Result of the composition of the Bank model and the BLP model from [FFR+07]

of the input models, distinguishes between shared and non-shared behaviors of the input models, and respects the hierarchical structure and parallelism of the input models.

- **Inputs** are two statecharts, for instance that conform to the UML State Machine meta-model.
- **Match** is an explicit mechanism that uses heuristics to propose correspondence relationships between states from the two statecharts. Correspondence relationships should be reviewed by domain experts to add missing correspondences and to remove false positives. The match mechanism is hybrid and combines both static matching and behavioral matching.
  - Static matching combines similarity degrees between states. Similarity degrees are computed from typographic and linguistic data (i.e. state names) and the position of a state in the hierarchical structure of both statecharts.
  - Behavioral matching generates similarity degrees between states based on their behavioral semantics, computing values of forward- and backward-bisimilarity.

Each matching is defined as a function assigning a normalized value to every pair of states. The closer a degree is to 1, the more similar two states are. Aggregating the static and behavioral heuristics to generate the overall similarity degrees between states and given a similarity threshold, they can determine a correspondence relation over the states of the input models.

- **Merge** constructs a model that contains shared behaviors of the input models as normal behaviors and non–shared behaviors as variabilities. Variabilities are represented using parametrization : non–shared transitions are guarded by conditions denoting the transitions' origins. Non-shared states are included in the output model without any provisions.
  - Shared states and transitions are identified as follows :
    - A state is shared if it is related to some state by a correspondence relation and is non–shared otherwise.
    - A transition t is shared : if the transition source x and target y states are shared (*i.e.*, a correspondence relation relates x with an x' and y with y') ; if there exists a transition t' between x' and y' whose event, whose condition and whose priority equals those of the transition t ; if the action of t' either equals action of t or if the two actions are independent. A pair of actions is independent if the execution order results in the same system behavior. A transition is non–shared otherwise.
  - Shared states are merged into a single state in the output model with their names concatenated.
  - Non–shared states are copied in the output model
  - Shared transitions are copied in the output model States names are and a new state is added to the resulting statechart specification
  - Shared transitions are copied to the resulting statechart specification with their event, their condition,their action and their priority unchanged.
  - Non–shared transitions are copied to the resulting statechart specification with their event, their action and their priority unchanged and with the condition concatenated with the name of the original UML statechart that the transition comes from.
- **Output** is a new structurally sound statechart. The output statechart contains both shared and non -shared behaviors.
- **Other** merging rules are available for computing with the hierarchical structure for parallel states and super states.

Fig. 4.5 shows the two input statecharts and the merged statechart as presented in [NSC+07]. The following equation 4.1 is the set of correspondence relations after revision by domain experts :

$$(s_0, t_0), (s_4, t_4), (s_2, t_1), (s_5, t_5), (s_3, t_2), (s_6, t_6), (s_3, t_3), (s_7, t_7) \tag{4.1}$$

Among the states of the *Call Logger Basic* and *Call Logger Voicemail*, only s_1 and s_8 are non–shared and thus copied to s_0, t_0 with no changes. Every other state has its name concatenated with the states it corresponds to. Boldface conditions capture the origins

of the respective transitions. For instance, the transition from (s_2,t_2) to (s_4,t_4) annotated with the condition **[ID=basic]** indicates a variable behavior that is applicable only for clients subscribing to basic. The definition of shared transitions is conservative in the sens that it requires such transitions to have identical events, conditions and priorities in both input models. While this is sound to ensure that behavior is preserved, it may results in redundant transitions. For example, the transitions from (s_2,t_1) to (s_3,t_2) and to (s_3,t_3) fire actions *callee = subscriber* and *callee = participant* respectively. Since the value of *callee* in state (s_3,t_3) equals *participant* and equals *subscriber* in (s_3,t_2), we can replace *callee* by *subscriber* or *participant* on transitions and merge redundant transitions.

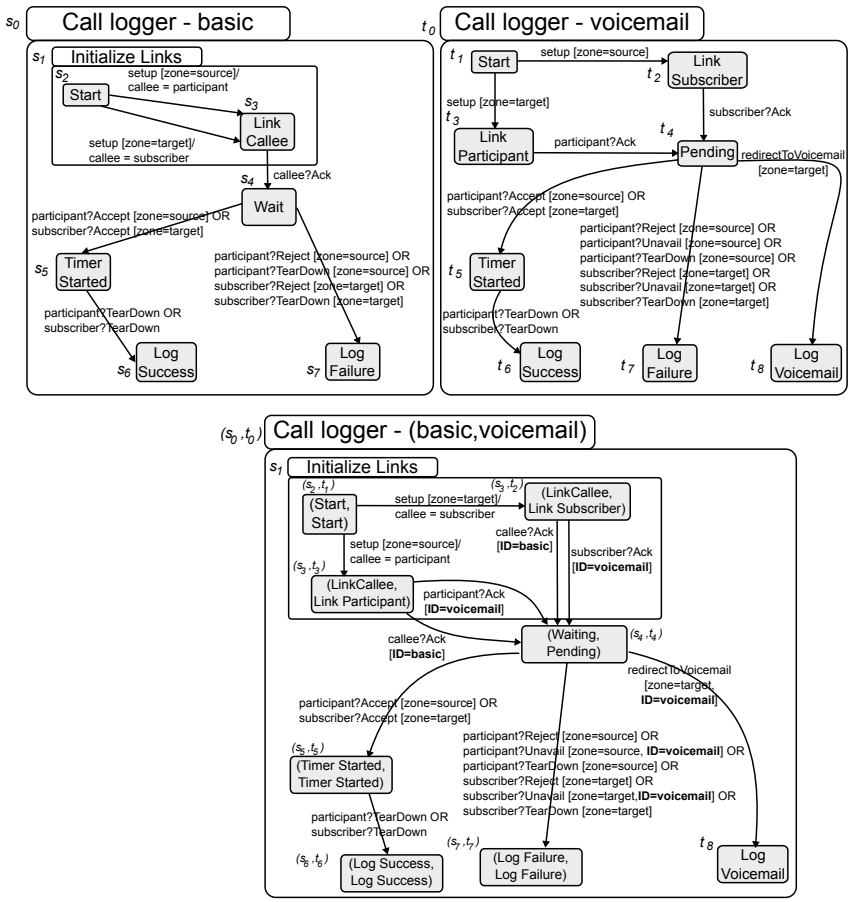


FIGURE 4.5 – Statecharts of the call logger feature variants from [NSC+07]. Conditions shown in boldface capture the origins of the respective transitions.

4.1.1.4 Composition of Orchestration of Services with ADORE

ADORE [3] is meant to modularize service orchestrations [MBFF10]. ADORE allows merging a set of partial orchestrations (*i.e.*, fragments) within a main orchestration and

3. http ://http ://rainbow.i3s.unice.fr/adore/wiki/doku.php

provides checking capabilities for detecting inconsistencies in the merged orchestrations.

- **Inputs** are a main ADORE orchestration and a set of ADORE fragments.
- **Match** is an explicit mechanism that determines where a fragment is statically inserted into the main orchestration. Fragments declare a *hook* activity, a *predecessor* activity and a *successor* activity. The *hook* activity indicates where the fragment will be connected in the main orchestration. *Predecessor* and *successor* activities are used for inconsistencies and behavior checking. An expert of the domain proposes a *composition unit* that is a set of bindings for the *Hook* activity with activities in the main orchestration.
- **Merge** is an algorithm that computes a set of actions (*e.g.*, add an activity, create an order relation, create a variable.) for each binding and executes this set of actions to merge fragments into the main orchestration.
- **Output** is the main ADORE orchestration augmented with the set of fragments unless no binding is provided.
- **Other** algorithms perform a preliminary merge of the set of fragments using logical unification and substitution in the case when multiple fragments must be woven into the main orchestration using the same *Hook* activity.

Fig. 4.6 is the main orchestration for the CaptureWitnessReport scenario of the Car Crash Crisis Management System (CCCMS)as presented in [MBFF10], and related to a common case study for assessing Aspect-Oriented Modeling approaches [KGM10]. Fig. 4.7 and Fig. 4.8 are two fragments, dealing respectively with "requesting video display" and "detecting fake crises" scenarios. Using the composition unit proposed by experts and illustrated in Listing 4.2, ADORE performs the merge of the two fragments into the main orchestration (see Fig. 4.9).

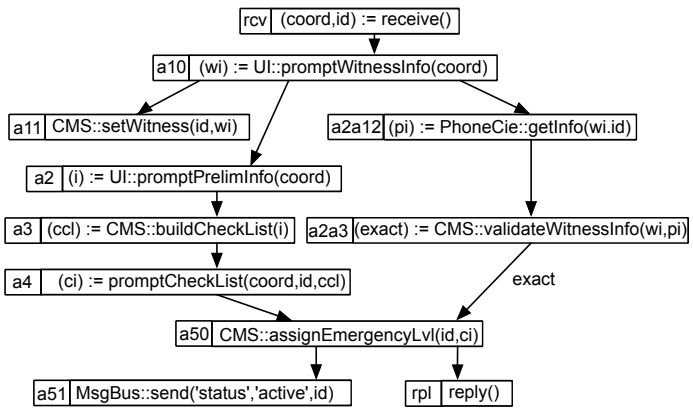


FIGURE 4.6 – The CaptureWitnessRecord workflow from [MBFF10]

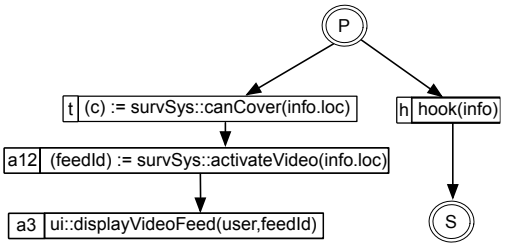


FIGURE 4.7 – The RequestVideo fragment from [MBFF10]

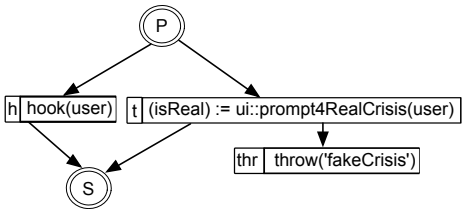


FIGURE 4.8 – The FakeCrisis fragment from [MBFF10]

```
composition cms::captureWitnessReport {
  apply requestVideo(user: 'coord')      => a3;
  apply fakeCrisisDetected               => a4;
}
```

Listing 4.2 – Composition Unit for the CCCMS case study.

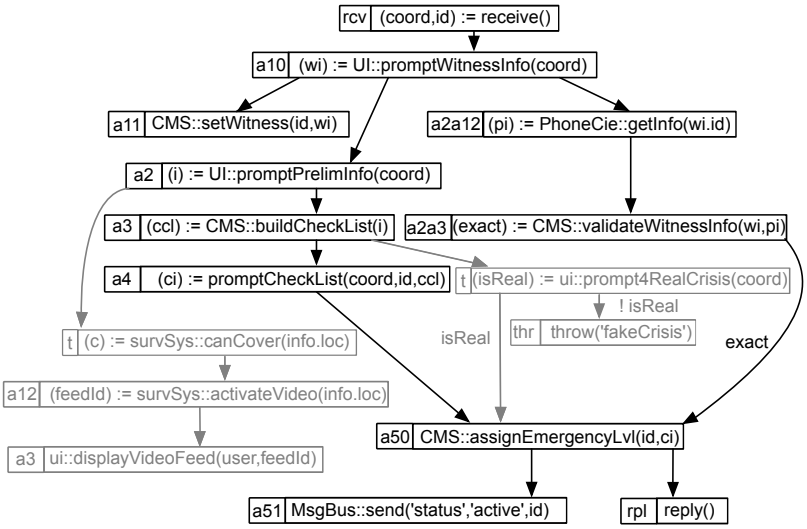


FIGURE 4.9 – The CaptureWitnessRecord workflow augmented with the RequestVideo and the FakeCrisis fragments from [MBFF10]. Grayed activities, guards and wait relationships comes from the fragments.

4.1.2 Capitalization on the Match and Merge Processes

The four operations presented previously seem radically different at first sight : each technique manipulates models that conform to different formalisms ; merge operations are implemented in different programming languages and are meant to be used for different purposes in software development such as global design, meta–modeling, feature interaction analysis and service orchestration modeling. However the central purpose is similar : given a pair of models, each technique builds a new model that contains no more and no less information than the initial models. In the resulting model, each pair of equivalent elements is modeled as a unique element and elements that are not equivalent are included with no changes.
In the following sections, we discuss the implementation of this core behavior of the merge operator in order to re–implement the four merge operators on the basis of a unique framework. We rigorously engineer a core framework that can be systematically reused to build new merge operators for different formalisms. The core framework satisfies the following properties for model merging that were proposed in [BCE+06 ; CNM11] :

- **Completeness :** Ensure that no data is lost along the merging process : each model element from the input models should be represented in the merged model.
- **Non-Redundancy :** no duplicate element exists in the merged model. Duplicate elements originating from the input models must lead to the creation of a single element in the merged model.
- **Minimality :** The merged model contains information that originates solely from the input models.
- **Singularity :** Any pair of matched model elements leads to the creation of a single element in the merged model.
- **Idempotency :** Guaranteeing completeness, non redundancy and minimality ensures that merging a model with itself should produce a model that is an exact copy of the original model.

Whilst some merging operators are dedicated to a given formalism (statecharts, class diagrams and service orchestrations) and Kompose is formalism independent, all these formalisms are described thanks to meta-models that conform to the OMG MOF [OMG10a]. In practice, the implementation of the MOF standard could be the one proposed by the Eclipse Modeling Framework (EMF), namely ECore [SBP+08].

The MOF standard and subsequently its ECore implementation propose to describe meta-models in an object-oriented manner. They provide the following language constructs for specifying a meta-model : package, classes, properties, multiple inheritance and different kinds of associations between classes. The semantics of these core object-oriented constructs that is shared by various languages (e.g., Java, C#).

Since all merging operators shared a decomposition into a match mechanism and a merge mechanism, we propose to capitalize these two mechanisms at the ECore level. Listing 4.3 is an optimized implementation of the capitalized merge operator that traverse the input models once. The algorithm loads the two input models and calls the *sum()* method. The *sum()* method is in charge of traversing the input models

```
function merge(model1 : Object, model2 : Object) : Object is
  // Loads elements for both models
  // Loading returns the root of the model
  root1 = loadModel(model1)
  root2 = loadModel(model2)
  // Calls the sum method for merging the two models
  merged_model = sum(root1,root2)
  // Saves the merged model
  saveModel(merged_model)
end
```

Listing 4.3 – **Algorithm of the *merge* operation**

from the top-most container elements and calls the *match()* method for each pair of model elements. The output of the *sum()* method is a merged model.

Section 4.1.3.4 details the proposed *match* algorithm and Section 4.1.3.5 discusses the proposed *sum* algorithm.

4.1.3 Application of the Unified Framework

For the purpose of validating the unified theoretical framework proposed in Chapter 2, we demonstrate that *(i)*a single model–alignment language successfully represents language–specific matchings and *(ii)*we successfully build a unique algorithm to handle existing merge operations.

4.1.3.1 Overview of the Model Composition Framework Customization

The definition of a new model composition framework for model merging follows the process of customization illustrated on Figure 3.6 in Section 3.2.1. Figure 4.10 illustrates the definition of a model composition framework for model merging.

Model merging relies on a match mechanism that detects equivalent model elements and a merge mechanism (see Section 4.1.3.5) that produces a result from equivalent model elements. We propose a model–alignment language for model merging (see Section 4.1.3.2) to describe mappings between meta–model elements. Mappings (see Section 4.1.3.3) are analyzed to automatically detect matches between model elements using a generic match algorithm (see Section 4.1.3.4). This generic matching mechanism may be customized to change the default equivalence behavior and to handle specific pattern–matching mechanism. Customizations are bound to the **`isEqual()`** method to reduce global customization effort.

4.1.3.2 A Specific Model–Alignment Language for Model Merging

For the purpose of this merge operation, we build a specific model–alignment language. This model–alignment language includes the concepts of *Mappings* and *Directives*, with no extension of *Filter* proposed and the *EquStrategy* selected to indicate

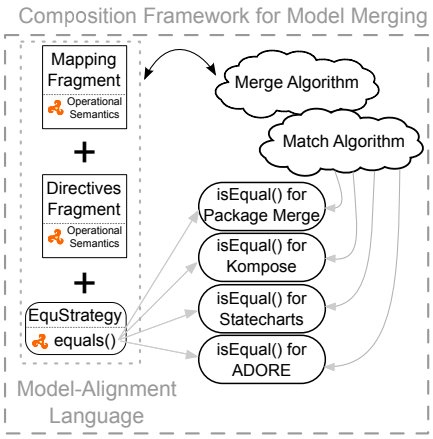


FIGURE 4.10 – Customization of the generic process to build a model composition framework for model merging.

equivalence between model elements. Figure 4.11 shows the specific model–alignment language that we use in this experiment.

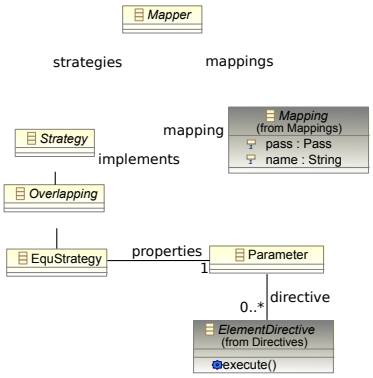


FIGURE 4.11 – Subset of the ModMap language for Model Merge

Since the focus of this approach is on homogeneous model merging (*i.e.*, merging models that conform to the same meta–model), mappings relates meta–classes with the same name. The use of the *EquStrategy* allows restricting equivalence between meta–classes to a set of properties of interest. Examples of mappings are proposed in Section 4.1.3.3 for every existing model merging technique.

4.1.3.3 Mappings and Matches

This section presents the mappings for every model merging technique. This mapping is the specification of a pattern–matching mechanism that allows detecting matches at the level of the models to compose.

UML Package Merge
Since UML Package Merge takes UML2 models as input and produce a UML model as output, we propose a specification of mappings among UML models such that every pair of NamedElement (*i.e.*, model element with a name) which property **`name`** are equal allows merging this pair of model elements into a single model element. Figure 4.12 shows the mappings needed to simulate UML Package Merge with our approach.

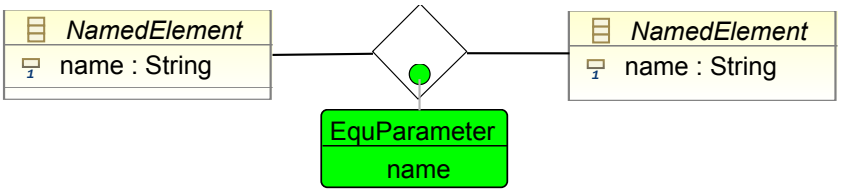


FIGURE 4.12 – Specification of the UML Package Merge mapping at the meta–class level

Kompose
Kompose is a generic composer on homogeneous models that works on any meta–model implying that a specialization has been provided to handle such meta–model. We illustrate the specification of mappings among ECore models which is one of the specialization provided by the Kompose tool. The basic composition of ECore models is defined such that every pair of ENamedElement (*i.e.*, a model element with a name) which property **`name`** are equal allows merging this pair of model elements into a single model element. In addition, we consider two EOperation equal if they have the same return type and identical operation parameters such that EParameters are of the same type. Figure 4.13 shows how we specify mappings for ECore models.

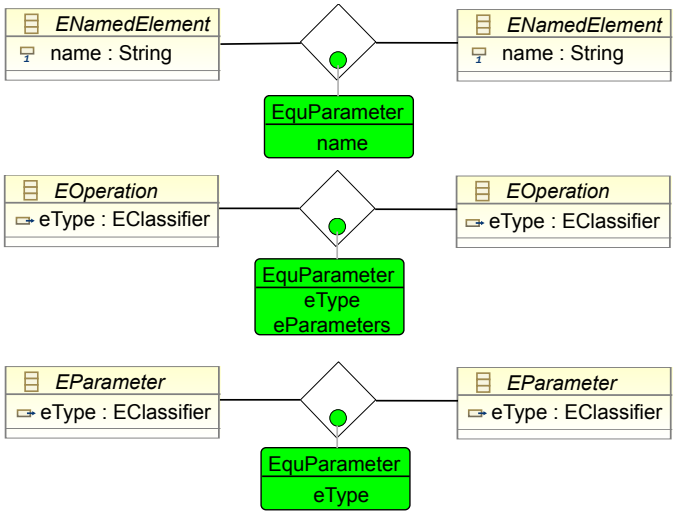


FIGURE 4.13 – Specification of the Kompose mappings at the meta–class level

Statecharts Merge
Matching and Merging Statecharts takes two statecharts as input and produce a new statechart. We choose to use the statechart diagrams from UML2 to represent the input and output models. For the statecharts to be properly merged, we consider that the property **`name`** of two Package, two ExecutionEvent, two Port or two Region must be equal. About Transition, the **`effect`**, **`guard`** and **`trigger`** properties must be equal. The **`specification`** property of two Constraint must be equal and the **`body`** property of two OpaqueBehavior must be equal. Figure 4.14 shows how we specify mappings for Statecharts models.

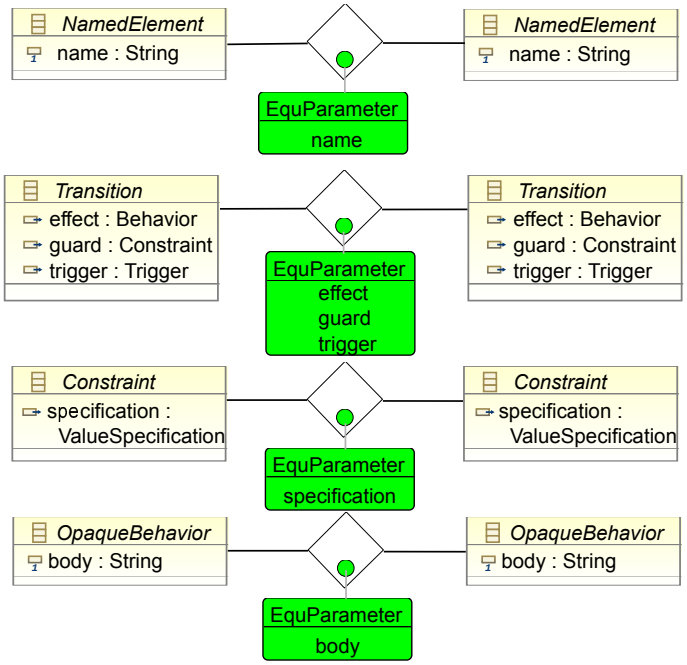


FIGURE 4.14 – Specification of the Statecharts Merge mappings at the meta–class level

ADORE Merge
ADORE weaves fragments of orchestration into a main orchestration. Both orchestrations conform to the ADORE meta–model that contains concepts of Relation, Activity and Variable. The equivalence of two Activity implies that the properties **`inputs`** are equal and the equivalence of two Variable implies that the properties **`type`** are equal. Figure 4.15 shows how we specify mappings for ADORE models.

4.1.3.4 A Unique Algorithm for Matching using Mappings

We propose a unique algorithm (see Listing 4.4) for matching model elements. The matching algorithm detects equivalence of a pair of model elements within an *isEqual()* method which default behavior is to compare values of properties and to return a boolean value.

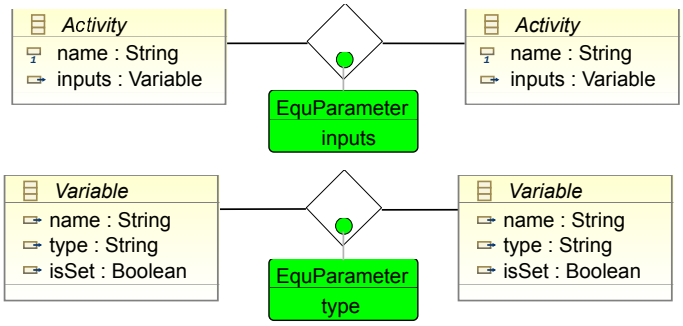


FIGURE 4.15 – Specification of the ADORE Merge mappings at the meta–class level

The default behavior of the *isEqual()* method is not enough to properly detect matches in various situations. The matching mechanism should be precise enough to avoid merging objects that are not expected to be merged. Since the matching mechanism is domain–specific (*i.e.*, matches are detected among elements that conform to a specific meta-model), we allow extending the pattern–matching mechanism to include specific heuristics and/or complex matching computations. The extension mechanism is limited to the redefinition of the *isEqual()* method to isolate domain-specific parts and to limit the effort of extension as low as possible.

From the examples of the four existing techniques, the *isEqual()* method thus may contain specific UML Package–Merge detection rules, heuristics and similarity degrees computation for UML statecharts, or data provided by experts such as Kompose signatures or activity matches for ADORE models.

We illustrate the redefinition of the *isEqual()* method for ADORE models in Listing 4.5 w.r.t. the proposition of matches from experts (see Listing 4.2).

The result of the match function allows merging model elements that are identified as equivalent.

```
function match (object1 : Object, object2 : Object) : Boolean is
  result := true
  type1 := object1.getMetaClass
  type2 := object2.getMetaClass
  m := getMapping(type1,type2)

  if m != void then
    boolean b := m.properties.forAll{p|
      if p.isReference() then
        match(get(p,object1),get(p,object2))
      else
        // equals represents the specific method
        // of testing the equivalence between
        // objects. This method may be extended
        // with domain-specific processing
        equals(get(p,object1),get(p,object2))
      end
    }
    result := b
end
```

Listing 4.4 – **Algorithm of the match operation**

```
operation isEqual(p:Property,o1:Object,o2:Object): Boolean is
  var a : adore::Activity
  var h : adore::Hook
  result := false
  a ?= o1
  h ?= o2
  // requestVideo
  if a.name.equals("a3") then
      // compare properties
      result := (get(p,a).size() == get(p,h).size()) and
        get(p,a).equals(get(p,h))
  else
    // fakeCrisisDetected
    if a.name.equals("a4") then
      // compare properties
      result := (get(p,a).size() == get(p,h).size()) and
        get(p,a).equals(get(p,h))
  end
  ...
end
```

Listing 4.5 – Implementation of the *isEqual()* method for ADORE models

4.1.3.5 A Generic Sum Algorithm

We propose a generic algorithm for deeply merging two model elements. This algorithm is shown in Listing 4.6. The *sum()* operation takes two elements as parameters and creates a new element as a result. The *sum()* function calls the *match()* function to compute matches (see Section 4.1.3.4).

```
function sum(object1 : Object, object2 : Object) : Object is
  pre type(object1) = type(object2)
  // creates a new object
  result = new object from type(object1)
  if match(object1, object2) then
    // merging properties
    foreach property p1 from object1
      foreach property p2 from object2
        if match(p1,p2) then
          // if relations, creates a new property and calls
          // sum() recursively
          if (p1 is a relation and p2 is a relation) then
            new_value = new property p
            set(new_value,sum(get(p1,object1),get(p2,object2)))
          else
            // merges values and resolve conflicts if any
            new_value = resolveConflicts(value(p1),value2(p2))
        end
        // add property to result
        add(result,new_value)
      end foreach
    end foreach
  end
end
```

Listing 4.6 – **Algorithm of the *sum()* operation**

4.1.4 Properties of the Merge Implementation

The concrete implementation of the merge operator satisfies the following properties

- **Completeness :** The implementation of the merge operator allows creating a single model element from two model elements identified as equivalent and allows copy non matching model elements in the merged model with no changes. No operation of filtering or destroying model elements appears in the implementation, thus satisfying the completeness property of the merge operator.
- **Non-Redundancy :** The merge operation is implemented in a single pass : both input models are traversed once and model elements that match are merged in single model elements. By construction, the merged model cannot contain redundant data. This statement holds with the following assumptions :
  - Mappings are one-to-one only : a single model element from one model is related to a single model element from another model.
  - Multiple mappings for a single model element are forbidden. Multiple mappings may end up in situations where the *sum()* algorithm produces different model elements that should be merged with one another.
- **Minimality :** The implementation of the merge operator manipulates model elements that comes from the two inputs models. Any new model element or value originates from the two input models.
- **Singularity :** The *sum()* function calls the *match()* function for any pair of model elements. The *match* function calls the *getMapping()* function to retrieve a map-

ping between two elements. We assume correct that the *getMapping()* function is correct. Since we call the *getMapping()* function for every pair of model elements, we ensure that every mapping from the mapping specification is processed. Thus, the correctness property holds.

– **Idempotency :** Assuming that mappings relate each construct of a model to itself, the *match()* operation should detect matches for every construct. Since we guarantee the completeness, non redundancy, and minimality properties for the merging process, the *sum()* method produces a merged model that is an exact copy of the input model.

4.1.4.1 Discussion

This approach provides significant results using the unification framework for model composition activities. Starting with four existing model composition approaches that were designed independently but still supporting the purpose of merging homogeneous models, we propose a unique algorithm that realizes the intention, a unique matching mechanism that supports the equivalence interpretation of correspondences and a unique representation of correspondences among MOF–like (meta–) models. As we expect, variability of the four existing approaches is encapsulated in the definition of language–specific mappings that may be provided by experts of the domain or automatically computed. This work properly illustrates how powerful our theoretical framework may be in action and how subsequent model composition tools may be adapted and reused in contexts that differ from the original work.

Objectively, we must keep in mind however that unification is possible to some extent. The global purpose of the model composition approaches should be close enough from one another and some minor characteristics may influence the final computation of the merge operation. For instance, the pre– and post– alignment of models provided by Kompose participates in the model merging activity. Since this capability is only available in Kompose and the boundaries of its action is a-priori infinite, we voluntarily discard model pre– and post– processing from the current proposition.

To conclude, we should consider the unified approach as the minimal set of mechanisms to support model merging for various languages, providing a highly reusable independent module. Further extensions will only provide operations that are context– or problem– specific.

4.2 Interoperability and Heterogeneous Composition

The goal of this experiment is to automate the integration of existing legacy [CBJ10]. Model integration is another form of model composition. The intent is to produce adapters between objects to allow data–sharing or transformation between them. The generation of these adapters requires the definition of mappings and the definition of a specific processing to properly to generate meaningful adapters. The definition of mappings is thus provided by domain experts between models of the legacy API.

The model composition framework for model integration based on these mappings is executed at runtime to convert concepts from one API in concepts of the other API.

4.2.1 Context

This experiment was conducted in the context of the MOPCOM–I project which focus on using MDE for software specification and software design on reliable model–based processes. The MOPCOM–I project is funded by the competitiveness cluster of Brittany called "Images et Réseaux". The project regroups four industrial partners and four academic partners as follows :

- Thales Systèmes Aéroportés (Brest, France) from the Business Group Airbone Systems (BGAS)
- Technicolor Rennes Research Center(Rennes, France)
- France Telecom Research Center (Lannion, France)
- Sodifrance (Rennes, France)
- Research Team CAMA (Components for Adaptable and Mobile Architecture) from ENST Bretagne (Brest, France)
- DTN (Développement de Nouvelles Technologies) research laboratory from ENSTA (Brest, France)
- Group ALCC (Architectures Logicielles et Composants de Confiance) from the UBS-Valoria research laboratory (Vannes, France)

The goal of the MOPCOM–I project is two-fold : *(i)*propose formal model–driven processes for software specification and design to support engineering activities ; *(ii)*use model–based techniques to strengthen the safety of software developments by supporting early design and post design verification techniques.

In the context of the MOPCOM–I project, the Triskell Team is in charge of two work packages : *(i)*WP 2.1 Models verification and *(ii)*WP 2.4 Reliable models and meta–models fusion. Our contribution in those work packages is validated against the Technicolor case study on video and broadcasting equipments management.

4.2.2 Technicolor Distribution and Broadcasting Devices Management

Technicolor is a provider of technologies, systems and services for managing video content from the content production activities to content broadcasting activities, including networking. For instance, the Thomson Extensible Management System for Digital TV deals with the intercommunication of heterogeneous legacy hardware devices (*i.e.*, Network Adapters, MPEG Multiplexers, Encoders, Decoders). Hardware devices are designed by different manufacturers and from different technologies that use specific API for control and management (see Figure 4.16). Tackling the heterogeneity, Thomson provides a distributed architecture with a set of remote user interfaces and administration servers that communicate with one another through an intermediate API called XMS. Administration servers main responsibility is thus to convert XMS orders into device–specific commands.

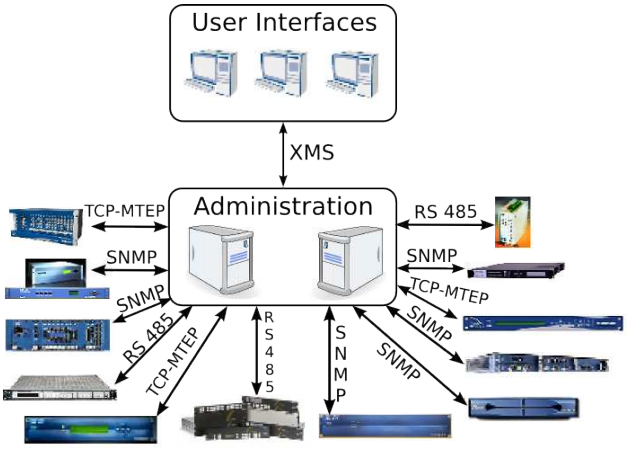


FIGURE 4.16 – Global View of the Management Architecture with some examples of managed physical devices

4.2.3 Legacy Systems and Translation Issues

Toward the purpose of integrating with various existing platforms and systems, Technicolor develops API for an extensive set of protocols such as MTEP, SNSM, XMS, TCP/IP, or RS232/485. The evolution of legacy equipments induces the development and the maintenance of several versions of APIs, both for the specific protocols and for the intermediate language XMS. This situation leads to the combinatorial explosion of the number of adapters to be developed.

In the context of the MOPCOM-I project, Technicolor proposed the case study of building adapters between the XMS API (the intermediate protocol for communication) and the MTEP API (a device–specific protocol for communication and control). The purpose of the case study is to identify existing flaws in the process of building adapters and propose techniques to increase its automation.

The state of practice in converting MTEP to XMS and conversely is based on informal textual descriptions which lead to ambiguous interpretations and subsequently more effort in designing, implementing and validating converters. Lack of formal representation and semantics also hinders the automatic synthesis of the translation process from the mapping specifications.

4.2.4 A Semi–Automated Solution for Integrating Legacy Systems

Our contribution on the Technicolor case study is to propose a model–driven approach to alleviate the implementation of multiple adapters. From a first step of automatic reverse–engineering relevant concepts from APIs to high–level models, we use the ModMap framework to *(i)*support the definition of mappings between concepts in these API high–level models ; and to *(ii)*automatically generate adapters for converting APIs using AOP techniques to avoid changes in legacy API.

The global process of the approach is composed of four steps as illustrated in Figure 4.17 :

1. **Model Abstraction From Legacy Code**
   We analyze the legacy code of the API to find all relevant classes. We automatically build an application model using reverse–engineering techniques.
2. **High Level Mappings Description**
   Designers of converters propose mappings at the model level between classes from the MTEP API and classes from the XMS API.
3. **Translation Strategies**
   Designers select translation strategies to specify how data should be transfered between model elements.
4. **Generation of Bidirectional Adapters**
   Models and mappings provide enough data to automatically generate bidirectional adapters as aspects. We propose a non invasive process to extend legacy code with adaptation capabilities.

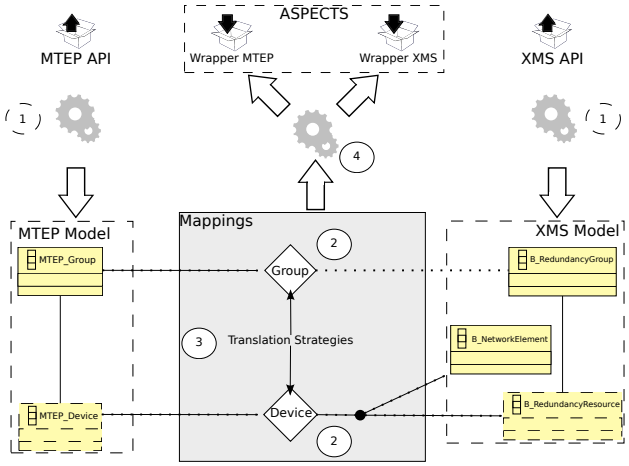


FIGURE 4.17 – The integration process is composed of four steps. We automatically extract models from the APIs. Users define mappings and select strategies for translating model elements. We automatically generate adapters for each API.

4.2.5 Application of the Unified Framework

Among the four steps of the global process, the description of high level mappings and the selection of translation strategies are the concrete application of the theoretical framework.

Section 4.2.5.2 presents the model–alignment language that we define for model integration, Section 4.2.5.3 illustrates how we use the model–alignment language to

design converters between two specific API and Section 4.2.5.4 details how we automatically generate bidirectional and non invasive adapters from the alignment of the two models of API.

4.2.5.1 Overview of the Model Composition Framework Customization

The definition of a new model composition framework for model integration follows the process of customization illustrated on Figure 3.6 in Section 3.2.1. Figure 4.18 presents the definition of a model composition framework for model integration.

Our definition of model integration in the context of heterogeneous legacy API is the ability to transfer objects that conform to a given structure into objects that conform to another structure and vice–versa. Transfer is achieved through a set of bidirectional transformations supported a mechanism of adaptation. From the specification of mappings between the two structures, we produce a set of adapters that are used at runtime to *convert* an object from an API into an object from another API. We propose a model–alignment language for model integration (see Section 4.2.5.2) to capture the specification of mappings proposed by experts of the domain. A specific interpreter traverses the specification of mappings (see Section 4.2.5.3) to produce a model of adaptation that defines the expected set of converters between two API given the set of translation strategies selected by experts.

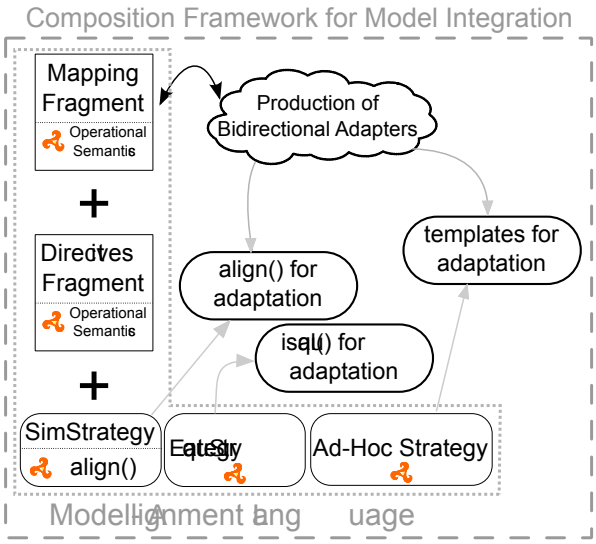


FIGURE 4.18 – Customization of the generic process to build a model composition framework for model integration.

4.2.5.2 A Specific Model–Alignment Language for Model Integration

For the purpose of model integration, we build a specific model–alignment language. The model–alignment language includes the concepts of *Mappings* and *Directives* with no extension of *Filter* proposed for this experiment. Strategies selected are

the *EquStrategy*, the *SimStrategy* and the *AdHocStrategy* to respectively define equivalence between model elements, to define similarity between model elements and align model elements given a set of atomic operations, and to propose an escape mechanism to express complex computations to properly adapt model elements.

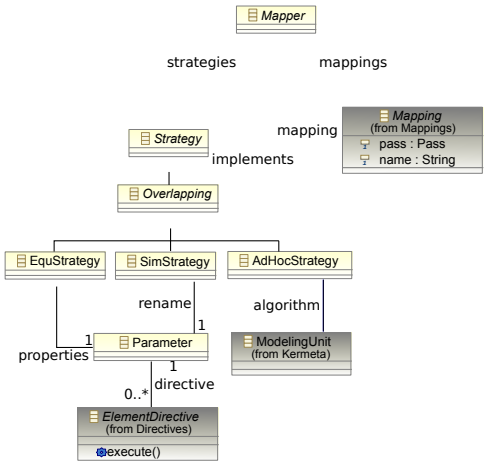


FIGURE 4.19 – Subset of the ModMap language for Model Integration

4.2.5.3 Design Converters for the Integration of MTEP and XMS

This section illustrates the definition of mappings and the selection of translation strategies between the MTEP API and the XMS API.

Step 1 from the global process extracts a model–based representation of the source code of an API using JaMoPP[4]. The extracted model contains model elements from the domain that we want to align with the model elements of another API. Figure 4.20 shows the model extracted from the MTEP API and Figure 4.21 shows the model extracted from the XMS API.

In the global process of the approach, step 2 and step 3 are the definition of mappings between sets of models elements and the selection of specific interpretations. We use the specific model–alignment language for model integration proposed in Section 4.2.5.2 and shown in Figure 4.19. With this subset of the ModMap language, we propose mappings for the integration of a subset of the MTEP API and a subset of the XMS API as illustrated in Figure 4.22. This mapping specifically illustrates mappings between the elements of *topology* and the elements of *redundancy management* of the two protocols.

4. http ://www.jamopp.org/index.php/JaMoPP

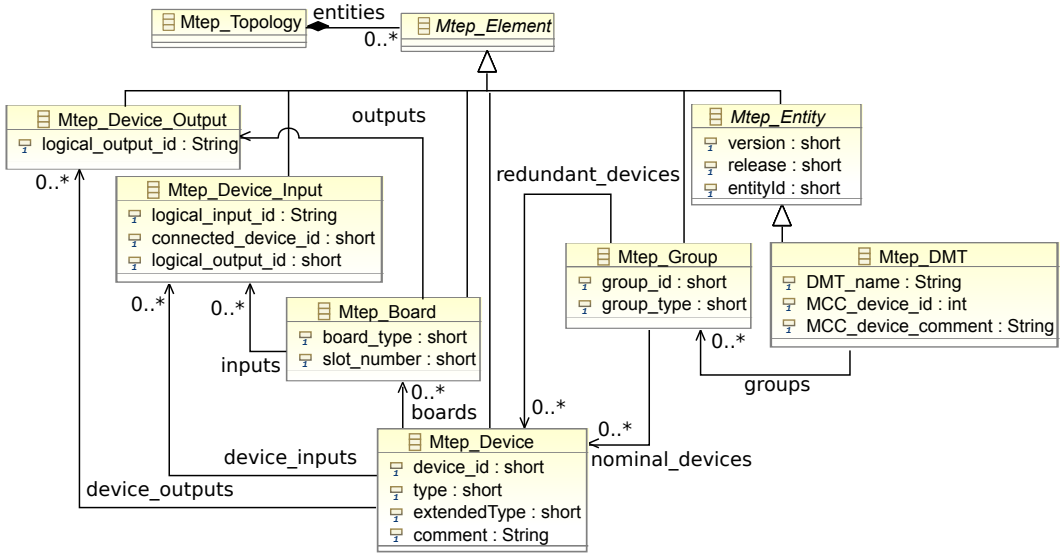


FIGURE 4.20 – Model of the MTEP API.

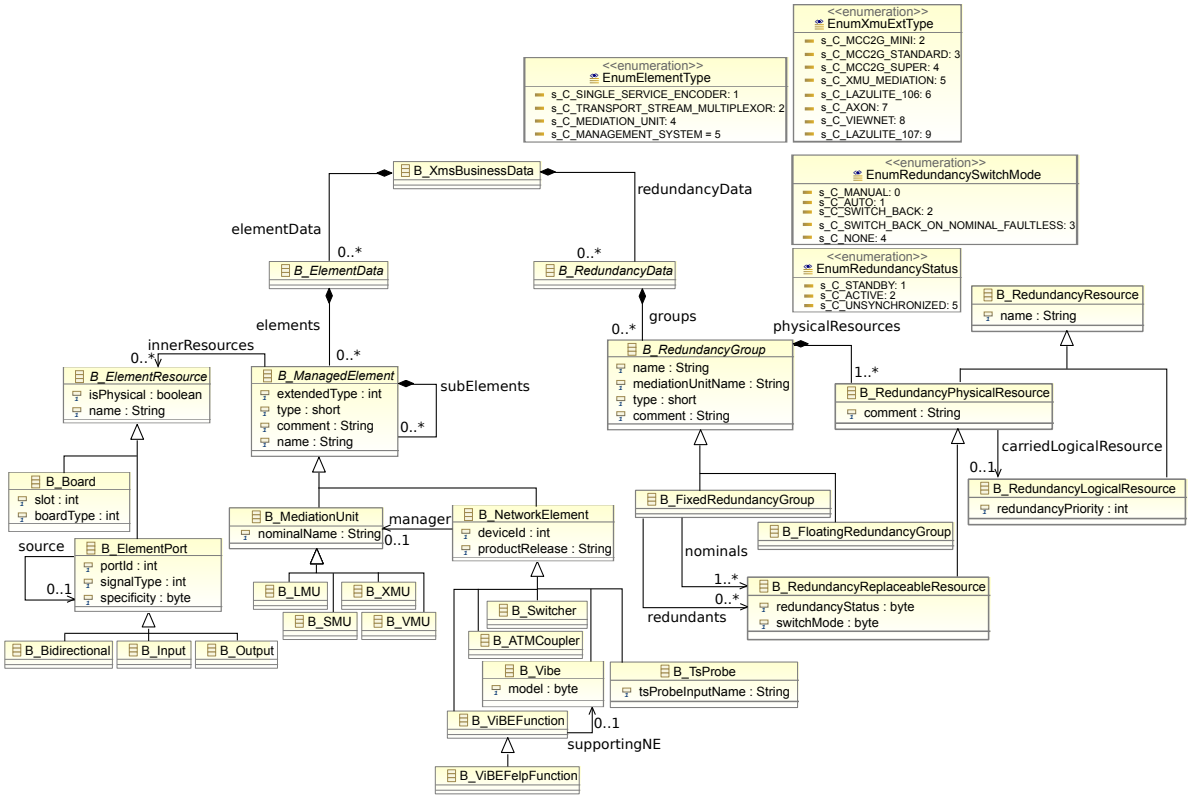


FIGURE 4.21 – Model of the XMS API.

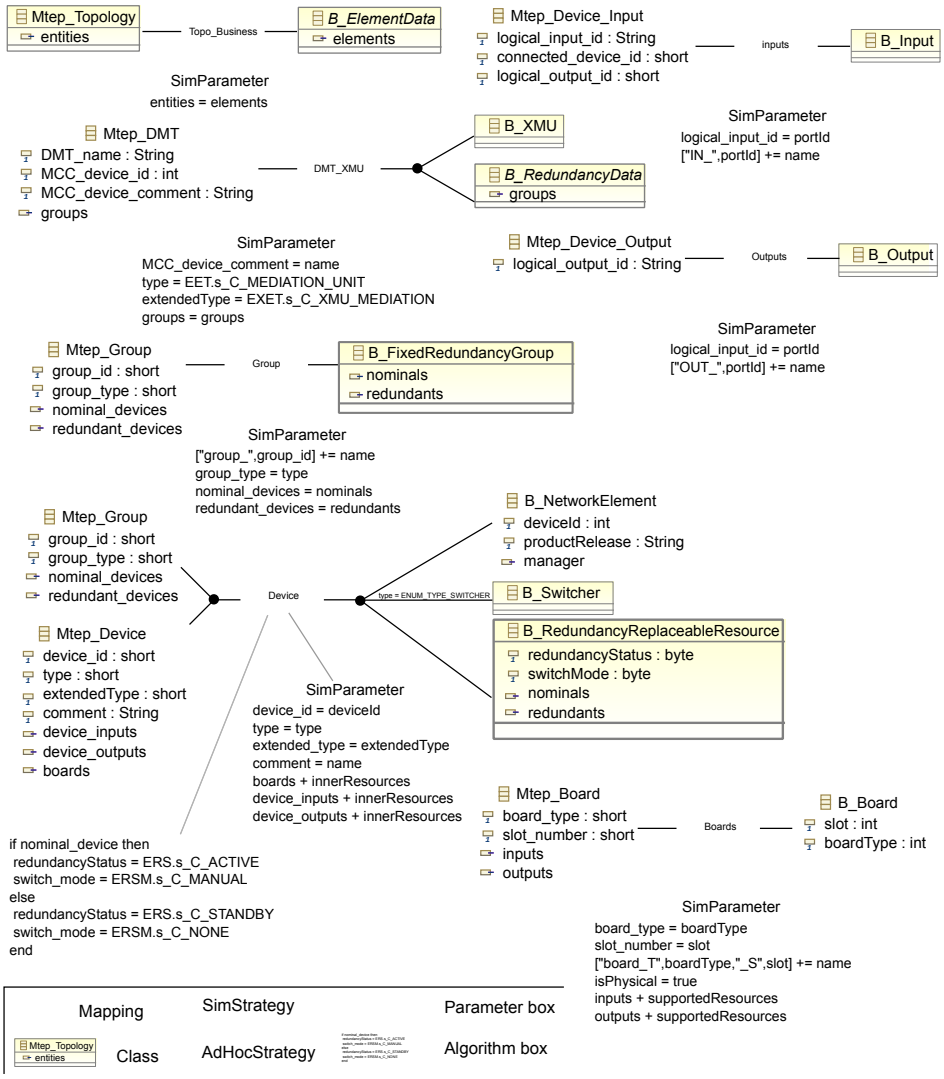


FIGURE 4.22 – Model of mappings for the integration of the MTEP and XMS API.

4.2.5.4 Generation of Bidirectional Non Invasive Adapters

The last step of the process is the automatic generation of adapters. Similarly to model–to–code transformations, the code generator uses two input parameters : the mapping model and the Kermeta code that supports the translation strategies. We adapted code generation methods to use aspect–weaving techniques, *i.e.*, we generate adapters as aspect to avoid the invasion of the original code of API. The production of non invasive adapters is composed of three stages.

1. Mappings are automatically converted to a model of adaptation. Each mapping is converted into two Kermeta aspects : one for the source model elements of the first API and one for the target model elements of the second API. These aspects contain Kermeta operations that encapsulate the adaptation between the two API. Strategies and additional alignment directives are translated into effective Kermeta transformation code.
2. One key concern of this approach is to be non–invasive regarding legacy API code. The model of adaptation contains definitions of both API classes and adapters. The generation of code from the model of adaptation would build a set of new classes that would overwrite legacy classes. We prevent this situation by removing class definitions that are equivalent between the model of the API and the model of adaptation. We apply a filtering method that only keeps new class members or additional utility classes. The filtering method checks names of classes and their signature to identify equivalent class definitions which are subsequently removed from the adaptation model.
3. Using the Kermeta Compiler, we process the model of adaptation to produce Scala aspect code. Classes that do not exist in the legacy code are created whereas classes that already exist are augmented with inter–type declarations. Inter–type declarations encapsulate the translation behavior between existing classes. Adapters between the two API are composed with the original legacy code at load–time using the Scala compiler. Load–time weaving is deferred until the point that a class loader loads a class file and defines the class to the Java Virtual Machine (JVM). As a consequence, additional capabilities we brought through the adapters production does not pollute existing code embedded in legacy API.

4.2.6 Evaluation

The evaluation of the approach is based on comparing efforts between classic development techniques (followed by domain experts) and between our semi-automated process. As a benchmark, we compare our solution to the Thomson Extensible Management System evolution on the specific example of MTEP to XMS protocol translations.

4.2.6.1 Impact of Automation on Adapters Production

The case study is based on a subset of Thomson MTEP to XMS conversion. This subset contains nine MTEP-related concepts and thirty XMS-related concepts defined

in their respective APIs.

From the correspondence specifications provided by experts, Thomson developers implemented twenty mappings to carry out the bijective translations between MTEP and XMS (see Figure 4.23 for details about mappings ratio). The application of our process on the same case study involves only seven bidirectional mappings, whose distribution is represented in Figure 4.24.

We can draw two conclusions from these figures :

1. We reduce the number of mapping descriptions to handle the case study.
2. Mapping descriptions complexity is globally reduced since 74% of mapping descriptions are One–to–one mappings and Many–to–One and Many–to–many mappings are scarcely used (13% for both kinds).

The first point comes from the use of a more adequate DSL to express mappings at the right level of abstraction. At the code level, developers implement complex mappings as one-to-one mappings because the implementation language do not offer higher-order mapping operators. The mapping language we propose offers more expressiveness to declare mappings so some of them, identified by experts, are not expressed anymore as single mappings but are encapsulated into higher-order mappings. The second point is related to the expressiveness of the DSL versus the implementation practices in Java. We observed that when people use low-level correspondence languages, they are more tempted to violate implementation practices of the Visitor Pattern to access incidental information (information from multiple concepts that do not take part in the original described mappings). Our approach limits this problem since the mapping DSL offers higher abstractions to retrieve data in a proper way : users are able to describe relations between mappings, so concepts involved in a mapping are confined to the original inputs and outputs of the wrappers.

Figure 4.25 is to be compared with Thomson implementation of adapters (100% of strategies would be Ad Hoc strategies). This figure shows that most of strategies (86%) used to handle the case study are semi-automatic. Of course, it is not possible to automatically define all mapping implementations : that is why we provide a way to use a more powerful language to implement the remaining mappings.

These results are a first indication, on a relatively small example, that the use of a high-level language for mapping descriptions helps reduce the number of adapters to be implemented. It also gives additional evidence that the use of generative techniques cuts down global complexity and effort to produce adapters.

4.2.6.2 Comparison of Effort

The second stage of our evaluation deals with effort estimation in terms of the number of Lines of Code (LOC). Thomson global adapter size for the case study is about 5350 LOCs to realize the bi-directional translation between MTEP and XMS protocols. Our approach is implemented using only 510 Kermeta LOCs. The effort has been evaluated to 136 hours of person work for the manual implementation of a new adapter, compared to 150 hours to handle the same example with our approach.

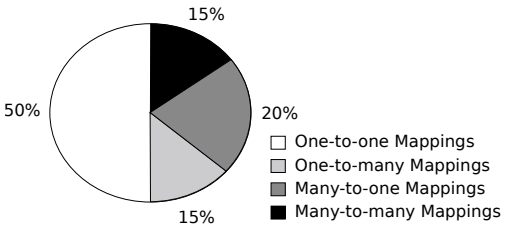


FIGURE 4.23 – Distribution of the twenty mappings identified by the experts of the domain and implemented with Java.

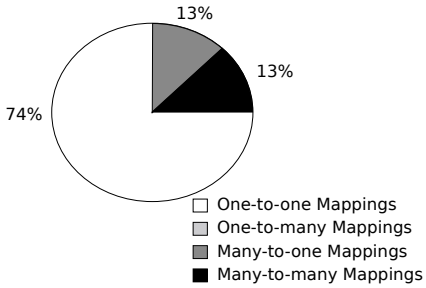


FIGURE 4.24 – Distribution of the eight mappings identified by experts of the domain and implemented with the ModMap mapping language.

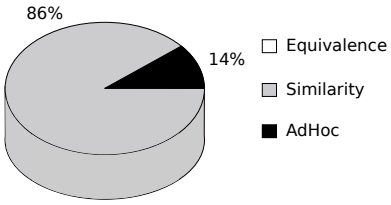


FIGURE 4.25 – Ratio of strategy types used to map the MTEP and the XMS API.

Considering up to 50 extra hours to take into account the introduction of a new mapping language and a new language for strategies definition for users, the effort needed to use our process is of 150 up to 200 hours (see Table 4.1) for the very first version of an adapter, which is slightly more than the manual approach.

However, the mean of the effort to produce a new version of an adapter for both approaches are 57 hours for a manual implementation versus 9 hours with the semi-automatic process.

These results have several consequences : First, we are able to say that our approach needs less manual implementation from users. Second, thanks to generative techniques, we were able to reduce the number of bugs in code and thus time spent in debugging has been drastically reduced. These improvements allow users to save some maintenance effort on the code in further evolutions. Figure 4.26 illustrates the

Production of a new adapter	**Manual Approach**		**Generative Approach**	
	v1	v2 (avg)	v1	v2 (avg)
Code length (LOC)	5350	-	570	-
Total TDEV (Hours)	136	**+57**	150-200	**+9**

TABLE 4.1 – Effort for manual and generative approaches for the production of a new adapter. The production of a second version (v2) increases the effort by an average of 57 hours for the manual approach and by an average of 9 hours for the generative approach.

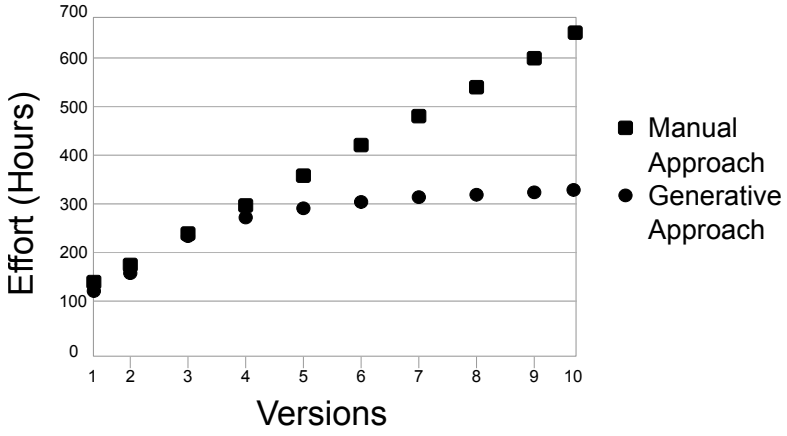


FIGURE 4.26 – Cumulative effort for the production of new versions of adapters using manual or generative approach : effort (time of development in month) is on the y-axis and versions on the x-axis.

effort reduction in the production of ten successive versions of this adapter. A manual approach induces a constant effort to develop and test new adapters versions. Our semi-automatic approach is expensive on the very first version (learning overhead and complex mapping definitions) but costs decrease with time as learning overhead decreases in further evolution. Even though benefits, in terms of efforts, observed on Figure 4.26 are relatively small, we have to keep in mind that this process is to be repeated, for instance, on the five API definitions presented on Figure 4.16 with, let us say 10 versions each. Since we want these APIs to be integrated with each other, it ends in the production of $(5 * 10)^4$ adapters. A potential extrapolation of our results would give an effort reduction of up to 87% by using our approach.

4.2.7 Discussion

This case study presents a process to semi–automatically produce adapters for existing legacy API. Mixing reverse–engineering, model–to–model transformation,

code generation and aspect weaving techniques together, we alleviate tedious and error–prone activities. This work on the Technicolor case study provides significant benefits in this specific context and also demonstrates that the definition of model composition goes beyond model merging and similar activities. Model integration is an activity that fits in the definition of model composition that we propose in the unified theoretical framework : the definition of mappings and the selection of specific strategies allow *(i)*describing mappings declaratively between legacy models, and allows *(ii)*automating the process of building converters.

In hindsight, we discover that specifics of a given model transformation is hard to capture within the boundaries of a proposed set of semantics (*i.e.*, strategies). Thus, we provide an escape mechanism to the meta–language using Kermeta to allow the definition of arbitrary complex mappings.

With this application of the unified theoretical framework in mind, we should further explore a set of software life–cycle activities such as those that we discussed on Section 1.3.1.7. Further work on the relationship between software life–cycle activities and specific model–alignment languages should help proposing and developing generic model composition frameworks dedicated to specific model–related activities.

4.3 Bridging the Gap between Structure and Behavior in the context of SOA

We illustrate model synchronization in the following case study. Synchronization is another kind of model composition activity that requires to capture changes between an arbitrary number of models and propagates changes if necessary to maintain global consistency among the various software artifacts. This experiment is summarized in [CMBF+11].

4.3.1 Service–Oriented Architecture Background

Developing a large–scale software system as a Service–oriented Architecture (SOA) involves the creation and integration of a variety of services. Services must be coordinated to adequately participate in the required behavior of the system. Model–driven development of such systems is highly likely to produce a variety of models capturing the many diverse design concerns that arise during development. The management of models in such multi–modeling environments is known to be challenging. In particular, activities related to checking and maintaining consistency among the multiple views of a system can be complex.

There is a need for techniques that developers can use to detect conflicts and divergences across multi-models of systems developed using SOA. Two models diverge when one model consists of elements that do not correspond to elements in the other model.

Our work specifically addresses the problem of synchronizing SOA business process models with domain models. The approach described in this paper provides SOA

designers with integrated generative and model composition techniques that can be used to automatically propagate divergence resolution strategies across these models. The core of the iterative synchronization approach consists of four major steps : *(i)*the generation of a structural model based on the data extracted from the business process model, *(ii)*the merge of the generated model with the initial domain model, *(iii)*the identification of formal divergences between these two models and finally *(iv)*the automated propagation of resolution strategies provided by experts.

4.3.2 Design a Car Crash Crisis Management System

4.3.2.1 The Crisis Management System

We illustrate the approach using a case study problem described in a Transactions on Aspect-Oriented Software Development (TAOSD) special issue on AOM [MBFF10]. The purpose of the special issue was to compare the application of existing AOM approaches on a common system development problem, namely the development of a Crisis Management System (CMS).

In the case study, a CMS is a system that facilitates the coordination of activities and of information flow between all stakeholders and parties that need to work together to handle a crisis.

4.3.2.2 The Car Crash Crisis Management

Among the multitude of crises handled by CMS, including terrorist attacks, epidemics, or accidents, we focus on car accidents. Car accidents are handled by the Car Crash CMS (CCCMS) which "includes all the functionalities of general crisis management systems, and some additional features specific to car crashes such as facilitating the rescuing of victims at the crisis scene and the use of tow trucks to remove damaged vehicles" [KGM10, §2.4, p.5]. The original system includes ten use cases described using textual scenarios. For ease of understanding, we illustrate our approach on the *Capture Witness Report (CWR)* use case only.

The CWR case study (use case #2 in the original document) captures the set of actions that a *Coordinator* takes to create a new *Crisis* based on the information reported by the *Witness* of a car accident. The main success scenario for this use case (extracted from the requirements document) is described in Figure 4.27. The subject of the use case is the CCCMS system represented by *System*. Two actors are involved in the sequence of activities needed to report a car crash : *(i)PhoneCompany* is the role played by an external partner that provides phone–related information, and *(ii)Coordinator* is the role played by the person who interacts with the CCCMS system through a graphical user interface to enter information.

We focus on the contribution of two experts in the definition of a solution to this CWR use case : a domain model expert (e_a) designs the structural view of the system and a business process expert (e_b) designs the behavioral view (*i.e.*, the set of activities and the flow of control between these activities) of the system.

Coordinator requests *Witness* to provide his identification.

1. *Coordinator* provides witness information to *System* as reported by the witness.
2. *Coordinator* informs *System* of location and type of crisis as reported by the witness.
   In parallel to steps 2 – 4 :
   - 2a.1 *System* contacts *PhoneCompany* to verify witness information.
   - 2a.2 *PhoneCompany* sends address/phone information to *System*.
   - 2a.3 *System* validates information received from the *PhoneCompany*.
3. *System* provides *Coordinator* with a crisis-focused checklist.
4. *Coordinator* provides crisis information to *System* as reported by the witness.
5. *System* assigns an initial emergency level to the crisis and sets the crisis status to active.

Use case ends in success.

FIGURE 4.27 – Textual Scenario of Use Case #2 : "Capture Witness Report"

4.3.2.3 Domain Model Design

Figure 4.28(a) is a class diagram that captures problem concepts identified from the requirements and that are relevant to the CWR use case. This domain class diagram (CD_D) is designed by e_d who formalizes his deep understanding of the various concepts manipulated in the CCCMS system. The main concepts with respect to the CWR use case are the following :

`Crisis` : is the concept shared by any CMS system. A **`Crisis`** occurs at a given location and at a given time, it has an emergency level, a status and possibly some additional information. A **`Crisis`** may be reported by a **`Witness`** and may include **`Mission`**s.

`Witness` : is a person who reports a **`Crisis`**.

`Mission` : is an action that should be taken when a **`Crisis`** is reported.

`CheckList` : is a list of things that should be checked with a **`Witness`**.

`CMSEmployee` : is a human resource who is qualified and capable of performing **`Mission`**s in the context of a **`Crisis`**.

4.3.2.4 Business Model Design

The business process model (BPM) associated with the CWR use case is represented in Figure 4.28(b). According to SOA principles, e_b designs this business process model with regard to his/her own understanding of the system. For better undestanding, we provide correspondences (black clouds) between the BPM activities and the steps in the textual scenario (see Fig. 4.27). The business process starts by receiving a crisis coordinator (**`coord`**) and a crisis identifier (**`id`**).

It contains two branches, executed in parallel. The left branch of the business process deals with the internal logic of the CWR scenario. The context of the current crisis is built by retrieving information from the witness of the crisis : the process requests preliminary information about the crisis and then refines the information it receives through subsequent exchanges between the system and the witness. In parallel (the right branch), the system calls an external partner (**`PhoneCompany`**) to check the information given by the witness of a crisis and prevent false or erroneous reports. When the two branches join, that is, when the system considers the crisis report to be genuine, the system assigns an emergency level to the crisis and updates the crisis status to **`active`**.

4.3.3 Challenges and Synchronization Process

The complete CCCMS implementation contains thirteen business processes, describing hundreds of activities and thousands of relations between activities. Manual synchronization of the various views of such a large system can be challenging, time–consuming and error–prone. This section highlights situations in which checking and maintaining consistency across models can benefit from the use of automatic synchronization mechanisms.

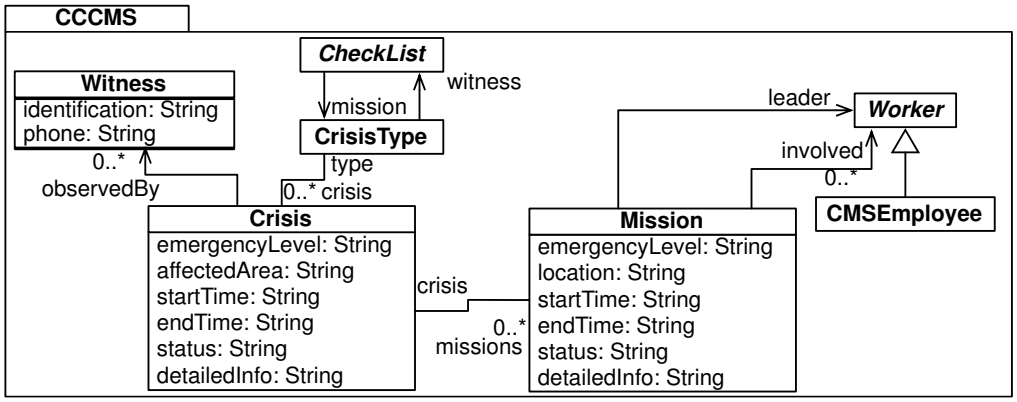
Since CD_D and BPM are defined by independent experts ($e_d \neq e_b$), one can encounter situations where types from the behavioral model (BPM) and types from the structural model (CD_D) diverge. We illustrate these divergences with examples from Section 4.3.2 below :

S_1**–Name Mismatch :** The business expert misspells a concept that already exists in CD_D. In Figure 4.28(b), e_d uses a **`CheckList`** type whereas e_b uses a **`Crisis-CheckList`** type. This situation illustrates naming conflicts that often occur across different views of the same system. For instance, the PROMPT [NM00] approach for aligning ontologies addresses this kind of conflicts among others.

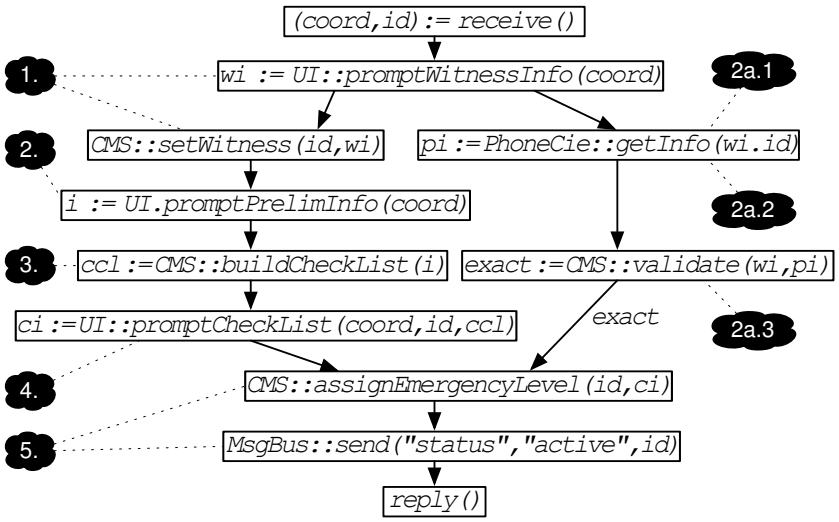
S_2**–Concept Enforcing :** The business expert uses data collected from an external partner, which are unknown from the domain point of view. In Figure 4.28(b), e_b uses information collected from the external agency *PhoneCompany* that is unkown to e_d and thus not modeled in the CD_D. This situation identifies the need to introduce externally defined artefacts (*i.e.*, provided by partner services) to the CD_D.

S_3**–Concept Usages :** The business expert uses his/her own data structure, *i.e.*, uses concepts defined in CD_D in an unforeseen way. In Figure 4.28(b), e_b uses a **`PreliminaryInformation`** concept in Activity 2. Since the original scenario indicates that the **`Coordinator`** should manipulate the location and type of the **`Crisis`**, we consider that e_b aggregated several artifacts already defined in CD_D (namely the location of the crisis and its type) in a single object for practical reasons. This situation illustrates how specific usage of data in a BPM can improve the CD_D.

Clearly, the synchronization of both CD_D and BPM is not a trivial problem. We identify two challenges related to these situations : *(i)*the automatic identification of



(a) Domain model (CD_D), extract.



Var.	Type
coord	CMSEmployee
id	String
wi	Witness
i	PreliminaryInformation

Var.	Type
ccl	CrisisCheckList
ci	CrisisInformation
pi	PhoneInformation
exact	boolean

(b) Business process model (BPM), graphical representation

We use here the graphical representation defined by ADORE [MBFF10] to represent business processes. Boxes represent activities (e.g., message reception, service invocation), and arrows represent causality relations (i.e., the associated partial order). A wait relation ($\mathtt{a} \rightarrow \mathtt{b}$*) means that* b *will wait for the end of* a *to start its own execution. A guard relation (*$\mathtt{a} \xrightarrow{v} \mathtt{b}$*) strengthens the wait semantics, and conditions the start of* b *to the value of* v*. Relations are combined using a conjunctive semantics* ($\wedge$).

FIGURE 4.28 – Initial model artifacts, proposed by experts.

such divergences (C_1) and *(ii)*the capture of resolution strategies and their automated propagation across models in the synchronization process (C_2). Figure 4.29 illustrates our approach that tackles these two challenges.

The first step of the process extracts data from the set of available BPM to derive a class diagram (CD_I) which contains all the concepts manipulated by this set of processes (1). Then, we use a *divergence detection* algorithm to identify occurrences of the situations (S_i) that we discussed previously (2). The detection of divergences leads to a phase of negotiation between experts from the domain and experts from the business process. Experts should consent on identifying *strategies* to resolve divergences (3) and to ultimately perform an accurate synchronization of CD_D and BPM. The last step of the process (4) propagates the resolution strategies using a dedicated algorithm (*strategies propagation*), which automatically applies changes in both CD_D and BPM.

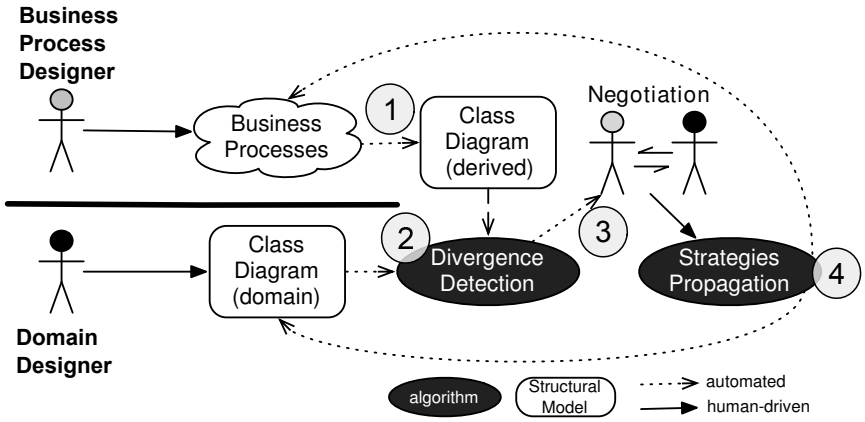


FIGURE 4.29 – SOA Models Synchronization : Process Overview

4.3.4 Identifying Model Divergences

This section presents the first two steps of the model synchronization process and the formalization of the divergence detection mechanism.

4.3.4.1 Naive Synchronization with Merge

The first step of the process extracts data from the BPM to derive a class–diagram (CD_I). The generation procedure visits all available business processes and extracts the types of all the declared variables.

Merging CD_I with CD_D using model composition techniques such as Kompose [FFR+07], produces a naive alignment of both models (see Figure 4.30). Naive alignment relies on an element matching process based on names. Elements with equivalent names are unified into a single element. For instance, the **`CMSEmployee`** element has been found in both CD_D and CD_I and therefore the merged model contains a single unified **`CMSEmployee`** element. Though simple, the naive alignment cannot align concepts that have different names. The default behavior of Kompose when such name–mismatches occur is to include the elements that do not match in the merged model. For instance, **`PreliminaryInformation`** is a concept from CD_I with no candidate match in CD_D.

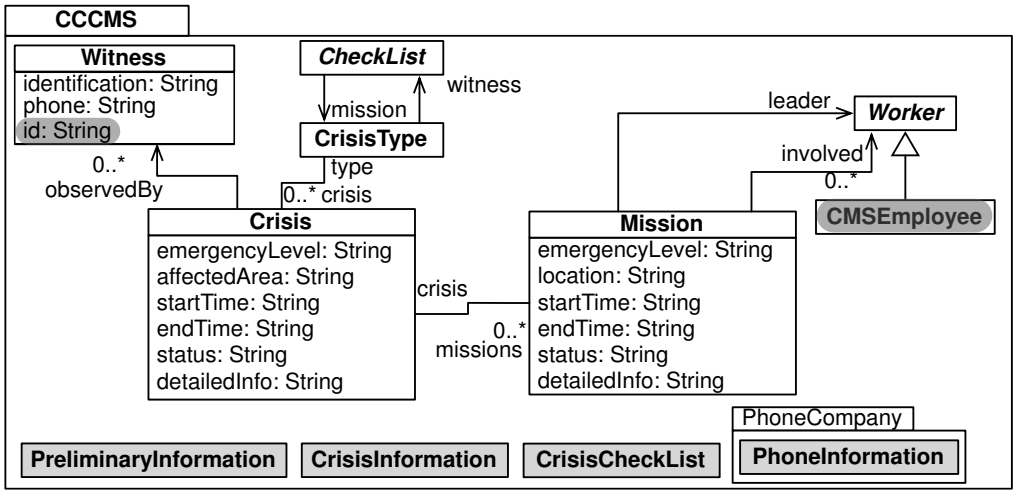


FIGURE 4.30 – Merged model : CD_D (white) $\oplus$ CD_I (gray)

We modified the default behavior of Kompose to record every operation used to produce the merged model. This record is analyzed to *(i)*validate every element that is automatically merged (*e.g.*, **`CMSEmployee`**) and to *(ii)*detect divergences between CD_D and CD_I.

4.3.4.2 Divergence Detection Mechanism

The analysis of the recorded operations leads to the detection of two kinds of divergences :

Point-of-view divergences occur when a model element from CD_I has no equivalent counterpart in CD_D (*e.g.*, **`PhoneInformation`**).

Structural divergences occur when a model element from CD_I has an equivalent counterpart in CD_D but the properties of the model element do not match with the properties of the corresponding model element in CD_D (*e.g.*, a "public" model element in CD_I is "private" in CD_D).

The divergence detection mechanism uses a matching operator and a set of signatures to compare a model element with another one. Let *match* be the predicate that checks if a model element of CD_I is equivalent to a model element of CD_D. With this match predicate, we formalize the kind of divergences as follows :

- *Point-of-view Divergence* refers to a model element in CD_I that has no equivalent model element in CD_D : $b \in CD_I s.t. \nexists\ d_i \in CD_D,\ match(b, d_i)$.
- *Structural Divergence* refers to a model element in CD_I that has equivalent model element in CD_D but whose properties do not match.

We formalize structural divergences according to the definitions provided by Barais *et al.* [BKB+08]. We defined two rules, used to reify the Class signature and the Property signature.

Class Signature. The signature of a Class encompasses its *identifier*, its *modifier*, possible *superclass*es and its *usage*. In the OO paradigm, the *category* and the *visibility* of classes provide additional information on how we may use these classes in a given OO program. A class is *internal* when it participates in calling internal services either as a value or as the type of a parameter of a service. For all other usages, we consider the class as *mixed*.

$$Class^{sig} = (Identifier, Modifier, Superclass, Usage)$$
$$Modifiers \in \{Category, Visibility\},\ Category \in \{abstract, concrete, final\}$$
$$Visibility \in \{private, protected, public\},\ Usage \in \{internal, mixed\}$$

CD_I reflects the usage of the class definitions at runtime and thus, classes are necessarily *concrete*, *public* with no *Superclass*es. In other words, we detect a divergence (c1) when a class in CD_I has an equivalent class in CD_D that is not *public* such that :

$$(c1)\ match(C_B, C_D) \wedge Visibility_{C_D} \neq public \tag{4.2}$$

Usage refers to the class usage in the business processes. This definition has an impact on the process of deriving CD_I : *(i)*classes that do not participate in calling an internal service are not captured by the data structure extraction process since we cannot modify the definition of a class provided by an external partner for compatibility reasons ; *(ii)*classes that are used both within internal and external services are *mixed*. They can only be enriched with additional information that cope with the initial definition of the class.

Regarding *Usage*, we detect a divergence (c2) when the usage of a class in CD_D is internal whereas an equivalent class is *mixed* in CD_I such that :

$$(c2)\ match(C_B, C_D) \wedge Usage_{C_D} = internal \wedge Usage_{C_B} = mixed \tag{4.3}$$

Property Signature. The signature of a property encompasses its *Identifier*, its scope of use (*Static*), its *Type* that is either a *Class* or a *Datatype* and its *Access*.

$$Property^{sig} = (Identifier, Static, Type, Access)$$
$$Static \in \{static, nonstatic\},\ Type \in Class \cup Datatype$$
$$Access \in \{read, write, rw, no\}$$

The first divergence (p1) that we may detect is if the two properties that we matched in CD_I and in CD_D have different types such that :

$$match(P_B, P_D) \wedge (p1)\ Type_{P_D} \neq Type_{P_B} \tag{4.4}$$

A property is *static* if it is common to all instances of this property and it is *nonstatic* otherwise. Properties that are used in BPM are necessarily *nonstatic* and thus we may detect the following divergence (p2) :

$$match(P_B, P_D) \wedge (p2)\ Static_{P_D} = static \tag{4.5}$$

Among these usual OO characteristics, we propose an additional *access* characteristic which determines how a property is accessed in BPM : *read* means that the property is only read by a service ; *write* means that the property is only written by a service ; *rw* means that the property is read and written by one or more services ; *no* is used in other cases. For instance, the property *id* of a **`Witness`** in Figure 4.28(b) is a *read* property since the property is read in activity `2a.1` and never written in any other activity. From this definition, we may detect two divergences : (p3) a property in CD_D is never accessed (*no*) or (*p3'*) a property in CD_D is not *rw* and an equivalent property in CD_I is accessed differently such that :

$$(p3)\ Access_{P_D} = no \ \vee (p3')\ (Access_{P_D} \neq rw \wedge Access_{P_D} \neq Access_{P_B}) \tag{4.6}$$

The formalization of the structural divergences allows propagating changes automatically in the CD_D regarding the structure and data of the BPM. Resolution of point–of–view divergences requires additional resolution strategies proposed by experts to solve the divergence situation presented in Section 4.3.3.

4.3.5 Application of the Unified Framework

4.3.5.1 Overview of the Model Composition Framework Customization

The definition of a new model composition framework for model synchronization follows the process of customization illustrated on Figure 3.6 in Section 3.2.1. Figure 4.31 illustrates the definition of a model composition framework for model merging.

Synchronization is another kind of model composition activity that requires to capture changes between an arbitrary number of models and propagates changes if necessary to maintain global consistency among the various software artifacts. Model synchronization is composed of two processes that synchronize structural divergences and point–of–view divergences from a set of resolution strategies. Details are provided in Section 4.3.6.

We propose a model–alignment language for model synchronization (see Section 4.3.5.2) to allow experts to propose resolution strategies between model elements. Automatic propagation of resolution strategies (see Section 4.3.6) allows the synchronization of both the domain model and the business model.

4.3.5.2 A Specific Model–Alignment Language for Model Synchronization

For the purpose of this synchronization operation, we build a specific model–alignment language and include the following interpretations :

- ReplaceStrategy to tackle structural divergences. Automatic detection of structural divergences would provide input data to the **`joinpoint()`** function.
- SimStrategy to deal with Name–mismatch situations by renaming selected concepts.

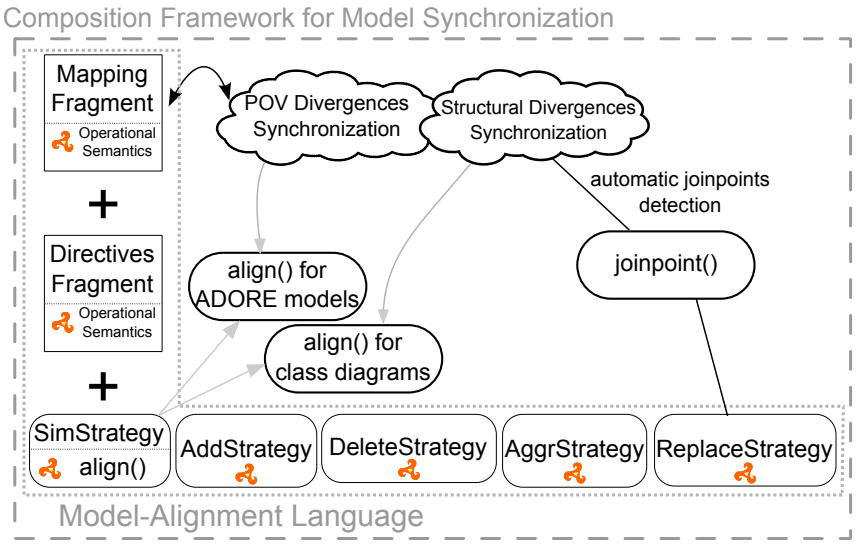


FIGURE 4.31 – Customization of the generic process to build a model composition framework for model synchronization.

– AddStrategy, DeleteStrategy, AggrStrategy and AugmentStrategy to handle concept enforcing and concept usage situations, *i.e.*, to replace existing structural information with what experts find more adequate.

Figure 4.32 shows the specific model–alignment language that we use in this experiment.

4.3.5.3 Proposing and Automating Resolution Strategies

In the global process of the approach, the negotiation step allow experts to propose strategies to resolve divergences between the CD_D and the BPM. This negotiation step produces a definition of mappings between model elements from BPM and model elements from CD_D and the selection of specific interpretations for each mapping. We use the specific model–alignment language for model synchronization proposed in Section 4.3.5.2 and shown in Figure 4.32. With this subset of the ModMap language, we propose mappings for the synchronization of business process models with models of the domain as illustrated in Figure 4.33.

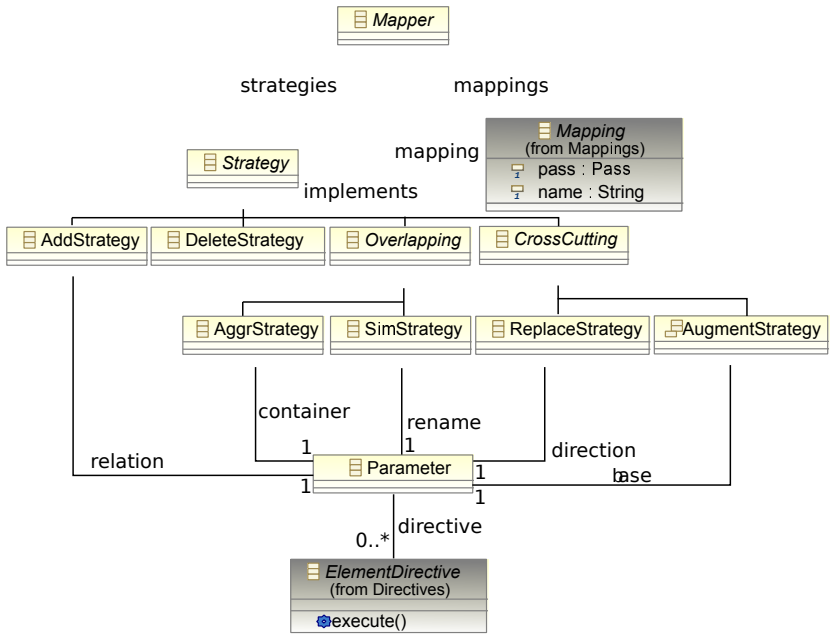


FIGURE 4.32 – Subset of the ModMap language for Model Synchronization

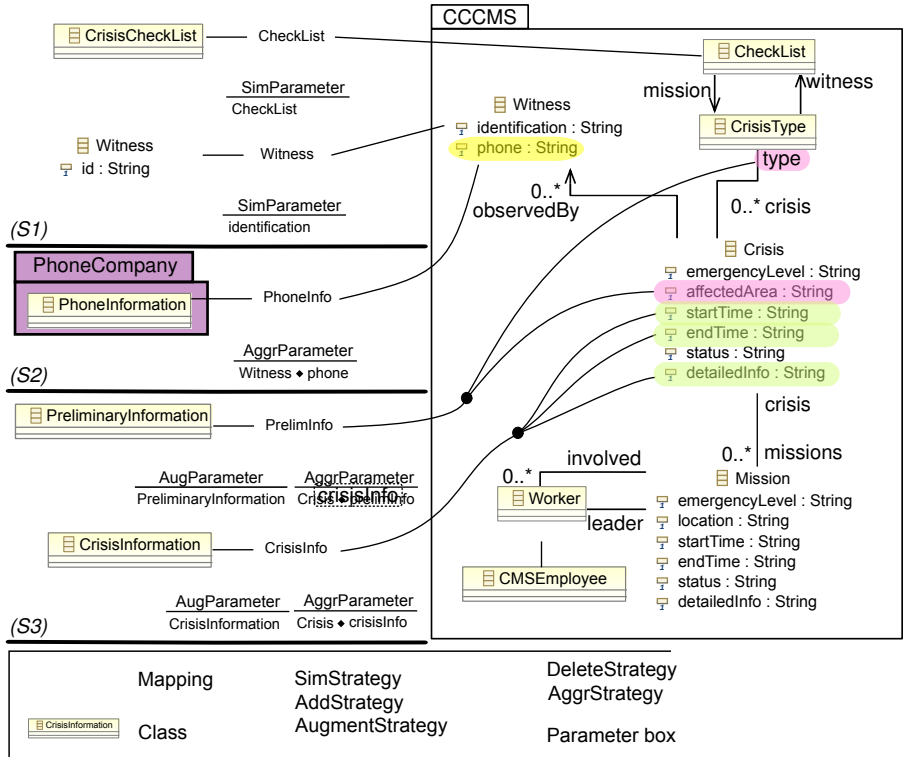


FIGURE 4.33 – Model of mappings for the synchronization of the BPM and the CD_D.

4.3.6 Propagation of the Resolution Strategies

The negotiation phase is important for experts to come to an agreement about how to deal with divergences in views. We capture their decisions in ModMap to allow automatic propagation of resolution strategies across models.

The purpose–specific processing reads each resolution strategy captured in the ModMap model and automatically produces a set of operations on both CD_D and BPM to synchronize the views.

In the following sections, we illustrate the interpretation of each resolution strategy with examples from the case study.

4.3.6.1 Name–Mismatch Strategy

The resolution of name–mismatches is straight-forward. The propagation process identifies every occurrences of a given name and replaces it with the name provided by the experts. The details of the propagation are discussed in the next subsections for both CD_D and BPM.

Domain model synchronization. We use the language of directives provided by the Kompose tool to rename model elements in CD_D. We adapted the Kompose tool to execute directives on a single model. Listing 4.7 lists the directives that the Kompose tool executes for modifying the name of **`CheckList`** in CD_D.

```
Directives{
  domainmodel::CheckList.name := "CrisisCheckList"
}
```

Listing 4.7 – Kompose directives for renaming the CheckList class of the domain model CD_D

Business process synchronization. We use a formal representation of business processes models, based on many-sorted first order logic [Mos10]. Thus, one can use logical substitution ($\theta = \{x \leftarrow x'\}$, [Sti81]) to replace in a given model m all occurrences of x by x'. We denote a $m\theta$ the model obtained after substitution. When several substitutions $\Theta = \{\theta_1, \ldots, \theta_n\}$ need to be performed on the same model, we denote as $m\Theta$ their parallel application on m. In the context of name mismatch strategies, the engine will generate the set of substitutions necessary to perform all the expected alignments : $\Theta = \{w.identification \leftarrow w.id\}$. Denoting as $\{bp_1, \ldots, bp_n\}$ the available business processes in the system, the enhanced SOA is therefore defined as $\{bp_1\Theta, \ldots, bp_n\Theta\}$.

4.3.6.2 Concept Enforcing and Concept Usage Strategies

The resolution of concept enforcing and concept usages situations may rely on a large number of operations for propagating changes. The details of the propagation are discussed in the next subsections for both CD_D and BPM.

Domain model synchronization Synchronization of CD_D for concept enforcing and concept usages relies on a set of Kompose directives to modify CD_D. We adopt two interpretations that are driven by the arity of the mapping relationship :

- When a mapping relationship relates only two model elements, the model element from CD_D is removed, the model element from CD_I is added to CD_D and a UML relation is created from the container of the initial model element from CD_D to the new model element in CD_D. For instance, experts decided to discard the **phone** property of the class **Witness** and use **PhoneInformation** instead. Property **phone** is removed from the class **Witness** and we create a new containment relation between **Witness** and **PhoneInformation**. This relation is named against the parameter of the replacement strategy.
- When a mapping relationship relates more than two model elements, the synchronization process is almost the same except that the model element from CD_I is considered as the container of the model elements from CD_D. Thus, we *move* the model elements from CD_D into the new model element in CD_D. For instance, experts agreed on using **PreliminaryInformation** instead of the two properties **type** and **affectedArea** from the class **Crisis**. **PreliminaryInformation** is thus enriched with the two properties **type** and **affectedArea** and a new containment relation is created between **Crisis** and **PreliminaryInformation**.

Listing 4.8 lists the directives that are applied on CD_D for replacing the **phone** property of the class **Witness** with **PhoneInformation**.

```
Directives{
  /*Creates a new PhoneInformation class
    and removes existing phone attribute
    in Witness*/
  create Class as $pi
  $pi.name = "PhoneInformation"
  destroy domainmodel::Witness::phone
  //Creates the phone relation
  create Association as $phone
  $phone.name = "phone"
  create Property as $phone_src
  $phone_src.aggregation =
   domainmodel::AggregationKind::
    #composite
  $phone_src.upper = 1
  $phone_src.type = domainmodel::Witness

  create Property as $phone_tgt
  $phone_tgt.upper = 1
  $phone_tgt.type = $pi

  $phone.memberEnd + $phone_src
  $phone.memberEnd + $phone_tgt
  /*Adds the PhoneInformation class and
    the phone relation*/
  domainmodel::packagedElement + $pi
  domainmodel::packagedElement + $phone }
```

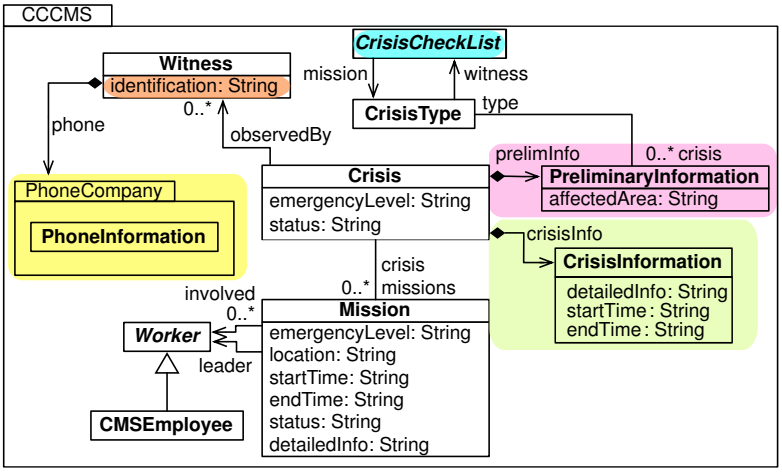
Listing 4.8 – Kompose directives for integrating PhoneInformation in the domain model CD_D

Business process synchronization The propagation of strategies for the resolution of concept enforcing and concept usage situations relies on logical substitution to propagate the new accesses (*e.g.*, $\{pi \leftarrow wi.phone\}$ to replace the variable *pi* by an access to the attribute *phone* contained in the variable *wi*). However, such replacements impose that we retrieve the "container" variable (*e.g.*, *wi*) that is necessary to access a specific property (*e.g.*, *phone*). Synchronization of **`PhoneInformation`** and **`phone`** illustrates the situation where the "container" variable already exists. Thus we use this variable to access to the phone information of a **`Witness`** and substitutions are propagated. When the "container" variable is not already available, we ask the experts how to initialize this "container" in BPM. After synchronization of **`PreliminaryInformation`** with **`type`** and **`affectedArea`**, **`PreliminaryInformation`** is contained by a **`Crisis`** object. Since no **`Crisis`** object is available in the initial process, experts propose the invocation of the **`getCrisis`** operation exposed by the **`CMS`** service. This operation stores a **`Crisis`** object in a variable *c*. This invocation is automatically inserted into the business process by the ADORE engine (after the **`receive`** acitivity) and default substitutions are executed.

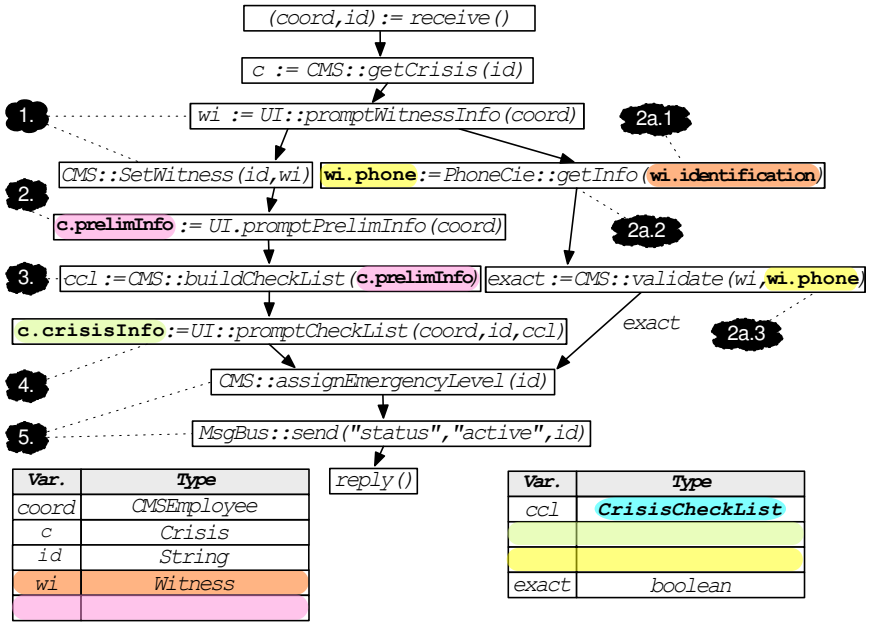
4.3.7 Discussion

This experiment proposes an approach for synchronizing business processes with domain models developed by different teams working on the same system, in the context of SOA. This approach leverages and integrates model composition and generative techniques to automate significant aspects of the synchronization process. Model synchronization is a second activity that fits in the definition of model composition that we propose in the unified theoretical framework : the definition of mappings and the selection of specific strategies help to *(i)*focus human intervention on the detection of divergences that require human judgment and experience, and to *(ii)*automate the process of propagating of the proposed resolution *strategies* to reduce the global effort of model synchronization.

This second application of the unified theoretical framework on model composition beyond model merging and model integration consolidates the intuitive claim that model composition encompasses many operations on models and that the unified theoretical framework is able to support these diverse operations. Further work on the relationship between software life–cycle activities and specific model–alignment languages should help proposing and developing generic model composition frameworks dedicated to specific activities and operations on models.



(a) Aligned Domain model



(b) Aligned Business process model

FIGURE 4.34 – Aligned models resulting from the propagation of the resolution strategies.

Conclusion

The growing complexity of designing and building software has transformed the state of practice in both industry and academy. Software development life–cycle in a multi–modeling environment with multiple actors is a prevailing trend that involves designing, analyzing and building multiple model–based artifacts. Academics interest for these topics of composing models in the large definition of model management that encompasses a large range of operations on models is still both worthwhile and challenging.

Adoption of the MDE practices in industry requires specific organizational, managerial and social factors to be successful endeavors [HRW11]. Still, progressive adoption brings new situations and new requirements that research is eager for finding adequate solutions.

The multitude of model composition frameworks available is a proof of success in the design of model composition frameworks that tackle specific situations and context. While the development of model composition framework is valuable and successful in a given number of situations, the development of techniques and tools of industrial quality is hindered by the incapacity of these frameworks to be easily adapted and reused over different situations and for different purposes.

I.1 A Decomposition of the Definition of Model Composition

Our contribution to this field is to propose a novel definition of model composition both to enhance the global understanding of this operation and to broaden the scope of application of model composition approaches in the MDE community.

Previous work about model composition that we discuss in Chapter 1, proposes various operators that handle a large range of operations on model, claims that mapping (*i.e.*, correspondences, relationships between model elements) is the real surplus value in the definition of meaningful model composition approaches, or combine both mappings and operations in single modeling artifacts. From the observation of these attempts to classify existing model composition approaches, we observe that mappings and interpretations of these mappings have a strong influence in the characterization of model composition approaches.

The main contribution of this thesis is thus to propose a novel definition of model

composition as a pair of a mapping and a set of interpretations. A mapping is a set of explicit relationships between sets of models or sets of model elements. Interpretation provides semantics to a mapping and participates to a specific model composition purpose.

I.1.1 Literature Review and Observations

The intuitive decomposition of model composition as pairs of mappings and interpretations is supported by categories for mappings and categories for interpretations. The category of mappings includes operator–based, pattern–based, rule–based, constraint–based, model–based and delta–based representations of a mapping. The category of interpretation includes fifteen kinds of interpretations divided in three categories which cover overlapping models, cross–cutting models and interacting models. We evaluate the relevance and precision of the two categories by conducting a systematic literature review. The systematic literature review leads us to explore proceedings of the major software engineering and MDE international and national conferences or journals to capture the state of practice about model composition in software engineering.

Research objectives, review protocol and results are presented in Chapter 1 following guidelines proposed by Biolchini *et al.* [BMA+05].

Proposing categories for both mappings and interpretations, validated with an empirical study by conducting a systematic literature review, we provide experts with an interpretive lens for model composition techniques analysis. This interpretive lens is a new apparatus in the early stages of software engineering and system development to compare and to select or to adapt an adequate existing model composition frameworks to cope with requirements, or to start designing and building a new model composition framework if necessary.

I.1.2 Formal Definition of Mappings and Interpretations

The intuitive proposition of the novel definition of model composition as a pair of mapping and interpretations is based on analogies with structures in mathematical logic and linguistics.

Structures in mathematical logic helps to bridge the gap between our vision and a grounded theory that already separates these concepts and that defines explicitly the relationships between these concepts. The exact nature of the relationships between a mapping and interpretations of this mapping is however dependent from a number of parameters that are barely captured in the current state of practice. We use the parallel with linguistics to explore how the purpose of a model composition and how the human contribution to this purpose influences both the nature of the relationships between a mapping and interpretations and the global intention of a given model composition framework.

In linguistics, a sign is composed of a signifier and a signified that respectively are the form which a sign takes and the concept it represents. Considering a mapping as

the form of the model composition and interpretations as the concept it represents, we use the concepts of denotation and connotation to explore further the relationship between a mapping and its interpretations. Denotation is the generic meaning of a pairing of a mapping and an interpretation of this mapping : model elements from one model relate to model elements from another model. Connotation refines the meaning of relate to take into account the context and the purpose of the mapping for a specific model composition goal.

Supporting this intuitive definition of mapping and interpretation, we propose a unified theory for model composition in Chapter 2. The theory proposes a formal definition of the various kinds of mappings and the various kinds of interpretations and leads to the definition of a framework for unifying model composition activities (see Chapter 3).

I.1.3 A Framework for Unifying Model Composition Activities

Chapter 3 presents the MODel MAPping (ModMap) tool that supports the definition of mappings and their interpretations for producing effective model composition languages and concrete model composition operators. The concrete implementation of the ModMap tool proposes *(i)*a language for the design of mappings between models and meta–models ; *(ii)*the operational semantics for each kind of interpretations ; *(iii)*a concrete syntax to ease the specification of mappings and the selection of specific interpretations and *(iv)*the global process for building new model composition operators. As an illustration, we present the process of building a specific model composition framework for model merging and the process of building a specific model composition framework for model integration.

I.1.4 Validation and Experiments

We evaluate the adequateness and relevance of the modeling framework for building model composition frameworks through three experiments : *(i)*we use the theoretical framework to unify four existing model merging techniques and propose a unique kernel for model composition ; *(ii)*we demonstrate the applicability of the framework on the integration of legacy API for the configuration and management of heterogeneous video and broadcasting equipments in collaboration with industrial partners from Technicolor [1] ; *(iii)*we demonstrate the applicability of the framework on the synchronization of heterogeneous models in the context of modeling service–oriented architectures (SOA).

In the next section, we propose perspectives for both improving the current state of the art about model composition and for further research.

1. http ://www.technicolor.com/en/hi/technology/research-and-innovation-centers/rennes

II.2 Perspectives

We provide a modeling framework that supports the definition of mappings and the definition of interpretations for building specific model composition operations. Providing semantics for mappings and interpretations, the modeling framework is a tool–kit that experts can customize to fit specific needs and to answer to specific situations.

We strongly believe that the main contribution of this thesis is an important step in fulfilling our vision of shifting from model composition as an operator that targets a specific purpose in a specific context to model composition as an operation that allows controlled customization and variability of the model composition process. The contribution of this thesis opens new tracks and fields or research that need additional and in–depth exploration.

II.2.1 Extension of the Systematic Literature Review

In the context of this thesis, we focus effort on the characterization of the key concepts of correspondence and interpretation in model composition. The systematic literature review results presented in Chapter 1 reflect the variability of correspondences and interpretations that existing model composition approaches put into action to achieve a specific purpose in a specific context. Along with the evolution of correspondence and interpretation categories, we envisage three further developments to this piece of work.

II.2.1.1 Influence of Software Development Activities

We are confident that software development activities influence the definition and the specification of both correspondences and interpretations. We consider worthwhile to extract further information both about the relation between correspondences and software activities, and about the relation between interpretations and software activities. Such information would help *(i)*characterizing which pair of a correspondence and an interpretation is relevant for a specific software activity and eventually *(ii)*providing a list of model composition approaches that supports such software activities. We presented preliminary data to achieve this goal but additional analysis is actually required.

II.2.1.2 Existing Model Composition Approaches Adaptation

In the current presentation of the systematic literature reviews, several model composition approaches support several software development activities. In the process of building new model composition operations, we believe that these techniques that already tackle various model composition challenges are good candidates for reuse and adaptation.

II.2.1.3 Classification Completeness

In the light of the presentation of the systematic literature review results in tables, we observe that some pairs of correspondence/interpretation and some pairs of correspondence/software activity have no candidate article identified. Towards providing a complete characterization of model compsition, additional research is needed to evaluate why such pairs have not been proposed yet and what scientific challenge may lie beneath.

II.2.2 Model Composition as a first–class Entity in MDE

Model composition in multi–modeling environments for software engineering is a key activity. Bézivin *et al.* believe that model composition techniques should be given a first–class status, similarly to what has be done with model transformation techniques [BBDF+06]. In the extension of UML with model composition semantics, Siobhán Clarke proposes that a "... subject–oriented design model supports a new kind of design construct, called a composition relationship that supports the specification of how design models should be composed" [Cla02, §3, p.6]. We envisage also to manipulate model composition as a first–class entity and propose two perpectives in this directions.

II.2.2.1 About a "composable" relationship in the Meta–Object Facility (MOF)

Observing that model composition is a relevant technique for supporting a wide range of activities on models, we may consider model composition as a key concept in MDE. The next step in the definition of model composition is to provide an abstract representation that we promote a level of abstraction up so that designers can define model composition as part of their model design. Model composition thus become a first-class entity at the meta-meta-level of MOF.

In other words, a new kind of relationship should be added to the Property Type of MOF (see Figure II.1) with the following semantics that mimics the semantics of MOF properties [OMG10a, §12.5, p.45] : **Property : :isComposition==true**

- An object may be *composed* with multiple objects
- Cyclic composition is valid : order of composition is determined by the concrete implementation of the property instance.
- Any composition property should be provided with specific semantics using an action language such as Kermeta.

The semantics of the new *composition* relationship would include the mapping and interpretation DSML semantics that are presented in Chapter 3. If available, such a relationship would ease the definition of model composition operation between objects and would allow to take into account such operation in systems and languages design.

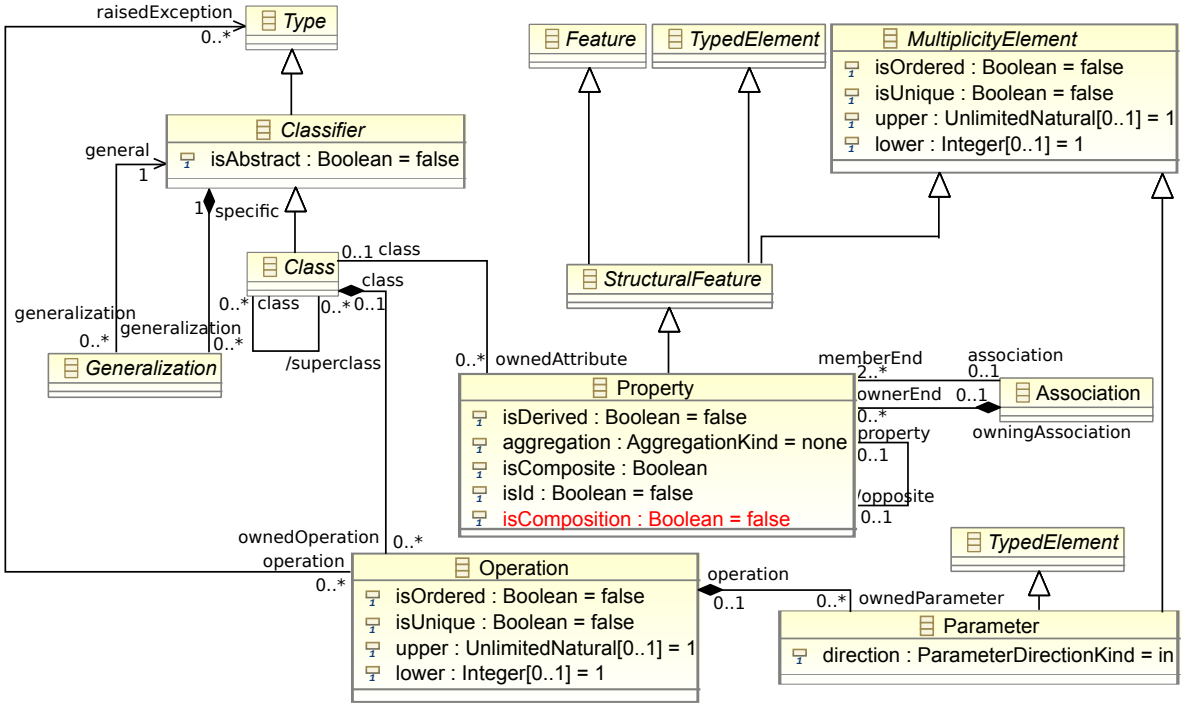


FIGURE II.1 – MOF core metamodel : EssentialMOF classes adapted from [OMG10a, §12.2, p.40]

II.2.2.2 High-Order Composition

A direct consequence of promoting model composition in the meta-meta-level of abstraction is that it allows manipulating model composition as any other model element. This is a requirement for a higher–order application of the model composition operator. Thus, our proposition should support composition of model compositions or in other words : higher–order model composition (HOC). Similarly to Higher–Order Transformations (HOT) [TJF+09], an HOC takes a model composition as an input and produces a model composition as an output. For instance, an HOC may define the model composition of two DSML $DSML_1$ and $DSML_3$ from the model composition of the two DSML $DSML_1$ with $DSML_2$ and the model composition of the two DSML $DSML_2$ with $DSML_3$. This opens new paths for research and application in synthesizing model composition frameworks from existing techniques.

II.2.3 Application and Future of ModMap

II.2.3.1 Extending the scope of application of ModMap

Preliminary results in generalizing model merging allow us to redefine four existing techniques by proposing a unique match module and a unique merge module. Further work is twofold : *(i)*propose additional experiments about various model com-

position techniques to assess scalability, relevance and limits in the application of the theoretical framework for unifying model composition techniques ; *(ii)*build a repository of libraries that support various purposes to provide reusable and extensible on-the-shelf model management components.

Similarly, we present in Chapter 3 two case studies that deal with model integration and model synchronization for which the ModMap framework is adequate. We think about using ModMap for the specification of additional software life–cycle activities such as model derivation, model orchestration, model consistency checking or even model reconfiguration.

II.2.3.2 Collaborations

In the light of the MOPCOM–I project and the successful application of ModMap to the Technicolor case studies, we are exploring future collaborations with several partners such as France Telecom and Thales Systèmes Aéroportés on different model mapping issues. We are currently carrying out preliminary discussions with France Telecom about the definition of mappings between WebServices and the functionalities of a given system to ease and automate the design of WebServices interfaces with regard to the service that they provide. Thales Systèmes Aéroportés interest in proposing mappings is threefold : *(i)*provide mappings model–to–model to help specifying transformations between specific DSML on “mission planning and debriefing” case study ; *(ii)*provide mappings between semi–structured documents and a specific DSML to provide serialization capabilities ; *(iii)*provide mappings between a specific DSML and the representation of the environment to help improving the definition of software interfaces.

Besides the direct application of ModMap for these specific case study, the ModMap framework needs *(i)*maturation for being usable in a industrial context and needs *(ii)*metrics to evaluate how it fits in particular contexts.

Glossary

AMW ATLAS Model Weaver. *vi*, *64*, *66*
AOM Aspect-Oriented Modeling. *9*, *33*, *40*, *92*, *141*
AOP Aspect–Oriented Programming. *7*, *130*
API Application Programming Interface. *3*, *111*, *129–133*, *136*, *137*, *139*, *157*
AST Abstract Syntax Tree. *45*

DSL Domain-Specific Language. *12*, *32*, *44*, *137*
DSML Domain-Specific Modeling Language. *xi–xiii*, *8*, *30*, *34*, *37*, *75–81*, *159–161*

ECore EMFCore. *8*, *90*
EML Epsilon Merging Language. *49*

GCFs Generic model composition frameworks. *2*, *63*, *65*
GPL General-purpose Programmation Language. *17*, *95*

JVM Java Virtual Machine. *136*

LOC Lines of Code. *137*

MDE Model-Driven Engineering. *1*, *5*, *7*, *8*, *29*, *72*, *129*, *155*, *156*, *159*
ModMap MODel MAPping. *83*
MOF Meta-Object Facility. *xi*, *8*, *9*, *128*, *159*

OCL Object-Constraint Language. *32*
OMG Object Management Group. *7*
OO Object–Oriented. *6*, *64*, *147*, *148*
ORM Object–Relational Mapping. *vi*, *64*, *66*

QoS Quality of Service. *37*
QVT Query/View/Transformation. *45*

SaaS Software as a Service. *29*

SE Software Engineering. *21*

SOA Service–Oriented Architecture. *111*, *153*

SoC Separation of Concerns. *6*, *43*

SPL Software Product Line. *9*, *29*, *32–34*, *40*

TGG Triple Graph Grammar. *48*

UML Unified Modeling Language. *xi*, *8*, *33*, *112*, *113*, *159*

Bibliography

[ACL+09] M. Acher, P. Collet, P. Lahire et al. « Composing Feature Models ». Dans : *Software Language Engineering*. Éd. par M. van den Brand, D. Gaševic et J. Gray. T. 5969. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 62–81. doi : 10.1007/978-3-642-12107-4_6. url : http://dx.doi.org/10.1007/978-3-642-12107-4_6.

[ACL+10] M. Acher, P. Collet, P. Lahire et al. « Managing Variability in Workflow with Feature Model Composition Operators ». Dans : *Software Composition*. Éd. par B. Baudry et E. Wohlstadter. T. 6144. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, p. 17–33. url : http://dx.doi.org/10.1007/978-3-642-14046-4_2.

[AEC+07] A. Anwar, S. Ebersold, B. Coulette et al. « Vers une approche à base de règles pour la composition de modèles. Application au profil VUML. » French. Dans : *L'Objet, Ingénierie Dirigée par les Modèles* 13.4/2007 (déc. 2007), p. 73–103. url : ftp://ftp.irit.fr/IRIT/MACAO/Coulette-etal-LObjet2007.pdf.

[AJT+09] S. Apel, F. Janda, S. Trujillo et al. « Model Superimposition in Software Product Lines ». Dans : *Theory and Practice of Model Transformations*. Éd. par R. Paige. T. 5563. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 4–19. doi : 10.1007/978-3-642-02408-5_2. url : http://dx.doi.org/10.1007/978-3-642-02408-5_2.

[ASM+10] M. Alférez, J. a. Santos, A. Moreira et al. « Multi-view Composition Language for Software Product Line Requirements ». Dans : *Software Language Engineering*. Éd. par M. van den Brand, D. Gaševic et J. Gray. T. 5969. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, p. 103–122. url : http://dx.doi.org/10.1007/978-3-642-12107-4_8.

[AT98] M. Aksit et B. Tekinerdogan. *Solving the Modeling Problems of Object-Oriented Languages By Composing Multiple Aspects Using Composition Filters*. AOP'98 Workshop Position Paper. 1998. url : http://wwwtrese.cs.utwente.nl/Docs/Tresepapers/FilterAspects.html.

[Ass11] Association for Computing Machinery. *ACM DL*. online. Juin 2011. url : http://portal.acm.org/.

[BA00] L. M. Bergmans et M. Aksit. « Composing Software from Multiple Concerns : A Model and Composition Anomalies ». Dans : *ICSE 2000 Workshop on Multi-Dimensional Separation of Concerns in Software Engineering*. 2000. URL : http://doc.utwente.nl/18812/.

[BBB+] W. Bast, M. Belaunde, X. Blanc et al. *MOF QVT final adopted specification*.

[BBDF+06] J. Bezivin, S. Bouzitouna, M. Del Fabro et al. « A Canonical Scheme for Model Composition ». Dans : *Model Driven Architecture - Foundations and Applications*. Éd. par A. Rensink et J. Warmer. T. 4066. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, p. 346–360. URL : http://dx.doi.org/10.1007/11787044_26.

[BBN+10] S. Bensalem, M. Bozga, T.-H. Nguyen et al. « Compositional verification for component-based systems and application ». Dans : *Software, IET* 4.3 (juin 2010), p. 181 –193. ISSN : 1751-8806. DOI : 10.1049/iet-sen.2009.0011.

[BCE+06] G. Brunet, M. Chechik, S. Easterbrook et al. « A manifesto for model merging ». Dans : *Proceedings of the 2006 international workshop on Global integrated model management*. GaMMa '06. New York, NY, USA : ACM, 2006, p. 5–12. DOI : http://doi.acm.org/10.1145/1138304.1138307. URL : http://doi.acm.org/10.1145/1138304.1138307.

[BCR05] A. Boronat, J. Carsí et I. Ramos. « MOMENT : a formal MOdel manageMENT tool ». Dans : *Summer School on Generative and Transformational Techniques in Software Engineering* (2005). URL : http://moment.dsic.upv.es/index.php?option=com_docman\&\#38;task=doc_download\&\#38;gid=36.

[BCR+07] A. Boronat, J. A. Carsí, I. Ramos et al. « Formal Model Merging Applied to Class Diagram Integration ». Dans : *Electron. Notes Theor. Comput. Sci.* 166 (jan. 2007), p. 5–26. DOI : 10.1016/j.entcs.2006.06.013. URL : http://portal.acm.org/citation.cfm?id=1223344.1223436.

[BE09] L. Bendix et P. Emanuelsson. « Requirements for Practical Model Merge - An Industrial Perspective ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par A. Schürr et B. Selic. T. 5795. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 167–180. DOI : 10.1007/978-3-642-04425-0_13. URL : http://dx.doi.org/10.1007/978-3-642-04425-0_13.

[BHP00] P. A. Bernstein, A. Y. Halevy et R. A. Pottinger. « A vision for management of complex models ». Dans : *SIGMOD Record (ACM Special Interest Group on Management of Data)* 29.4 (2000), p. 55–63. ISSN : 0163-5808. DOI : http://doi.acm.org/10.1145/369275.369289. URL : http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.3340.

[BKB+08] O. Barais, J. Klein, B. Baudry et al. « Composing Multi-view Aspect Models ». Dans : *Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on*. 2008, p. 43 –52. doi : 10.1109/ICCBSS.2008.12.

[BL73] D. E. Bell et L. J. LaPadula. « Secure Computer Systems : Mathematical Foundations and Model ». Dans : *The MITRE Corporation Bedford MA Technical Report M74244 May* 1.M74-244 (1973), p. 42.

[BLTN10] E. Brottier, Y. Le Traon et B. Nicolas. « Composing Models at Two Modeling Levels to Capture Heterogeneous Concerns in Requirements ». Dans : *Software Composition*. Éd. par B. Baudry et E. Wohlstadter. T. 6144. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, p. 1–16. doi : 10.1007/978-3-642-14046-4_1. url : http://dx.doi.org/10.1007/978-3-642-14046-4_1.

[BMA+05] J. Biolchini, P. G. Mian, Ana et al. *Systematic Review in Software Engineering*. Rap. tech. Technical Report RT-ES 679/05, Systems Engineering et Computer Science Department, COPPE/UFRJ, 2005.

[BS81] S. Burris et H. Sankappanavar. *A Course In Universal Algebra*. Springer-Verlag, 1981, p. xvi, 276.

[BSM+07] K. Balasubramanian, D. C. Schmidt, Z. Molnar et al. « Component-Based System Integration via (Meta)Model Composition ». Dans : *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*. 2007, p. 93 –102. doi : 10.1109/ECBS.2007.24.

[BTF05] I. Balaban, F. Tip et R. Fuhrer. « Refactoring support for class library migration ». Dans : *OOPSLA '05 : Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. San Diego, CA, USA : ACM, 2005, p. 265–279. isbn : 1-59593-031-0. doi : http://doi.acm.org/10.1145/1094811.1094832.

[BWH+08] N. Boucké, D. Weyns, R. Hilliard et al. « Characterizing Relations between Architectural Views ». Dans : *Software Architecture*. Éd. par R. Morrison, D. Balasubramaniam et K. Falkner. T. 5292. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 66–81. url : http://dx.doi.org/10.1007/978-3-540-88030-1_7.

[BWH10] N. Boucké, D. Weyns et T. Holvoet. « Composition of architectural models : Empirical analysis and language support ». Dans : *J. Syst. Softw.* 83 (11 2010), p. 2108–2127. issn : 0164-1212. doi : http://dx.doi.org/10.1016/j.jss.2010.06.011. url : http://dx.doi.org/10.1016/j.jss.2010.06.011.

[Bar08] C. Bartelt. « Consistence preserving model merge in collaborative development processes ». Dans : *CVSM '08 : Proceedings of the 2008 international workshop on Comparison and versioning of software models*. Leipzig, Germany : ACM, 2008, p. 13–18. isbn : 978-1-60558-045-6. doi : 10.1145/1370152.1370157. url : http://dx.doi.org/10.1145/1370152.1370157.

[Bar64] R. Barthes. « Éléments de sémiologie ». French. Dans : *Communications* 4.1 (1964), p. 91–135. issn : 0588-8018. doi : 10.3406/comm.1964.1029. url : http://www.persee.fr/web/revues/home/prescript/article/comm_0588-8018_1964_num_4_1_1029.

[Bel04] A. Belapurkar. *Use AOP to maintain legacy Java applications*. Mar. 2004. url : http://www.ibm.com/developerworks/java/library/j-aopsc2.html.

[Ber03] P. A. Bernstein. « Applying Model Management to Classical Meta Data Problems ». Dans : *Proc. First Biennal Conference on Innovative Data Systems Research, 2003. CIDR'03*. 2003.

[CBJ10] M. Clavreul, O. Barais et J.-M. Jézéquel. « Integrating Legacy Systems with MDE ». Dans : *ICSE'10 : Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering and ICSE Workshops*. T. 2. Cape Town, South Africa 2010, p. 69–78. url : http://www.irisa.fr/triskell/publis/2010/CLAVREUL10a.pdf.

[CCS11] CCSD-CNRS. *HAL*. online. Juin 2011. url : http://hal.archives-ouvertes.fr/index.php.

[CDK+07] F. Curbera, M. Duftler, R. Khalaf et al. « Bite : Workflow Composition for the Web ». Dans : *Service-Oriented Computing - ICSOC 2007*. Éd. par B. Krämer, K.-J. Lin et P. Narasimhan. T. 4749. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, p. 94–106. url : http://dx.doi.org/10.1007/978-3-540-74974-5_8.

[CDRP08] A. Cicchetti, D. Di Ruscio et A. Pierantonio. « Managing Model Conflicts in Distributed Development ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par K. Czarnecki, I. Ober, J.-M. Bruel et al. T. 5301. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 311–325. url : http://dx.doi.org/10.1007/978-3-540-87875-9_23.

[CJC11] C. Clasen, F. Jouault et J. Cabot. « Virtual Composition of EMF Models ». Anglais. Dans : *7èmes Journées sur l'Ingénierie Dirigée par les Modèles (IDM 2011)*. Lille, France 2011. url : http://hal.inria.fr/inria-00606374/PDF/VirtualModels_IDM2011.pdf.

[CMBF+11] M. Clavreul, S. Mosser, M. Blay-Fornarino et al. « Service–Oriented Architecture Modeling : Bridging the Gap Between Structure and Behavior ». Dans : *MODELS'11 : Proceedings of the 14th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. To appear.* 2011.

[CNM11] M. Chechik, S. Nejati et S. Mehrdad. « A Relationship-Based Approach to Model integration ». Dans : *Journal of Innovations in Systems and Software Engineering* (2011).

[CØV02] K. Czarnecki, K. Østerbye et M. Völter. « Generative Programming ». Dans : *Object-Oriented Technology ECOOP 2002 Workshop Reader*. Éd. par J. Hernandez et A. Moreira. T. 2548. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2002, p. 15–29. URL : http://dx.doi.org/10.1007/3-540-36208-8_2.

[CRE+08] A. Cicchetti, D. D. Ruscio, R. Eramo et al. « Automating Co-evolution in Model-Driven Engineering ». Dans : *Enterprise Distributed Object Computing Conference, IEEE International* 0 (2008), p. 222–231. ISSN : 1541-7719. DOI : http://doi.ieeecomputersociety.org/10.1109/EDOC.2008.44.

[CRR+07] R. Chitchyan, A. Rashid, P. Rayson et al. « Semantics-based composition for aspect-oriented requirements engineering ». Dans : *Proceedings of the 6th international conference on Aspect-oriented software development*. AOSD '07. Vancouver, British Columbia, Canada : ACM, 2007, p. 36–48. ISBN : 1-59593-615-7. DOI : http://doi.acm.org/10.1145/1218563.1218569. URL : http://doi.acm.org/10.1145/1218563.1218569.

[CSN08] K. Chen, J. Sztipanovits et S. Neema. « Compositional Specification of Behavioral Semantics ». Dans : *Design, Automation, and Test in Europe*. Éd. par R. Lauwereins et J. Madsen. Springer Netherlands, 2008, p. 253–265. ISBN : 978-1-4020-6488-3. URL : http://dx.doi.org/10.1007/978-1-4020-6488-3_19.

[CT90] K. M Chandy et S. Taylor. *A Primer for Program Composition Notation*. Technical Report 10. Pasadena, CA, USA : California Institute of Technology, 1990. URL : http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail\&id=oai%3Acaltechcstr%3A00000071.

[Cha08] D. Chandler. *Semiotics for Beginners*. Daniel Chandler (University of Wales, Aberystwyth), 2008. ISBN : 9781874166559. URL : http://dominicpetrillo.com/ed/Semiotics_for_Beginners.pdf.

[Cla02] S. Clarke. « Extending standard UML with model composition semantics ». Dans : *Sci. Comput. Program.* 44.1 (2002), p. 71–100. ISSN : 0167-6423. DOI : http://dx.doi.org/10.1016/S0167-6423(02)00030-8.

[Cur54] H. B. Curry. « The logic of program composition , in Applications Scientifiques de la Logique Mathématique : Actes du 2 ». Dans : *e Colloque International de Logique Mathématique, Paris - 25-30 Août 1952, Institut Henri Poincaré*. 1954, p. 97–102.

[DFB+05a] M. Didonet, D. Fabro, J Bézivin et al. « Applying Generic Model Management to Data Mapping ». Dans : *Proceedings of BDA 2005*. Saint-Malo, France 2005, p. 343–355.

[DFB+05b] M. Didonet, D. Fabro, J. Bézivin et al. « AMW : a generic model weaver ». Dans : *Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM05)*. 2005. url : http://www.sciences.univ-nantes.fr/lina/atl/www/papers/IDM_2005_weaver.pdf.

[DRMM+10] D. Di Ruscio, I. Malavolta, H. Muccini et al. « Developing next generation ADLs through MDE techniques ». Dans : *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*. ICSE '10. New York, NY, USA : ACM, 2010, p. 85–94. doi : http://doi.acm.org/10.1145/1806799.1806816. url : http://doi.acm.org/10.1145/1806799.1806816.

[Dij97] E. W. Dijkstra. *A Discipline of Programming*. 1st. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1997. isbn : 013215871X.

[EPK06] K.-D. Engel, R. F. Paige et D. S. Kolovos. « Using a Model Merging Language for Reconciling Model Versions ». Dans : *ECMDA-FA*. Éd. par A. Rensink et J. Warmer. T. 4066. Lecture Notes in Computer Science. Springer, 2006, p. 143–157. isbn : 3-540-35909-5.

[ES06] M. Emerson et J. Sztipanovits. « Techniques for metamodel composition ». Dans : *The 6th OOPSLA Workshop on Domain-Specific Modeling, OOPSLA 2006*. ACM, ACM Press, 2006, p. 123–139.

[Els11] Elsevier BV. *SciVerse ScienceDirect*. online. Juin 2011. url : http://www.hub.sciverse.com/action/home.

[FBB+07] F. Fleurey, E. Breton, B. Baudry et al. « Model-Driven Engineering for Software Migration in a Large Industrial Context ». Anglais. Dans : *MoDELS'07*. Nashville, TN, USA États-Unis 2007. url : http://hal.inria.fr/inria-00477566/PDF/fleurey07a.pdf.

[FBF+08] F. Fleurey, B. Baudry, R. France et al. « A Generic Approach for Automatic Model Composition ». Dans : *Models in Software Engineering*. Éd. par H. Giese. T. 5002. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 7–15. doi : 10.1007/978-3-540-69073-3_2. url : http://dx.doi.org/10.1007/978-3-540-69073-3_2.

[FDV07] D. Fabro, M. Didonet et P. Valduriez. « Semi-automatic model integration using matching transformations and weaving models ». Dans : *SAC '07 : Proceedings of the 2007 ACM symposium on Applied computing*. Seoul, Korea : ACM, 2007, p. 963–970. isbn : 1-59593-480-4. doi : http://doi.acm.org/10.1145/1244002.1244215.

[FEB06] J. M. Favre, J. Estublier et M. Blay. *L'ingénierie dirigée par les modèles : au-delà du MDA*. French. Hermes-Lavoisier, 2006.

[FFR+07] R. France, F. Fleurey, R. Reddy et al. « Providing Support for Model Composition in Metamodels ». Dans : *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*. 2007, p. 253. doi : 10.1109/EDOC.2007.55.

[FGF+08] M. Fritzsche, W. Gilani, C. Fritzsche et al. « Towards Utilizing Model-Driven Engineering of Composite Applications for Business Performance Analysis ». Dans : *Model Driven Architecture - Foundations and Applications*. Éd. par I. Schieferdecker et A. Hartman. T. 5095. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 369–380. url : http://dx.doi.org/10.1007/978-3-540-69100-6_26.

[GG10] A. Goncalves et A. Goncalves. « Object-Relational Mapping ». Dans : *Beginning Java EE 6 Platform with GlassFish 3*. Apress, 2010, p. 61–121. isbn : 978-1-4302-2890-5. url : http://dx.doi.org/10.1007/978-1-4302-2890-5_3.

[GHJ+95] E. Gamma, R. Helm, R. Johnson et al. *Design patterns : elements of reusable object-oriented software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. isbn : 0-201-63361-2.

[GJ05] R. Groenmo et M. Jaeger. « Model-driven semantic Web service composition ». Dans : *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*. 2005. doi : 10.1109/APSEC.2005.81.

[GKR+08] H. Grönniger, H. Krahn, B. Rumpe et al. « MontiCore : a framework for the development of textual domain specific languages ». Dans : *Companion of the 30th international conference on Software engineering*. ICSE Companion '08. New York, NY, USA : ACM, 2008, p. 925–926. doi : http://doi.acm.org/10.1145/1370175.1370190. url : http://doi.acm.org/10.1145/1370175.1370190.

[GS03] G. Gössler et J. Sifakis. « Composition for Component-Based Modeling ». Dans : *Formal Methods for Components and Objects*. Éd. par F. S. de Boer, M. M. Bonsangue, S. Graf et al. T. 2852. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, p. 443–466. url : http://dx.doi.org/10.1007/978-3-540-39656-7_19.

[GW06] H. Giese et R. Wagner. « Incremental Model Synchronization with Triple Graph Grammars ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par O. Nierstrasz, J. Whittle, D. Harel et al. T. 4199. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, p. 543–557. URL : http://dx.doi.org/10.1007/11880240_38.

[GW09] H. Giese et R. Wagner. « From model transformation to incremental bidirectional model synchronization ». Dans : *Software and Systems Modeling* 8 (1 2009), p. 21–43. ISSN : 1619-1366. URL : http://dx.doi.org/10.1007/s10270-008-0089-9.

[Goo11] Google. *Google Scholar*. online. Juin 2011. URL : http://scholar.google.fr.

[HHJ+08] J. Henriksson, F. Heidenreich, J. Johannes et al. « Extending grammars and metamodels for reuse : the Reuseware approach ». Dans : *Software, IET* 2.3 (juin 2008), p. 165 –184.

[HK03] J. H. Hausmann et S. Kent. « Visualizing model mappings in UML ». Dans : *SoftVis '03 : Proceedings of the 2003 ACM symposium on Software visualization*. San Diego, California : ACM, 2003, p. 169–178. ISBN : 1-58113-642-0. DOI : http://doi.acm.org/10.1145/774833.774858.

[HKG+10] Z. Hemel, L. Kats, D. Groenewegen et al. « Code generation by model transformation : a case study in transformation modularity ». Dans : *Software and Systems Modeling* 9 (3 2010), p. 375–402. ISSN : 1619-1366. URL : http://dx.doi.org/10.1007/s10270-009-0136-1.

[HR04] D. Harel et B. Rumpe. « Meaningful modeling : what's the semantics of "semantics" ? » Dans : *Computer* 37.10 (oct. 2004), p. 64 –72. ISSN : 0018-9162. DOI : 10.1109/MC.2004.172.

[HRW11] J. Hutchinson, M. Rouncefield et J. Whittle. « Model-driven engineering practices in industry ». Dans : *Proceeding of the 33rd international conference on Software engineering*. ICSE '11. Waikiki Honolulu, HI, USA : ACM, 2011, p. 633–642. ISBN : 978-1-4503-0445-0. DOI : http://doi.acm.org/10.1145/1985793.1985882. URL : http://doi.acm.org/10.1145/1985793.1985882.

[IBM11] IBM. *IBM Technical Journal*. online. Juin 2011. URL : http://www.research.ibm.com/journal/.

[IEE05] IEEE Computer Society. « IEEE Standard for Software Verification and Validation ». Dans : *IEEE Std 1012-2004 (Revision of IEEE Std 1012-1998)* (août 2005), p. 0–110. DOI : 10.1109/IEEESTD.2005.96278.

[IEE11a] IEEE (Institute of Electrical and Electronics Engineers). *Computer Society Digital Library*. online. Juin 2011. URL : http://www.computer.org/portal/web/csdl/home.

[IEE11b] IEEE (Institute of Electrical and Electronics Engineers). *IEEE Xplore*. online. Juin 2011. URL : http://ieeexplore.ieee.org/Xplore/guesthome.jsp.

[IK04] I. Ivkovic et K. Kontogiannis. « Tracing evolution changes of software artifacts through model synchronization ». Dans : *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*. 2004, p. 252 –261. DOI : 10.1109/ICSM.2004.1357809.

[Ini11] Inist-CNRS. *Refdoc*. online. Juin 2011. URL : http://www.refdoc.fr/.

[JBF10] J.-M. Jézéquel, O. Barais et F. Fleurey. *Model Driven Language Engineering with Kermeta*. English. Éd. par Joao M. Fernandes, Ralf Lammel, Joao Saraiva et al. LNCS 6491, Springer, 2010. URL : http://hal.archives-ouvertes.fr/inria-00538461/PDF/Jezequel10b.pdf.

[JFB08] C. Jeanneret, R. France et B. Baudry. « A reference process for model composition ». Dans : *Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling*. AOM '08. New York, NY, USA : ACM, 2008, p. 1–6. DOI : http://doi.acm.org/10.1145/1404920.1404921. URL : http://doi.acm.org/10.1145/1404920.1404921.

[JKB+06] A. Jackson, J. Klein, B. Baudry et al. « Executable Aspect Oriented Models for Improved Model Testing ». Dans : *ECMDA workshop on Integration of Model Driven Development and Model Driven Testing*. Bilbao, Spain Espagne 2006. URL : http://hal.inria.fr/inria-00512544/en/.

[JWE+07] P. Jayaraman, J. Whittle, A. Elkhodary et al. « Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par G. Engels, B. Opdyke, D. Schmidt et al. T. 4735. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, p. 151–165. DOI : 10.1007/978-3-540-75209-7_11. URL : http://dx.doi.org/10.1007/978-3-540-75209-7_11.

[JZF+09] J. Johannes, S. Zschaler, M. Fernández et al. « Abstracting Complex Languages through Transformation and Composition ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par A. Schürr et B. Selic. T. 5795. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 546–550. DOI : 10.1007/978-3-642-04425-0_41. URL : http://dx.doi.org/10.1007/978-3-642-04425-0_41.

[Jac90] M. Jackson. « Some complexities in computerbased systems and their implications for system development ». Dans : *CompEuro '90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*. 1990, p. 344 –351. DOI : 10.1109/CMPEUR.1990.113645.

[Jea08] C. Jeanneret. « An Analysis of Model Composition Approaches ». Mém.de maîtr. Ecole Polytechnique Fédérale de Lausanne, 2008.

[Jez08] J.-M. Jezequel. « Model driven design and aspect weaving ». Dans : *Software and Systems Modeling* 7 (2 2008), p. 209–218. issn : 1619-1366. url : http://dx.doi.org/10.1007/s10270-008-0080-5.

[KAAK09] J. Kienzle, W. Al Abed et J. Klein. « Aspect-oriented multi-view modeling ». Dans : *Proceedings of the 8th ACM international conference on Aspect-oriented software development*. AOSD '09. New York, NY, USA : ACM, 2009, p. 87–98. doi : http://doi.acm.org/10.1145/1509239.1509252. url : http://doi.acm.org/10.1145/1509239.1509252.

[KEG11] KEG. *Arnetminer*. online. Juin 2011. url : http://arnetminer.net/index.jsp.

[KGM10] J. Kienzle, N. Guelfi et S. Mustafiz. « Crisis Management Systems : A Case Study for Aspect-Oriented Modeling ». Dans : *Transactions on Aspect-Oriented Software Development VII*. Éd. par S. Katz, M. Mezini et J. Kienzle. T. 6210. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, p. 1–22. isbn : 978-3-642-16085-1. url : http://dx.doi.org/10.1007/978-3-642-16086-8_1.

[KHJ06] J. Klein, L. Hélouet et J.-M. Jézéquel. « Semantic-based Weaving of Scenarios ». Dans : *proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD'06)*. Bonn, Germany : ACM, 2006.

[KJP05] J. Klein, J.-M. Jézéquel et N. Plouzeau. « Weaving Behavioural Models ». Dans : *In First Workshop on Models and Aspects, Handling Crosscutting Concerns in MDSD at ECOOP 05*. 2005.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar et al. « Aspect-oriented programming ». Dans : *ECOOP'97 – Object-Oriented Programming*. Éd. par M. Aksit et S. Matsuoka. T. 1241. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1997, p. 220–242. isbn : 978-3-540-63089-0. url : http://dx.doi.org/10.1007/BFb0053381.

[KM10] P. Kelsen et Q. Ma. « A Modular Model Composition Technique ». Dans : *Fundamental Approaches to Software Engineering*. Éd. par D. Rosenblum et G. Taentzer. T. 6013. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, p. 173–187. url : http://dx.doi.org/10.1007/978-3-642-12029-9_13.

[KPP06] D. Kolovos, R. Paige et F. Polack. « Merging Models with the Epsilon Merging Language (EML) ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par O. Nierstrasz, J. Whittle, D. Harel et al. T. 4199. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, p. 215–229. url : http://dx.doi.org/10.1007/11880240_16.

[KUL+10] F. Krause, J. Uhlendorf, T. Lubitz et al. « Annotation and merging of SBML models with semanticSBML ». Dans : *Bioinformatics* 26.3 (2010), p. 421–422. doi : 10.1093/bioinformatics/btp642. eprint : http://bioinformatics.oxfordjournals.org/content/26/3/421.full.pdf+html. url : http://bioinformatics.oxfordjournals.org/content/26/3/421.abstract.

[Kit04] B. Kitchenham. « Procedures for Performing Systematic Reviews ». Dans : *Joint Technical Report NICTA Technical Report 0400011T1* 33 (2004). url : http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.3308&rep=rep1&type=pdf.

[LMV+07] P. Lahire, B. Morin, G. Vanwormhoudt et al. « Introducing Variability into Aspect-Oriented Modeling Approaches ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par G. Engels, B. Opdyke, D. Schmidt et al. T. 4735. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, p. 498–513. url : http://dx.doi.org/10.1007/978-3-540-75209-7_34.

[LNK+01] A. Ledeczi, G. Nordstrom, G. Karsai et al. « On metamodel composition ». Dans : *Control Applications, 2001. (CCA '01). Proceedings of the 2001 IEEE International Conference on*. 2001, p. 756 –760. doi : 10.1109/CCA.2001.973959.

[LP03] V.-C. Liang et C. Paredis. « A port ontology for automated model composition ». Dans : *Simulation Conference, 2003. Proceedings of the 2003 Winter*. T. 1. 2003, p. 613 –622. doi : 10.1109/WSC.2003.1261476.

[Let07] K. Letkeman. *Comparing and merging UML models in IBM Rational Software Architect : Ad-hoc modeling - Fusing two models with diagrams*. English. IBM. 2007. url : http://www.ibm.com/developerworks/rational/library/07/0410_letkeman/.

[Lor98] D. Lorenz. « Visitor Beans : An Aspect–Oriented Pattern ». Dans : *Object–Oriented Technology : ECOOP'98 Workshop Reader*. Éd. par S. Demeyer et J. Bosch. T. 1543. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1998, p. 579–579. isbn : 978-3-540-65460-5. url : http://dx.doi.org/10.1007/3-540-49255-0_130.

[MBFF10] S. Mosser, M. Blay-Fornarino et R. France. « Workflow Design Using Fragment Composition ». Dans : *Transactions on Aspect-Oriented Software Development VII*. Éd. par S. Katz, M. Mezini et J. Kienzle. T. 6210. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, p. 200–233. doi : 10.1007/978-3-642-16086-8_6. url : http://dx.doi.org/10.1007/978-3-642-16086-8_6.

[MBJ+07] B. Morin, O. Barais, J.-M. Jézéquel et al. « Towards a Generic Aspect-Oriented Modeling Framework ». Dans : *Models and Aspects workshop, at ECOOP 2007*. Berlin, Germany Allemagne 2007. url : http://hal.archives-ouvertes.fr/inria-00505222/PDF/morin07a.pdf.

[MBJ08] B. Morin, O. Barais et J.-M. Jézéquel. « Weaving Aspect Configurations for Managing System Variability ». Dans : *VaMoS'08 : 2nd Int. Workshop on Variability Modelling of Software-Intensive Systems*. 2008.

[MBN+09] B. Morin, O. Barais, G. Nain et al. « Taming Dynamically Adaptive Systems using models and aspects ». Dans : *Proceedings of the 31st International Conference on Software Engineering*. ICSE '09. Washington, DC, USA : IEEE Computer Society, 2009, p. 122–132. ISBN : 978-1-4244-3453-4. DOI : http://dx.doi.org/10.1109/ICSE.2009.5070514. URL : http://dx.doi.org/10.1109/ICSE.2009.5070514.

[MFJ05] P.-A. Muller, F. Fleurey et J.-M. Jézéquel. « Weaving Executability into Object-Oriented Meta-Languages ». Dans : *Proceedings of MODELS/UML'2005*. Éd. par S. K. L. Briand. T. 3713. LNCS. Montego Bay, Jamaica : Springer, 2005, p. 264–278. URL : http://www.irisa.fr/triskell/publis/2005/Muller05a.pdf.

[MKB+08] B. Morin, J. Klein, O. Barais et al. « A Generic Weaver for Supporting Product Lines ». Dans : *EA '08 : Proceedings of the 13th international workshop on Early Aspects*. Leipzig, Germany : ACM, 2008, p. 11–18. ISBN : 978-1-60558-032-6. DOI : http://doi.acm.org/10.1145/1370828.1370832.

[MMP+10] I. Malavolta, H. Muccini, P. Pelliccione et al. « Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies ». Dans : *Software Engineering, IEEE Transactions on* 36.1 (jan. 2010), p. 119 –140. DOI : 10.1109/TSE.2009.51.

[MMW02] K. Mens, T. Mens et M. Wermelinger. « Supporting unanticipated software evolution through intentional software views ». Dans : *ECOOP 2002 Workshop on Unanticipated Software Evolution*. 2002.

[MPL+09] B. Morin, G. Perrouin, P. Lahire et al. « Weaving Variability into Domain Metamodels ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par A. Schürr et B. Selic. T. 5795. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 690–705. URL : http://dx.doi.org/10.1007/978-3-642-04425-0_56.

[Mos10] S. Mosser. « Behavioral Compositions in Service-Oriented Architecture ». Thèse de doct. Université de Nice-Sophia Antipolis, 2010.

[NB04] K. Nahrstedt et W.-T. Balke. « A taxonomy for multimedia service composition ». Dans : *Proceedings of the 12th annual ACM international conference on Multimedia*. New York, NY, USA : ACM, 2004, p. 88–95. DOI : http://doi.acm.org/10.1145/1027527.1027544. URL : http://doi.acm.org/10.1145/1027527.1027544.

[NM00] N. F. Noy et M. A. Musen. « PROMPT : Algorithm and Tool for Automated Ontology Merging and Alignment ». Dans : *AAAI/IAAI*. AAAI Press / The MIT Press, 2000, p. 450–455. ISBN : 0-262-51112-6.

[NSC+07] S. Nejati, M. Sabetzadeh, M. Chechik et al. « Matching and Merging of Statecharts Specifications ». Dans : *ICSE '07 : Proceedings of the 29th international conference on Software Engineering*. ICSE'07. Washington, DC, USA : IEEE Computer Society, 2007, p. 54–64. isbn : 0-7695-2828-7. doi : http://dx.doi.org/10.1109/ICSE.2007.50.

[OMG01] OMG. *Model Driven Architecture (MDA)*. online. Juil. 2001. url : http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01.

[OMG07] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*. online. Juil. 2007. url : http://www.omg.org/cgi-bin/doc?ptc/2007-07-07.

[OMG10a] OMG. *MOF Specification v.2.4 - Beta 2*. online. Déc. 2010. url : http://www.omg.org/spec/MOF/2.4/Beta2.

[OMG10b] OMG. *Uml Superstructure Specification v2.4 - Beta 2*. online. Nov. 2010. url : http://www.omg.org/spec/UML/2.4/.

[OMK09] J. Oldevik, M. Menarini et I. Krüger. « Model Composition Contracts ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par A. Schürr et B. Selic. T. 5795. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 531–545. url : http://dx.doi.org/10.1007/978-3-642-04425-0_40.

[OO07] K. Oliveira et T. de Oliveira. « A Guidance for Model Composition ». Dans : *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*. 2007, p. 27. doi : 10.1109/ICSEA.2007.5.

[Opd92] W. F. Opdyke. « Refactoring Object-Oriented Frameworks ». Thèse de doct. Champaign, IL, USA : University of Illinois at Urbana-Champaign, 1992.

[Ove11] Oversity Limited. *CiteULike*. online. Juin 2011. url : http://www.citeulike.org/home.

[Oxf11] Oxford University Press. *Oxford Journals*. online. Juin 2011. url : http://bioinformatics.oxfordjournals.org/.

[PB09] R. Pottinger et P. Bernstein. « Associativity and Commutativity in Generic Merge ». Dans : *Conceptual Modeling : Foundations and Applications*. Éd. par A. Borgida, V. Chaudhri, P. Giorgini et al. T. 5600. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 254–272. doi : 10.1007/978-3-642-02463-4_14. url : http://dx.doi.org/10.1007/978-3-642-02463-4_14.

[PBB+09] G. Perrouin, E. Brottier, B. Baudry et al. « Composing Models for Detecting Inconsistencies : A Requirements Engineering Perspective ». Dans : *Requirements Engineering : Foundation for Software Quality*. Éd. par M. Glinz et P. Heymans. T. 5512. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 89–103. doi : 10.1007/978-3-

642-02050-6_8. URL : http://dx.doi.org/10.1007/978-3-642-02050-6_8.

[PBC+11] C. Parra, X. Blanc, A. Cleve et al. « Unifying Design and Runtime Adaptations Using Aspect Models ». Anglais. Dans : *Science of Computer Programming* (19 jan. 2011). DOI : 10.1016/j.scico.2010.12.005. URL : http://hal.inria.fr/inria-00564592/en/.

[PDCS+01] C. Paredis, A. Diaz-Calderon, R. Sinha et al. « Composable Models for Simulation-Based Design ». Dans : *Engineering with Computers* 17 (2001), p. 112–128. DOI : 10.1007/PL00007197. URL : http://dx.doi.org/10.1007/PL00007197.

[PGP+07] C. Pons, R. Giandini, G. Perez et al. « An Algebraic Approach for Composing Model Transformations in QVT ». Dans : *4th International Workshop on Software Language Engineering at the 10th International Conference MoDELS 2007*. ATEM. Citeseer, 2007.

[PR04] J. Park et S. Ram. « Information systems interoperability : What lies beneath ? » Dans : *ACM Trans. Inf. Syst.* 22 (4 2004), p. 595–632. ISSN : 1046-8188. DOI : http://doi.acm.org/10.1145/1028099.1028103. URL : http://doi.acm.org/10.1145/1028099.1028103.

[PRB+09] L. Pedro, M. Risoldi, D. Buchs et al. « Composing Visual Syntax for Domain Specific Languages ». Dans : *Human-Computer Interaction. Novel Interaction Methods and Techniques*. Éd. par J. Jacko. T. 5611. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 889–898. DOI : 10.1007/978-3-642-02577-8_97. URL : http://dx.doi.org/10.1007/978-3-642-02577-8_97.

[PVSG+08] J. von Pilgrim, B. Vanhooff, I. Schulz-Gerlach et al. « Constructing and Visualizing Transformation Chains ». Dans : *Model Driven Architecture - Foundations and Applications*. Éd. par I. Schieferdecker et A. Hartman. T. 5095. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 17–32. URL : http://dx.doi.org/10.1007/978-3-540-69100-6_2.

[Par72] D. L. Parnas. « On the criteria to be used in decomposing systems into modules ». Dans : *Commun. ACM* 15 (12 1972), p. 1053–1058. ISSN : 0001-0782. DOI : http://doi.acm.org/10.1145/361598.361623. URL : http://doi.acm.org/10.1145/361598.361623.

[Pen11] Penn State College of Information and Technology. *CiteSeerX Digital Library and Search Engine*. online. Juin 2011. URL : http://citeseerx.ist.psu.edu/.

[RCE08] J. Rubin, M. Chechik et S. M. Easterbrook. « Declarative approach for model composition ». Dans : *MiSE '08 : Proceedings of the 2008 international workshop on Models in software engineering*. Leipzig, Germany :

ACM, 2008, p. 7–14. ISBN : 978-1-60558-025-8. DOI : http://doi.acm.org/10.1145/1370731.1370734.

[RGF+06] Y. REDDY, S. GHOSH, R. FRANCE et al. « Directives for Composing Aspect-Oriented Design Class Models ». Dans : *Transactions on Aspect-Oriented Software Development I*. Éd. par A. RASHID et M. AKSIT. T. 3880. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, p. 75–105. URL : http://dx.doi.org/10.1007/11687061_3.

[Res11] RESEARCHR. *Researchr*. online. Juin 2011. URL : http://researchr.org/.

[SBP+08] D. STEINBERG, F. BUDINSKY, M. PATERNOSTRO et al. *EMF : Eclipse Modeling Framework (2nd Edition)*. 2008.

[SE06] M. SABETZADEH et S. EASTERBROOK. « View merging in the presence of incompleteness and inconsistency ». Dans : *Requir. Eng.* 11.3 (2006), p. 174–193. ISSN : 0947-3602. DOI : http://dx.doi.org/10.1007/s00766-006-0032-y.

[SFS+08] P. SÁNCHEZ, L. FUENTES, D. STEIN et al. « Aspect-Oriented Model Weaving Beyond Model Composition and Model Transformation ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par K. CZARNECKI, I. OBER, J.-M. BRUEL et al. T. 5301. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 766–781. URL : http://dx.doi.org/10.1007/978-3-540-87875-9_53.

[SJ07] J. STEEL et J.-M. JÉZÉQUEL. « On model typing ». Dans : *Software and Systems Modeling* 6 (4 2007), p. 401–413. ISSN : 1619-1366. URL : http://dx.doi.org/10.1007/s10270-006-0036-6.

[SY10] M. SHONLE et T. T. YUEN. « Compose & conquer : modularity for end-users ». Dans : *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*. ICSE '10. New York, NY, USA : ACM, 2010, p. 191–194. DOI : http://doi.acm.org/10.1145/1810295.1810327. URL : http://doi.acm.org/10.1145/1810295.1810327.

[Spr11] SPRINGER-VERLAG GMBH. *SpringerLink*. online. Juin 2011. URL : http://www.springerlink.com/.

[Sti81] M. E. STICKEL. « A Unification Algorithm for Associative-Commutative Functions ». Dans : *J. ACM* 28 (3 1981), p. 423–434. ISSN : 0004-5411. DOI : http://doi.acm.org/10.1145/322261.322262. URL : http://doi.acm.org/10.1145/322261.322262.

[TJF+09] M. TISI, F. JOUAULT, P. FRATERNALI et al. « On the Use of Higher-Order Model Transformations ». Dans : *Model Driven Architecture - Foundations and Applications*. Éd. par R. PAIGE, A. HARTMAN et A. RENSINK. T. 5562. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, p. 18–33. URL : http://dx.doi.org/10.1007/978-3-642-02674-4_3.

[TT08] W. Tansey et E. Tilevich. « Annotation refactoring : inferring upgrade transformations for legacy applications ». Dans : *OOPSLA '08 : Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. Nashville, TN, USA : ACM, 2008, p. 295–312. ISBN : 978-1-60558-215-3. DOI : http://doi.acm.org/10.1145/1449764.1449788.

[Tae04] G. Taentzer. « AGG : A Graph Transformation Environment for Modeling and Validation of Software ». Dans : *Applications of Graph Transformations with Industrial Relevance*. 2004, p. 446–453. DOI : 10.1007/b98116. URL : http://dx.doi.org/10.1007/b98116.

[VAVB+07] B. Vanhooff, D. Ayed, S. Van Baelen et al. « UniTI : A Unified Transformation Infrastructure ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par G. Engels, B. Opdyke, D. Schmidt et al. T. 4735. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, p. 31–45. URL : http://dx.doi.org/10.1007/978-3-540-75209-7_3.

[VV95] M Vargas-Vera. « Using Prolog Techniques to Guide Program Composition ». Thèse de doct. University of Edinburgh, 1995.

[WJ08] J. Whittle et P. Jayaraman. « MATA : A Tool for Aspect-Oriented Modeling Based on Graph Transformation ». Dans : *Models in Software Engineering*. Éd. par H. Giese. T. 5002. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 16–27. URL : http://dx.doi.org/10.1007/978-3-540-69073-3_3.

[WS08] I. Weisemöller et A. Schürr. « Formal Definition of MOF 2.0 Metamodel Components and Composition ». Dans : *Model Driven Engineering Languages and Systems*. Éd. par K. Czarnecki, I. Ober, J.-M. Bruel et al. T. 5301. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, p. 386–400. URL : http://dx.doi.org/10.1007/978-3-540-87875-9_28.

[Wac07] G. Wachsmuth. « Metamodel Adaptation and Model Co-adaptation ». Dans : *ECOOP 2007 - Object-Oriented Programming*. Éd. par E. Ernst. T. 4609. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, p. 600–624. URL : http://dx.doi.org/10.1007/978-3-540-73589-2_28.

[Wag08] D. Wagelaar. « Composition Techniques for Rule-Based Model Transformation Languages ». Dans : *Proceedings of the 1st international conference on Theory and Practice of Model Transformations (ICMT'08)*. Berlin, Heidelberg : Springer-Verlag, 2008, p. 152–167. DOI : http://dx.doi.org/10.1007/978-3-540-69927-9_11. URL : http://dx.doi.org/10.1007/978-3-540-69927-9_11.

[XLH+07] Y. Xiong, D. Liu, Z. Hu et al. « Towards automatic model synchronization from model transformations ». Dans : *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ASE '07. Atlanta, Georgia, USA : ACM, 2007, p. 164–173. isbn : 978-1-59593-882-4. doi : http://doi.acm.org/10.1145/1321631.1321657. url : http://doi.acm.org/10.1145/1321631.1321657.

[ZC07] J. Zhang et B. Cheng. « Towards Re-engineering Legacy Systems for Assured Dynamic Adaptation ». Dans : *Modeling in Software Engineering, 2007. MISE '07 : ICSE Workshop 2007. International Workshop on*. 2007, p. 10. doi : 10.1109/MISE.2007.14.

[ZDD06] A. Zito, Z. Diskin et J. Dingel. « Package Merge in UML 2 : Practice vs. Theory ? » Dans : *Model Driven Engineering Languages and Systems*. Éd. par O. Nierstrasz, J. Whittle, D. Harel et al. T. 4199. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, p. 185–199. url : http://dx.doi.org/10.1007/11880240_14.

[ZLL09] D. Zhang, S. Li et X. Liu. « An Approach for Model Composition and Verification ». Dans : *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*. 2009, p. 1102 –1107. doi : 10.1109/NCM.2009.271.

List of figures

List of tables

VU :

Le Directeur de Thèse
(Nom et Prénom)

VU :

Le Responsable de L'Ecole Doctorale

VU pour autorisation de soutenance

Rennes, le

Le Président de l'Université de Rennes 1

Guy CATHELINEAU

VU après soutenance pour autorisation de publication :

Le Président de Jury,
(Nom et Prénom)

Résumé

L'Ingénierie Dirigée par les Modèles (IDM) est basée sur le principe d'abstraction et de séparation des préoccupations pour gérer la complexité du développement de logiciels. Les ingénieurs s'appuient sur des modèles dédiées à la résolution d'un problème particulier. Dans le cadre de l'IDM, la composition de modèles est un domaine de recherche très actif qui vise à automatiser les tâches de recomposition des modèles. La quasi-inexistence de consensus pour comparer les techniques existantes entraîne une explosion de l'effort nécessaire pour produire de nouveaux outils spécifiques à partir de techniques existantes.

La contribution principale de cette thèse est de proposer une définition originale de la composition de modèles comme étant une paire correspondance-interprétation. A partir de cette définition, nous proposons un cadre théorique qui (1) unifie les représentations des techniques de composition existantes et qui (2) automatise le développement d'outils de composition de modèles. La contribution principale s'appuie sur deux sous-contributions supplémentaires :

- Nous proposons des catégories pour classer les techniques de correspondance et les interprétations existantes.
- Nous proposons un langage de modélisation spécifique pour la définition de correspondances génériques entre modèles et la définition d'interprétations.

Un prototype logiciel a été développé et utilisé dans le cadre du projet MOPCOM-I du pôle de compétitivité Images & Réseaux de la région Bretagne. La validation de la contribution a été démontrée sur un cas d'étude proposé par Technicolor pour l'intégration de librairies existantes dédiées à la gestion d'équipements numériques de diffusion vidéo.

Abstract

Model-Driven Engineering (MDE) is a software development methodology that relies on the Separation of Concerns (SoC) and Abstraction principles to deal with complexity. Thinking in terms of higher levels of abstraction and building dedicated models to address specific concerns allow decomposing a problem into more manageable subproblems. Within the framework of MDE, model composition is an active field of research that focuses on automating the composition of model-based artifacts in a multi-modeling environment. However the lack of a common formalism for comparing existing approaches hinders their adaptation and reuse for building new model composition techniques. The main contribution of this thesis is to propose a novel definition of model composition as a pair of a mapping and an interpretation. This definition paves the way to a theoretical framework that (1) unifies existing representations of model composition techniques and (2) automates the process of building model composition tools. The main contribution is supported by two subsidiaries propositions : - We propose categories to classify existing mapping techniques and existing model composition interpretations. - We define a language that supports the definition of generic mappings among models and the definition of interpretations. We validate the contribution through two experiments : (1) a systematic literature review validates the proposed categories for mappings and interpretations ; (2) a prototype that supports the model composition approach has been tested on an industrial case study from Technicolor about the composition of legacy APIs for the management of heterogeneous video and broadcasting equipments.