جامعة سيدي محمد بن عبد الله بفاس
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ ⴷ ⴼⴰⵙ
UNIVERSITE SIDI MOHAMED BEN ABDALLAH DE FES
كلية العلوم ظهر المهراز فاس
ⵜⴰⵖⵣⵡⵜ ⵏ ⵜⵣⵔⴼⵜ ⵎⵓⵃⵔⴰⵣ ⴼⴰⵙ
FACULTE DES SCIENCES DHAR EL MEHRAZ FES

Formation Doctorale : STIC (Sciences et Technologies de l'Information et de Communication)

Discipline : Génie Logiciel

Spécialité : Informatique

Laboratoire : LIMS

THESE DE DOCTORAT

Présentée par
Nisrine EL MARZOUKI

Model composition in multi-modeling approaches based on Model Driven Engineering

Composition des modèles dans les approches de multi-modélisation basée sur l'Ingénierie Dirigée par les Modèles

Soutenue le devant le jury composé de :

Pr. Ahmed ZINEDINE	**FSDM – Université Sidi Mohamed Ben Abdellah**	**Président**
Pr. Adil KENZI	**ENSA – Université Sidi Mohamed Ben Abdellah**	**Rapporteur**
Pr. Adil ANWAR	**EMI – Université Mohamed V**	**Rapporteur**
Pr. Mustapha HAIN	**ENSAM – Université Hassan II**	**Rapporteur**
Pr. Mohamed EL FAR	**FSDM – Université Sidi Mohamed Ben Abdellah**	**Examinateur**
Pr. Aicha MAJDA	**FST – Université Sidi Mohamed Ben Abdellah**	**Examinateur**
Pr. Oksana NIKIFOROVA	**Université de Riga, Lettonie**	**Examinateur**
Pr. Mohammed EL MOHAJIR	**FSDM – Université Sidi Mohamed Ben Abdellah**	**Directeur de thèse**
Pr. Younes LAKHRISSI	**ENSA – Université Sidi Mohamed Ben Abdellah**	**Encadrant**

Acknowledgements

The research work presented in this thesis has been supported by the research grant of the Erasmus+ EU program for a mobility period between Sidi Mohammed Ben Abdellah University and Riga Technical University and also by an Excellence Scholarship of National Center for Technological Research (CNRT : Centre National de Recherche Technologique).

To my family, for their dedicated partnership and love in every moment of my life.

Remerciements

Avant tout, je remercie Dieu le tout puissant, de m'avoir donné le courage et de m'avoir guidé, pour pouvoir bien mener ce modeste travail.

Je tiens à présenter mes gratitudes et mes vifs remerciements, à mon encadrant Monsieur **Younes Lakhrissi**, qui m'a lancé dans le monde de la recherche, qui m'a encouragé et a cru dans mes capacités et compétences. Je vous remercie pour tout le soutien, l'énergie, la vision et l'encouragement tout au long de ce travail. Vos solides connaissances, m'ont permis de travailler sur de nouvelles voies de recherche avec certitude et rigueur. Je pense sincèrement m'être enrichi à votre contact.

Je tiens à remercier mon directeur de thèse, **Mohammed El Mohajir**, pour ses orientations et ses précieux conseils tout au long de cette thèse.

Je présente aussi nos remerciements aux rapporteurs d'avoir accepté d'évaluer mon travail. Je les remercie du temps qu'ils ont consacré à la lecture de ce manuscrit, pour l'intérêt ainsi que ses remarques qu'ils ont accordé à mon travail.

J'adresse également mes remerciements à l'équipe de recherche à l'université technique de Riga en Lettonie et particulièrement mon encadrante **Oksana Nikiforova,** pour l'accompagnement, les directives et l'aide qu'elle m'a fourni dans la réalisation de ce travail.

Je remercie tous les professeurs qui ont contribué à mes formations complémentaires durant cette thèse. Je vous remercie de la clarté, de la richesse de votre enseignement, et de votre gentillesse qui nous ont grand ouvert les portes de la connaissance.

J'exprime aussi toute ma gratitude aux membres de jury qui m'ont fait l'honneur de participer à ma soutenance.

Enfin je remercie les personnes qui ont contribué de près ou de loin à la réalisation de ce travail, qu'ils trouvent ici l'expression de mes gratitudes et ma parfaite considération.

Merci à vous.

Résumé

Les travaux de recherche présentés dans cette thèse se focalisent sur la composition des modèles dans le cadre de l'Ingénierie Dirigée par les Modèles (MDA : Model Driven Architecture) et s'inscrit dans la continuité des travaux de l'approche « Two hemisphere Model Driven Architecture » développée au sein de notre équipe de recherche. En effet, on constate, de plus en plus, qu'un système logiciel réel est bien trop complexe pour pouvoir être décrit par un seul modèle. Ce qui rend tout raisonnement global sur le système difficile. De nombreux modèles devront être créés pour le spécifier, soit à divers niveaux d'abstraction, soit selon divers points de vue, soit encore selon des domaines fonctionnels différents et complémentaires.

Les travaux effectués dans le cadre de notre thèse se focalisent sur la définition de l'opération de composition de modèles issus d'une phase de conception décentralisée. Cette opération vise l'élaboration d'une démarche pour la fusion des diagrammes de classes afin de combler les lacunes de l'opération de la composition des modèles, la formalisation du processus de composition sur le niveau structurel et comportemental, ainsi que la proposition d'une approche pour la composition des modèles tout en définissant un référentiel de gestion des conflits qui permettra d'assurer la conformité entre les modèles réalisés et garantir la cohérence de l'assemblage tout au long du processus de composition.

Pour réaliser ces objectifs, nous avons travaillé sur un prototype conceptuel qui commence avec une phase de comparaison afin de déterminer si deux instances correspondent en utilisant un ensemble de critères de comparaison et de propriétés syntaxiques définies au niveau métamodèle. Cette comparaison des relations nous permet de capturer les différentes correspondances entre les éléments des modèles cibles. Ces correspondances seront fournies en entrée de l'étape de fusion. Le modèle de correspondances est conforme à un métamodèle qui définit la structure des modèles qui lui sont conformes et fournit les constructions nécessaires à la gestion des relations de correspondance et des conflits générés.

La composition de plusieurs diagrammes de classes provenant de différentes sources justifie notre motivation pour la conception de ce prototype en se basant sur l'ingénierie dirigée par les modèles et l'approche «Two hemisphere Model Driven Architecture», qui favorisent la modélisation et la conception des systèmes de manière indépendante de la plate-forme et décrivent la transformation du processus métier en diagrammes UML.

Nous montrons que pour atteindre cet objectif, il faut en premier lieu, composer les métamodèles, puis les modèles. Nous montrons dans ce travail comment ces compositions peuvent être réalisées sans modifier les éléments composés, en utilisant des webservices.

Cette approche est validée d'abord par la réalisation d'un plugin basé sur l'approche des webservices. Ce plugin ou service est capable de s'exécuter dans plusieurs environnements et d'interagir avec plusieurs applications.

Mots clés :

Ingénierie Dirigée Par Les Modèles, Model Driven Architecture, Processus De Composition, Règles De Transformation De Modèles, Two Hemisphere Model Driven Architecture, Prototype, Fusion, Référentiel De Gestion Des Conflits, Plugin, Webservices.

Abstract

The research work presented in this thesis focuses on Model Composition in the framework of Model Driven Architecture (MDA) and is a continuity of «The-Two hemisphere Model Driven Architecture" methodology developed within our research team.

Indeed, actually we are finding that a real software system is too complex to be described by a single model, which makes any global reasoning about the system difficult.

In this context many models should be created to specify a complex system, either at various levels of abstraction, various points of view, or according to different and complementary functional domains.

The main goal of this thesis is to formalize the operation of model composition in order to design complex systems according to the vision of business partners.

The work carried out during research focuses on the definition of a specific process of model composition to design models resulting from a decentralized modeling phase and aims to develop an approach for the fusion of class diagrams, while defining a repository to manage conflicts, ensure conformity between models made and guarantee the consistency of the assembly throughout the composition process.

To achieve these goals, we set up a conceptual composition prototype that starts with a comparison phase in order to determine the feasibility of matching two instances using a set of comparison criteria and syntax properties.

This study allows us to capture the different correspondences between the elements of the target models, which will be provided as an input of the merge step and should conform to a metamodel that defines the structure of the models and provides the main constructs to manage the correspondence relationships and generated conflicts.

The Composition of several class diagrams coming from different source justify our motivation for this prototype based on Model Driven Architecture and the two-hemisphere model driven approach, which propose the use of business process modelling and concept modelling to represent the systems in platform independent manner and describes how to apply transformations from business process model into UML models.

Applied to our context, this means being able to compose models without modifying them and without modifying the language or the execution environment.

We show that to achieve this goal, we must first compose the metamodels, then the models. We show in this work how these compositions can be performed without modifying the compound elements, using webservices, which is an architectural approach for application development.

This approach is first validated by the creation of a plugin based on the webservices approach. This plugin or service is able to run in multiple environments and interact with multiple applications.

Key Words:

Model Driven Architecture, Model Composition, Composition Process, Model Transformation Rules, Two Hemisphere Model Driven Architecture, Prototype, Fusion, Conflicts Management Repository, Plugin, Webservices.

List of Figures

List of Tables

Table of contents

CHAPTER I. GENERAL INTRODUCTION

I.1. Thesis French Summary

This thesis work was realized within the framework of the Information and Communication Sciences and Technologies network in software engineering. It was carried out under the supervision of three structures: LIMS (Laboratoire Informatique, Modélisation et Systèmes), SIGER (Laboratoire Systèmes Intelligents, Géoressources & Énergies Renouvelables) at Sidi Mohammed Ben Abdellah University Fes, and Riga Technigal University Latvia, in the context of an Erasmus exchange.

As I wrote my thesis entirely in English, I chose to present this section in French, in order to support the international dimension of this thesis.

I.1.1. Contexte

Ce travail de recherche se situe dans le cadre de l'Ingénierie Dirigée par les Modèles (MDA : Model Driven Architecture). Plusieurs recherches autour du MDA [MDA09] ont abouti à la définition de langages et outils permettant des opérations sur les modèles telles que la transformation ou encore la vérification. D'autres opérations, comme la composition ou la fusion demeurent par contre insuffisamment étudiées.

Notre travail se focalise sur la définition de l'opération de composition de modèles de conception issus d'une phase de conception décentralisée et vise l'élaboration d'une démarche pour la fusion des modèles.

I.1.2. Motivations

Il n'est plus besoin de justifier que la complexité et la taille des logiciels s'accroît, alors que les contraintes de qualité et de délais se durcissent.

Devant le défi permanent de la complexité, les principes de solution ne sont pas légion, ils sont connus "de toute éternité" : diviser pour régner, abstraire pour comprendre. Le défi de la qualité et des délais passe, lui, par la réutilisation. Par contre, la déclinaison de ces principes est fortement tributaire des technologies utilisées. En terme de Génie logiciel, "diviser" se décline dans les diverses manières de décomposer une application logicielle en problèmes "indépendants" résolus dans des parties (programmes, modules, services etc.) indépendantes ; et par la manière de recomposer ces parties pour obtenir le logiciel voulu.

De même dans le contexte de l'ingénierie dirigée par les modèles afin de réduire la complexité et pour améliorer la réutilisation, le produit à construire est aussi divisé en parties construites indépendamment et ensuite assemblées.

Par ailleurs, le MDA (Model Driven Architecture) est aujourd'hui en passe de devenir le nouveau paradigme en matière de développement d'application. L'objectif derrière l'utilisation d'un tel paradigme est d'augmenter la productivité et réduire le temps du développement des systèmes complexes au moyen de modèles qui sont beaucoup moins liés à la technologie et qui sont beaucoup plus proches du domaine métier.

Cette abstraction des problèmes complexes, rend les systèmes plus faciles à spécifier et à maintenir. L'une des variantes les plus connues de l'IDM est le standard MDA (Model Driven Architecture) qui, à travers l'utilisation de trois niveaux d'abstraction, a pour but de séparer la logique métier de l'application de la technologie qui sera utilisée pour la réaliser.

L'objectif est de permettre de supprimer le lien direct entre les applications et le codage qui leur est associé facilitant leur interopérabilité et les rendant ainsi moins sensibles aux évolutions technologiques.

Le concept du MDA impose que la composition se fasse (en grande partie) en terme de parties existantes (programmer en réutilisant) ; et inversement que les parties existantes soient suffisamment flexibles pour satisfaire aux besoins d'un grand nombre de décomposition (programmer pour réutiliser).

Cette approche récente propose "simplement" que les parties à construire et à assembler soient des modèles et non pas des programmes.

C'est ainsi que le problème de la composition de modèles est devenu un thème important de du MDA et l'utilisation de multiples paradigmes de modélisation pour développer un système complexe est à la fois inévitable et essentielle.

I.1.3. Problématique

Un système logiciel réel est bien trop complexe pour pouvoir être décrit par un seul modèle ce qui rend tout raisonnement global sur le système difficile. De nombreux modèles devront être créés pour le spécifier, soit à divers niveaux d'abstraction, soit selon divers points de vue, soit encore selon des domaines fonctionnels différents et complémentaires. [Voir Figure 1]

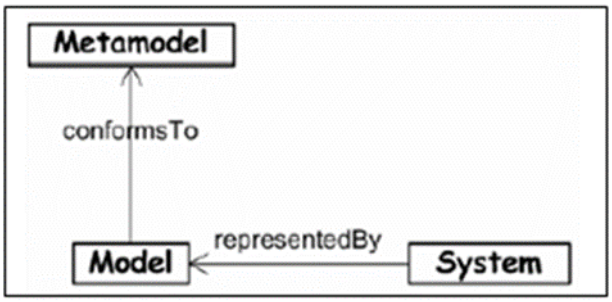


Figure 1 Exigences de la composition des modèles [MDA09]

L'objectif de la composition des modèles est de faciliter l'utilisation conjointe de modèles hétérogènes pendant le cycle de développement. En effet, la composition est toujours inhérente à la phase de décomposition précédente.

Dans la littérature la décomposition peut se faire de différentes manières; citons notamment la décomposition par point de vue, par aspect, par sujet et par rôle. La décomposition par point de vue vise à analyser et concevoir par objets de systèmes complexes avec une démarche centrée acteur. A

chaque acteur du système est associé un point de vue. A chaque point de vue s'appliquant sur le système est associé un ensemble de vues.

Le résultat d'une telle modélisation est un modèle unique, partageable, accessible suivant plusieurs points de vue. La décomposition par aspects est fondée sur la séparation entre les préoccupations fonctionnelles et les préoccupations dites « transversales » lors du développement logiciel. L'idée de la modélisation par aspects découle de la programmation par aspects et propose de considérer les aspects dans les modèles.

La décomposition par sujets est une autre technique de séparation. Cette approche est fondée sur une séparation multidimensionnelle des préoccupations, permettant de couvrir différents types de préoccupations (métier, technologiques, règles de gestion, etc.). Elle permet d'identifier un ensemble de spécifications et de comportements reflétant la perception du monde réel correspondant à une vision générique d'un acteur. Dans la décomposition par rôle, un objet interagit avec le monde de manière subjective selon la perspective du contexte de l'utilisateur.

L'apparition de la notion de rôle vient du fait que les propriétés extrinsèques d'un objet peuvent changer dans le temps. En effet, un objet peut être sujet à des classifications multiples durant son cycle de vie, et à chaque fois il joue un rôle adapté à une situation particulière.

Dans ce contexte, plusieurs approches existent ; elles se différencient notamment au niveau du type des opérateurs de composition proposés. Les adaptations sur lesquelles s'appuie la composition portent sur les constituants des modèles tels que les classes, les méthodes, les attributs ou les associations.

Certaines adaptations sont comparables au niveau des modèles et au niveau applicatif : il s'agit d'adaptations concernant les éléments communs entre ces deux niveaux, principalement les classes, attributs, méthodes ou paquetages. Les adaptations portant sur les autres éléments spécifiques aux modèles n'ont évidemment aucun équivalent au niveau applicatif.

En effet, plusieurs recherches autour du MDA ont abouti à la définition de langages et outils permettant des opérations sur les modèles telles que la transformation [Bézivin, 04] ou encore la vérification. D'autres opérations, comme la composition ou la fusion demeurent par contre insuffisamment étudiées.

I.1.4. Contribution de la thèse et publications

Les travaux effectués dans le cadre de notre thèse, cherchent à combler les lacunes de l'opération de la composition des modèles, la formalisation du processus de composition sur le niveau structurel et comportemental, ainsi que la proposition d'une prototype pour la composition des modèles dans le cadre de l'approche « The Two Hemisphere Model Driven Architecture » et en se basant sur des webservices afin de mettre en place un plugin qui est capable de s'exécuter dans plusieurs environnements et d'interagir avec plusieurs applications.

Nous montrons que pour atteindre cet objectif, il faut en premier lieu, composer les métamodèles, puis les modèles. Nous montrons dans ce travail comment ces compositions peuvent être réalisées sans modifier les éléments composés, en utilisant ces webservices.

Les architectures webservices permettent de développer, de déployer et de gérer opérationnellement des applications distribuées, constituées de services aux fonctionnalités

complémentaires, potentiellement hétérogènes et interopérables. Les webservices favorisent drastiquement l'indépendance des cycles de vie, qu'il s'agisse des cycles de conception, de développement, ou de déploiement en production.

Pour réaliser ces objectifs, nous avons commencé par développer un prototype conceptuel qui se divise en quatre phases : la première phase de comparaison permet de déterminer si deux entités correspondent en utilisant un ensemble de critères de comparaison définies au niveau du métamodèle. Cependant, nous avons constaté que la définition des types de relations est un passage crucial afin de capturer les différentes correspondances entre les éléments des modèles cibles.

Ces correspondances seront fournies en entrée de l'étape de fusion. Le modèle de correspondances est conforme à un métamodèle qui définit la structure des modèles qui lui sont conformes et fournit les constructions nécessaires à la gestion des relations de correspondance.

La deuxième phase de fusion consiste à l'identification des éléments des modèles décrivant le même concept.

Cette phase nous permet de stocker les différentes relations de correspondances établies entre les éléments des modèles d'entrée et redirigés les conflits que nous pouvons rencontrés au référentiel de la résolution des conflits.

La troisième phase est une phase de vérification qui vise à vérifier certaines propriétés syntaxiques et sémantiques et décrit la chaîne de vérification du modèle de sortie.

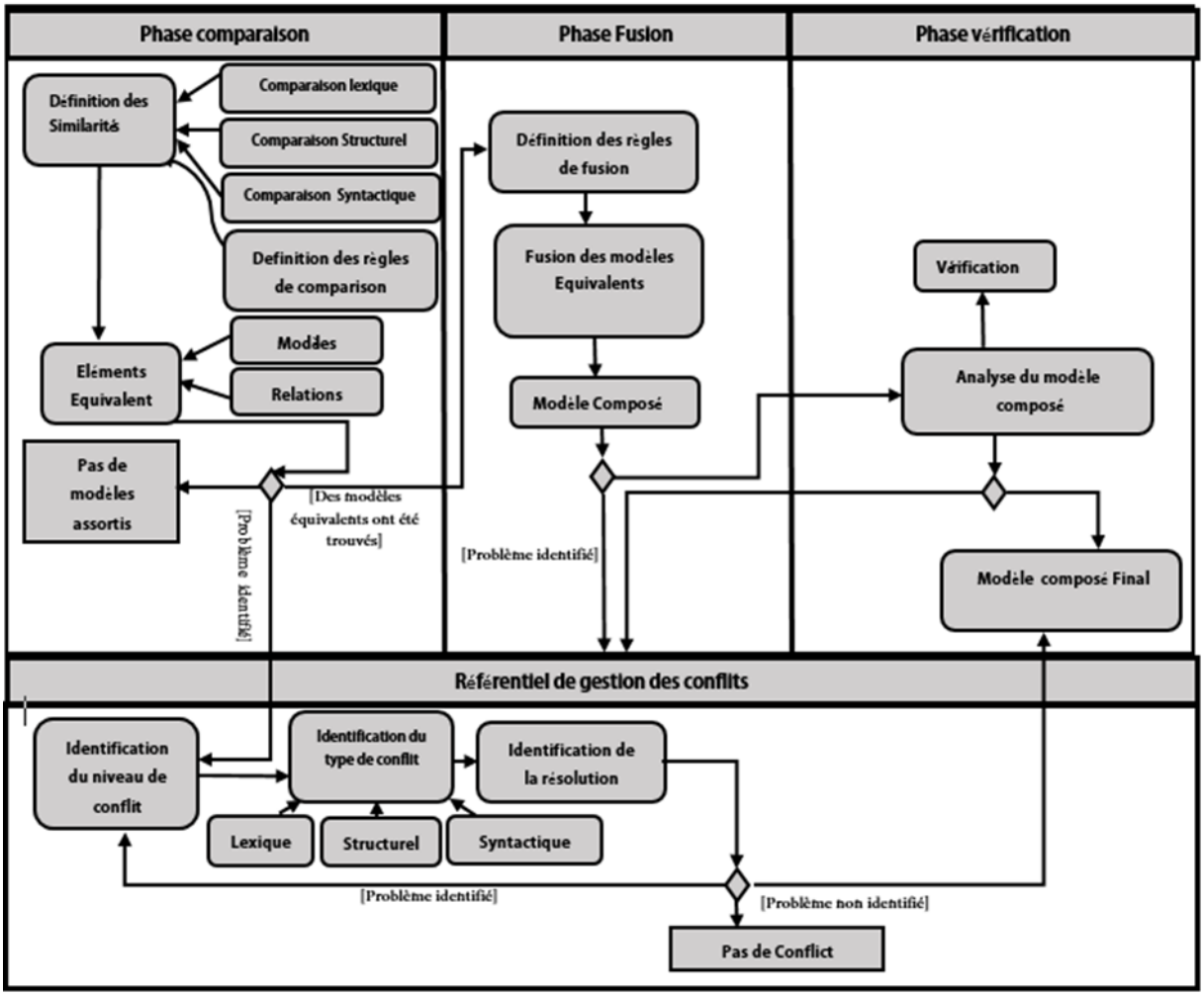


Figure 2 Processus de composition des modèles

La quatrième phase est une phase de vérification de la conformité du modèle composé (Voir Figure 2).

Cependant, une phase transverse de résolution des conflits garantit la cohérence des modèles manipulés durant le processus de composition, dans notre processus cette phase est présentée sous forme d'un référentiel interactif qui permet de résoudre les conflits compositionnels en identifiant pour chaque type de conflit une règle de résolution (Figure 3).

Dans cette phase d'harmonisation des modèles cibles, nous avons identifié trois types de conflits possibles : les conflits syntaxiques, les conflits structurels et les conflits sémantiques.

Le premier type de conflit concerne les conflits causés par la polysémie et la synonymie des classes introduites au cours de la conception des modèles cibles (et qui n'appartenaient pas au dictionnaire initial), ou encore des primitives de classes (opérations ou attributs).

Le deuxième type concerne les conflits structurels entre classes. Nous distinguons ici ceux qui sont liés au type d'association entre deux classes, de ceux qui sont liés à la hiérarchie d'héritage. Les conflits de hiérarchie incluent eux-mêmes les cycles d'héritage (qui peuvent apparaître lorsqu'on veut fusionner les hiérarchies issues des modèles partiels) et les conflits de niveau (classe parente à un certain niveau dans un modèle partiel et à un autre niveau dans un autre modèle partiel).

Le troisième type de conflit concerne les conflits sémantiques qui concernent des éléments de modélisation.

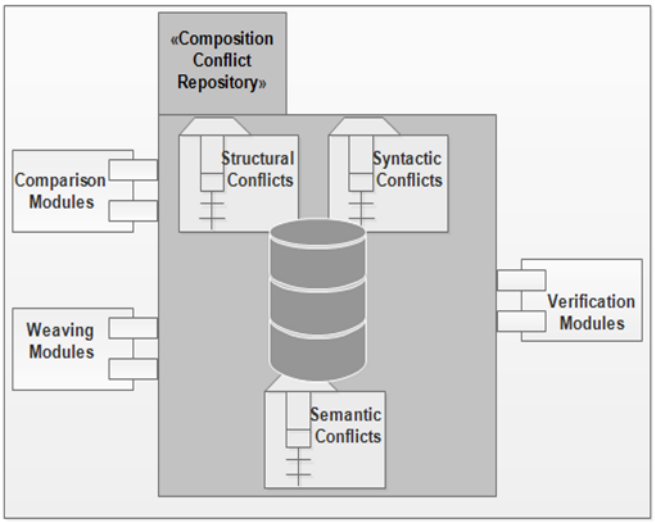


Figure 3 Modélisation graphique de la gestion des conflits de composition

La phase de validation de notre processus a donné lieu à un plugin qui concrétise le concept de la composition des modèles et en utilisant plusieurs language, des appels API et des webservices.

Publications

Dans ce contexte et durant la durée de cette thése, nous avons produits plusieurs publications :

- « El Marzouki N., Lakhrissi Y., El Mohajir M., Nikiforova.O. Enhancing Conflict Resolution Mechanism for Automatic Model Composition ». Web of Science, applied computer systems, Riga Technical University Journal (doi: 10.1515/acss-2016-0006).

- « El Marzouki N., Lakhrissi Y., El Mohajir M. A Comparative Study of Structural Model Composition Methods and Techniques ». David Journal (doi: 10.17265/1934-7332/2016.03.005).

- « El Marzouki Nisrine, Younes Lakhrissi, Oksana Nikiforova, Mohammed El Mohajir, Konstantins Gusarovs. Behavioral And Structural Model Composition Techniques: State Of Art And Research Directions », WSEAS journal. 2017

- « El Marzouki N., Lakhrissi Y., El Mohajir M., Nikiforova.O. Toward a Generic Metamodel for Model Composition Using Model Transformation ». Scopus Procedia Computer Science, volume 104, 2017, Pages 564- 571, Elsevier.

- « El Marzouki N., EL AISSI M., LOUKILI Y.; Lakhrissi Y., El Mohajir M., Nikiforova.O., Implementing a Digital Workspace in the Era of Covid-19 Based on Model Compositon), to 2020 6th IEEE Congress on Information Science and Technology (CiSt).

- « Ayoub KORCHI A., Mohamed Karim KHACHOUCH M., BENJELLOUN S., El Marzouki N., Lakhrissi Y. Toward Moroccan Virtual University: Technical Proposal') for The IEEE international conference on electronics, control, optimization and computer science icecocs'20.

- « El Marzouki N., Lakhrissi Y., El Mohajir M., Nikiforova.O. May 2017 The Application Of An Aytomatic Model Composition Prototype On the- Two Hemishpere Model Driven Approach, DOI: 10.1109/WITS.2017.7934673 Conference: 2017 IEEE International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)

- « Ņikiforova, O., El Marzouki, N., Kuņicina, N., Vangheluwe, H., Florin, L., Iacono, M., Al-Ali, R., Orue, P. Several Issues on Composition of Cyber- Physical Systems Based on Principles of the Two- Hemisphere Modelling», Scopus In: Proceedings of the 4th Workshop of the MPM4CPS COST Action, Poland, Gdańsk, 15-16 September, 2016. Malaga: Departamentos Lenguajes y Ciencias de la Computación Universidad de Málag, 2016, pp.44-55

- « El Marzouki N., Lakhrissi Y., El Mohajir M. A Study of Behavioral and Structural Composition Methods and Techniques ». IEEE Explore (doi: 978-1-4673-7689-1/16/2016 IEEE)

I.1.5. Objectifs de cette thèse

Ce travail de recherche se situe dans le cadre de l'Ingénierie Dirigée par les Modèles (IDM). Plusieurs recherches autour de l'IDM ont abouti à la définition de langages et outils permettant des opérations sur les modèles telles que la transformation ou encore la vérification.

D'autres opérations, comme la composition ou la fusion demeurent par contre insuffisamment étudiées. Notre travail se focalise sur la définition de l'opération de composition de modèles de conception issus d'une phase de conception décentralisée et vise l'élaboration d'une démarche pour la fusion des modèles.

L'objectif de la thèse est de formaliser l'opération de composition de modèles de conception.

Le cadre général de ce travail s'inscrit dans la continuité des travaux autour de la méthodologie « The Two hemisphere Model Driven Architecture » [NKA+15].

I.1.6. Organisation de la thèse

Le travail de thèse est présenté dans ce mémoire selon six chapitres.

Le Chapitre II propose une description synthétique du MDA. Il en rappelle dans une première étape les grandes lignes de notre sujet et présente ensuite l'approche MDA de l'OMG [OMG09] comme exemple d'une approche de développement orientée modèle. Une classification des types de transformations est également présentée dans ce chapitre qui se conclut par un positionnement de l'approche Two Hemisphere Model Driven Architecture [NKA+15] dans l'architecture MDA.

Le Chapitre III décrit un état de l'art sur la composition de modèles. Il présente d'abord la notion de composition dans le standard UML2 [UML11], puis dresse une taxonomie des travaux autour de la composition dans les approches de multi-modélisation.

Le Chapitre IV est dédié à la présentation conceptuelle de notre proposition. Il décrit tout d'abord une approche générique de composition de modèles sous la forme d'un processus dirigé par des règles de transformation, puis propose une spécialisation des règles pour réaliser la composition en se basant sur de l'approche « The Two Hemisphere Model Driven Architecture » [NKA+15]. Une formalisation de l'opération de composition est également décrite dans ce chapitre.

Enfin, nous positionnons notre méthode dans cette approche et montrons la valeur ajouté de la composition suivant les étapes de cette approche.

Le Chapitre V détaille les aspects applicatifs de notre approche. Il décrit premièrement les étapes de réalisation de notre méthode de composition en utilisant le concept des webservices. En deuxième lieu, nous montrons l'utilisation de ce concept dans la composition de modèles à travers plusieurs interactions entre des applications qui utilisent le même plugin développé.

Enfin, dans **le Chapitre VI**, nous concluons ce mémoire en récapitulant les points forts de l'approche tout en en indiquant aussi ses limites. Nous détaillons également les perspectives offertes par ce travail et les voies de recherche actuelles et futures.

I.2. General Context and scope

The research work presented in this thesis focuses on Model Composition in the framework of Model Driven Architecture (MDA) and is a continuity of «The-Two hemisphere Model Driven Architecture" methodology developed within our research team.

. Computer systems have never been more central to the corporate strategy today. The features they offer, ease of use, reliability, performance and robustness are the queens qualities that allow companies to be competitive.

Indeed, actually we are finding that a real software system is too complex to be described by a single model, which makes any global reasoning about the system difficult. In this context many models should be created to specify a complex system, either at various levels of abstraction, various points of view, or according to different and complementary functional domains

To cope with the complexity of the execution platforms, it was necessary to define an approach to tame complexity. This approach should be flexible and generic in order to adapt to any type of platform.

Guided by software engineering models, or MDA (Model Driven Architecture), corresponds to the definition of such an approach. MDA apply separation of concerns between the business logic of computer systems and used platforms and is based on the massive use of models.

The purpose of model composition is to facilitate the joint use of heterogeneous models during the development cycle.

Indeed, the composition is always inherent in the preceding decomposition phase. In the literature the decomposition can be done in different ways; let us quote in particular the decomposition by point of view [Finkelsetin et al., 92 ; Kriouile, 95 ; Mili et al., 01], by aspect [Kiczales et al., 97], by subject [Ossher et al., 95] and by role [Pernici, 90 ; Kristensen et al., 96].

The decomposition by point of view aims to analyze and design by objects of complex systems with an actor-centered approach. Each actor in the system has a point of view associated with it.

A set of views is associated with each point of view that applies to the system.

The result of such a modeling is a unique, shareable model, accessible from several points of view. The breakdown by aspects is based on the separation between functional concerns and so-called "transversal" concerns during software development. The idea of aspect-based modeling arises from aspect-based programming and proposes to consider aspects in models. Another separation technique is subject decomposition.

This approach is based on a multidimensional separation of concerns, making it possible to cover different types of concerns (business, technological, management rules, etc.). It makes it possible to identify a set of specifications and behaviors reflecting the perception of the real world corresponding to a generic vision of an actor.

In role decomposition, an object interacts with the world subjectively from the perspective of the user's context. The appearance of the notion of role comes from the fact that the extrinsic properties of an object can change over time. Indeed, an object can be subject to multiple classifications during its life cycle, and each time it plays a role adapted to a particular situation.

In this context, several approaches exist; they differ in particular in terms of the type of composition operators offered. The adaptations on which the composition is based relate to the constituents of models such as classes, methods, attributes or associations.

Some adaptations are comparable at the model level and at the application level: they are adaptations concerning the elements common between these two levels, mainly the classes, attributes, methods or packages. The adaptations relating to the other elements specific to the models obviously have no equivalent at the application level.

Indeed, several researches around MDA have led to the definition of languages and tools allowing operations on models such as transformation [Bézivin, 04] or verification [Ober et al., 06]. Other operations, such as composition or fusion, however, remain insufficiently studied.

I.3. Contribution

The main goal of this thesis is to formalize the operation of model composition in order to design complex systems according to the vision of business partners.

The work carried out during research focuses on the definition of a specific process of model composition to design models resulting from a decentralized modeling phase and aims to develop an approach for the fusion models, while defining a repository to manage conflicts, ensure conformity between models made and guarantee the consistency of the assembly throughout the composition process.

To achieve these goals, we set up a composition prototype that starts with a comparison phase in order to determine the feasibility of matching two instances using a set of comparison criteria and syntax properties.

This study allows us to capture the different correspondences between the elements of the target models, which will be provided as an input of the merge step and should conform to a metamodel that defines the structure of the models and provides the main constructs to manage the correspondence relationships and generated conflicts.

The Composition of several class diagrams coming from different source justify our motivation for this prototype based on Model Driven Architecture and the two-hemisphere model driven approach, which propose the use of business process modelling and concept modelling to represent the systems in platform independent manner and describes how to apply transformations from business process model into UML models [UML11].

We show that to achieve this goal, we must first compose the metamodels, then the models. We show in this work how these compositions can be performed without modifying the compound elements, using webservices, which is an architectural approach for application development.

A webservices architecture differs from a classic monolithic approach in that it breaks down an application to isolate key functions. Each of these functions is called a "service" and these services can be developed and deployed independently of each other. Thus, each can operate without affecting the others.

Webservices [Babris et al,19] propose an architectural style where applications are decomposed into small independent building blocks (the webservices), each of them focused on a single business capability. Webservices communicate with each other with lightweight mechanisms and they can be deployed and maintained independently, which leads to more agile developments and technological independence between them [Babris et al,19]. The decomposition of a system into webservices forces developer teams to build webservice compositions to provide their customers with valuable services [21]. It seems that the decentralized nature of webservices makes the choreography approach more appropriate to define these compositions [Babris et al,19].

This approach is first validated by the creation of a plugin based on the webservices approach. This plugin or service is able to run in multiple environments and interact with multiple applications.

I.4. Organization

The thesis work is presented in this dissertation in six chapters.

Chapter II provides a synthetic description of MDA. In a first step, he recalls the main lines and then presents the OMG's MDA [OMG09] approach as an example of a model-oriented development approach. A classification of the types of transformations is also presented in this chapter which

concludes with a positioning of the Two Hemisphere Model Driven Architecture approach in the MDA architecture.

Chapter III describes a state of the art on the composition of models. It first presents the notion of composition in the UML2 standard [UML11], then draws up a taxonomy of work around composition in multi-modeling approaches.

Chapter IV is dedicated to the conceptual presentation of our proposal. It first describes a generic model composition approach in the form of a process driven by transformation rules, then proposes a specialization of rules to achieve the composition by going on the Two Hemisphere Model Driven Architecture approach. . A formalization of the composition operation is also described in this chapter. Finally, we position our method in this approach and show the added value of the composition following the steps of this approach.

Chapter V details the application aspects of our approach. It first describes the stages of our composition method using the concept of webservices. Second, we show the use of this concept in the in model composition through several interactions between several applications.

Finally, in **Chapter VI**, we conclude this thesis by recapitulating the strengths of the approach while also indicating its limitations. We also detail the perspectives offered by this work and current and future research avenues.

CHAPTER II. MOTIVATION AND PROBLEM STATEMENT

II.1. Introduction

Model Driven Architecture (MDA) is a branch of software engineering which aims to operationalize and capitalize the model concept. For this, the MDA is based on various concepts such as meta-models, modelling languages and model transformations, the main goal of this approach is to reduce the existing gap between software dedicated to a particular business and a specific technological platform on which the software must be running.

For this, the MDA initiative considers the software development process according to various models:

- A model independent of software aspects called CIM (Computer Independent Model) which contains information relating to the business domain,

- A platform independent model called PIM (Platform Independent Model) which contains information relating to the business domain of the application,

- A platform description model called PDM (Platform Description Model) which contains information relating to the integration of the PIM within the platform,

- A platform specific model or PSM (Platform Specific Model) which contains all the technical details related to the realization.

More precisely, the approach recommended by the MDA initiative consists, from a model dedicated to a business and according to one or more specific deployment contexts, to obtain by refinement a model "adapted" to a particular platform. This approach is then based on a Y cycle.

II.2. Model Driven Architecture

Models: offer many advantages. Those who practice or any other UML modeling language familiar with them [UML11]. The most important benefit they provide is to specify different levels of abstraction, helping manage the inherent complexity of applications.

The very abstract models are used to present the general architecture of an application or a place in an organization, while very concrete models can accurately specify network communication protocols or timing algorithms. Even if the models are at different levels of abstraction, it is possible to express refinement relations between them. True traceability links, these relations are guarantees the consistency of a set of models representing an application.

The variety of modeling capabilities and the ability to express links traceability are decisive assets to manage complexity. Another clear advantage of models is that they can be presented in size graphic, thereby facilitating communication between the actors of IT projects. The graphical models used are among the most relational models, which allow to specify the structure of the databases. The graphical

representation of these models offers a significant productivity gain. The gossips say that model is the best way to lose time because, ultimately, we need to write code anyway. Similarly, the famous saying stating that a good diagram is worth a thousand words, sometimes we hear replicate that a scheme can match more than a thousand speeches, depending on how we the interpreter. These criticisms are aimed right in the absence of knowledge of good modeling practices, that is to say, the model engineering. This is why it is essential to acquire good modeling practices to determine how, when, what and why model and to fully exploit the advantages of the models. The OMG [OMG09] has defined MDA (Model Driven Architecture) in 2000 for this purpose. The approach MDA recommends the widespread use of models and offers the first answers how, when, what and why model. Without claiming to be a Bible modeling, listing all the good practices, it aims to highlight the qualities intrinsic models, such as sustainability, productivity and integration of platforms execution. MDA includes the definition of several standards including UML, MOF and XMI [UML11].

OMG: The OMG (Object Management Group) is a non-profit consortium of industrialists and researchers, which aims to set standards to solve interoperability problems of information systems. [OMG, 03a]

MDA concept: MDA is to develop models of business logic independently of implementing systems platforms and to transform these models to automatically dependent models of platforms. The complexity of platforms no longer appears in the business logic models but is found in the transformation [Bézivin, 04] [MDA, 06].

The advantages of MDA are therefore the perpetuation of the enterprise business logic through the development of models, the productivity of this business logic through automation transformations models and the integration of platforms for performance through the integration of these into model transformations.

The principle key of MDA is the use of models for different phases of the application development cycle. Specifically, MDA advocates the development of requirements models (CIM) analysis and design (PIM) and code (PSM).

The main goal of MDA is the development of sustainable models, independent of the technical details of execution platforms to enable automatic generation of the entire code and applications to achieve a significant increase in productivity.

Other models, such as models of supervision, audit or corporate organization, are not yet integrated into the MDA approach but will be quickly without difficulty, given its openness to new researches. [MDA, 06]

General architecture of MDA approach: Figure 4 provides an overview of the MDA approach. We note that the construction of a new application begins with the development of one or more requirements models (CIM). It continues with the development of analytical models and abstract design of the application (PIM). These should in theory be partially generated from the CIM so that traceability links are established. PIM models are perennial models, which do not contain information on implementation platforms.

To effectively carry out the application, you must then build specific models of implementing platforms. These models are obtained by a transformation of PIM by adding the technical information related to platforms.

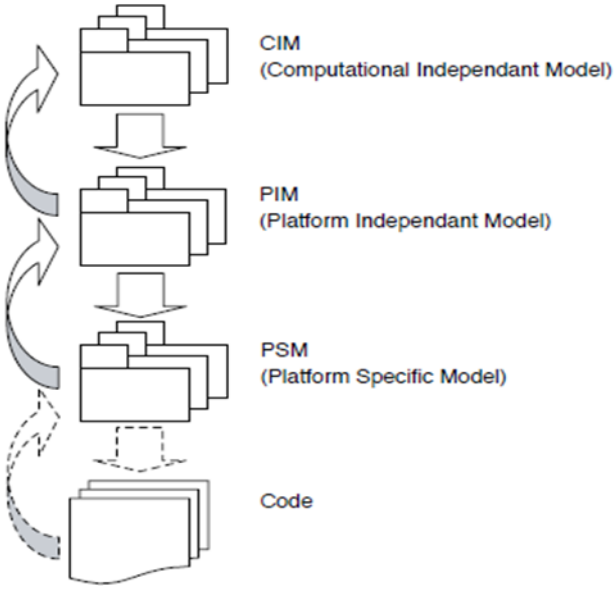


Figure 4 Overview of MDA approach [MDA, 06]

II.2.1. The CIM requirements model (Computation Independent Model)

The first thing to do when building a new application is understood to specify client requirements. Although very early, this step should greatly benefit models.

The goal is to create a model of the future application requirements. Such a model should represent the application in its environment to define the services offered by the application and what other entities with which it interacts.

Creating a requirements model is of paramount importance. This helps to articulate the traceability links with models that will be built in other phases of the application development cycle, such as analysis and design models. A lasting connection is created with the application of customer needs.

The requirements models can even be considered as contractual elements, to be used as reference when we want to ensure that an application meets the customer requirements.

It is important to note that a requirements model contains no information on the implementation of the application or on treatments. Therefore, in the MDA vocabulary, requirements models are called CIM (Computation Independent Model), literally "independent model of programming." [MDA, 06]

With UML, a requirements model can be summarized in a use case diagram. These contain indeed the functionality provided by the application (use case) and the various entities that interact with it (actors) without providing information on the operation of the application.

II.2.2. The analysis model and abstract design PIM (Platform Independent Model)

Once the requirements model is made, the work of analysis and design can begin. In the MDA approach, this phase also uses a model. The class diagram is included. Analysis and design are for more than thirty years the steps where modeling is most present, first with the methods Merise and Coad / Yourdon then with object methods Schlear and Mellor, OMT, Booch and OOSE. These methods all offer their own models. Today, UML has become the reference for performing all analysis and design models. By design, it is meant the step to structure the application into modules and sub-modules. The application of design patterns, or Design Patterns, GoF (Gang of Four) is part of the design stage. For cons, the application of technical patterns, specific to certain platforms, represents another step. We therefore consider here the abstract design, that is to say one that is achievable without any knowledge of implementation techniques.

II.2.3. The model code or concrete design PSM (Platform Specific Model)

Once the analysis and design models produced, the code generation work can begin. This phase, the most delicate of MDA, must also use templates. It includes the application of technical design patterns.

MDA considers the application code can be easily obtained from code patterns. The main difference between a model code and an analysis or design model lies in the fact that the code pattern is bonded to a platform of execution. In the MDA vocabulary, these code templates are called PSM (Platform Specific Model).

Code models are used primarily to facilitate code generation from an analysis and design model. They contain all the information necessary to operate a platform for execution, such as information for manipulating file systems or authentication systems.

It is sometimes difficult to differentiate between application codes templates. For MDA, the application code boils down to a series of text lines as a Java file, while a code model is rather a structured representation including, for example, loop concepts, condition, instruction, component , event, etc.

The writing code from a code pattern is therefore a relatively trivial operation. To develop code templates, MDA proposes, inter alia, the use of UML profiles. A UML profile is an adaptation of UML to a particular area. For example, the UML Profile for EJB is an adaptation of UML to the field of EJB. With this profile, it is possible to develop code templates for EJB development.

The role of code templates is mainly to facilitate the code generation. They are essentially productive but are not necessarily sustainable. The other important feature of code templates is that they make the connection with the execution platforms. This notion platform execution is very important MDA for it is that defines the famous separation of concerns [TAR+99].

II.2.4. MDA and UML

MDA believes that the analysis and design models must be independent of any platform implementation. By not incorporating the implementation details until very late in the development cycle, it is possible to maximize the separation of concerns between application logic and implementation techniques [TAR+99].

UML is recommended by the MDA approach as the language to use to carry out independent analytical models and design implementation platforms.

This is why in the MDA vocabulary these models are called PIM (Platform Independent Model).

Note that MDA only advocate the use of UML and does not exclude that other languages may be used. In addition, MDA gives no indication as to the number of models to develop or as to the method used to develop these PIM.

Whatever the language used or the role of analysis and design models is to be sustainable and to link the requirements model and the application code. These models must also be productive since they are the foundation of all the code generation process defined by MDA. The productivity of the PIM means they must be sufficiently precise and contain sufficient information for an automatic code generation is possible.

Transformation models: we have to review the three types of the most important models for MDA that are the CIM, PIM and PSM. We also saw that it was important to establish traceability links between these models. In fact, MDA establishes these links automatically through the implementation of transformation models.

Model transformations are essentially advocated by MDA transformations from CIM to PIM and PIM to PSM. MDA is also considering the inverse transformations: code to PSM, PSM to PIM and PIM to CIM.

We cannot over emphasize the importance of model transformations. It is they who bear the intelligence of the methodological application building process. They are strategic and are part of the know-how of the company or organization that executes, because they hold application development quality rules. Aware of this, MDA advocates modelling model transformations themselves. After all, a model transformation can be considered as an application. It is therefore natural to model its demands, its analysis and its design and code templates to automatically generate the code for processing.

Modeling technologies: We have seen the rule of models in the MDA approach. This is what mainly differentiates the classic software engineering approaches such as OMT (Object Management Technique) OOSE (object-oriented software engineering) or BCF (Business Component Factory), which place objects or components in the foreground. We have also seen that MDA called for the development of different models of CIM requirements model, analysis model and abstract design model code and PIM and PSM concrete design. In reality, MDA is much more general and advocates model any information necessary for the application development cycle. So we can find test model, deployment, platform, etc. To structure this set of models, MDA defines modeling formalism.

The modeling formalism MOF (Meta Object Facility): in modeling formalism is a language to express models. Each model is expressed in a modeling formalism. The models have their own requirements formalism, which is different from the formalism expression analysis models and abstract design. Remember also that the formalism advocated for expression analysis and design models is

UML. Formalism defines the concepts and relationships between concepts necessary for expression patterns. We will return largely on this subject later in the book. The expression UML analysis and design models formalism defines, among others, the concepts of class and object and the relationship indicating that an object is an instance of a class. The concepts of models and modeling formalism are not sufficient to implement MDA. We saw that it was also very important to express traceability links and transformations between models. To do this it is essential to work not only at the models, but also in the modeling formalisms. We must express the relationship between the concepts of the different formalism. For example, we need to express the UML class concept must be transformed into the concept of Java class. For that, MDA advocates model modeling formalisms themselves. The objective is to have a formalism expression modeling formalisms models. In the jargon of MDA, such formalism is called a metaformalism, and models that can express are called Meta. So we can make an analogy between meta-models and modeling formalisms. The question that then arises is whether it is possible to build a metametaformalism for expressing metaformalism or a metametameta ... -formalism, and so on. MDA responds only three levels are required:

Model, modeling formalism, also called metamodel, and metaformalism. To stop the rise in meta levels, MDA causes the metaformalism either to itself its own formalism. A metametaformalism is therefore no longer necessary. In MDA, there is only one metaformalism, the MOF (Meta Object Facility). Also metametamodel called the MOF can express modeling formalisms, or meta, allowing themselves to express models. Figure 5 illustrates this concept model, modeling formalism (meta) and metaformalism (metametamodel). In the rest of the book, we use rather MDA vocabulary, that is to say, model, metamodel and metametamodel.

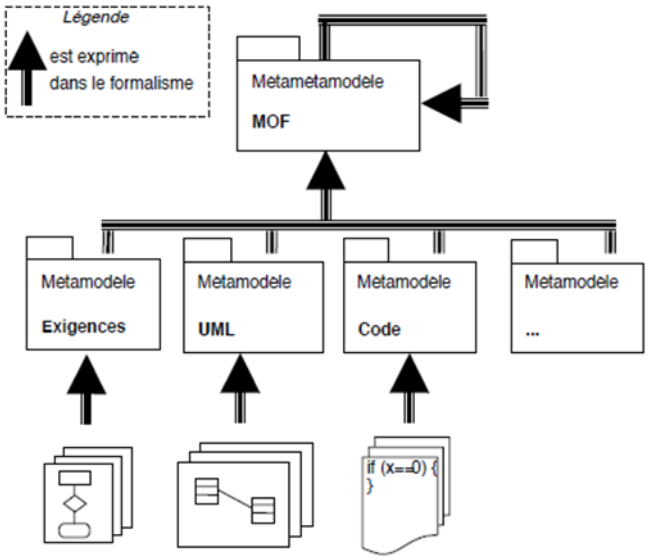


Figure 5 Model, metamodel and meta-meta model [MDA09]

The idea of a Meta formalism is not new in itself to that is accustomed to manipulate language grammars. In the XML world, a formalism is represented as XML schema. An XML schema defines the concepts and relationships between concepts allowing the expression of XML documents, and XML schemas are themselves XML documents. This is only possible because there is a schema called schema XML, ie a metaformalism. In the XML world, the schema for XML schemas is also the last level as it is in itself its own schema. The same analogy can be made between programming languages

and the BNF (Bachus Naur Form), the language used to express the syntax of languages. In almost all reference manuals languages (Java, Ada, C ++, etc.), there is a BNF definition. For example, a Java program is expressed dance Java formalism that is defined by a grammar in BNF, the latter defining itself.

Meta model UML: the previous section introduced the model concepts, and metametamodel metamodel. We have seen that MDA advocated the use of different models and each model was consistent with a metamodel, itself conforms to metametamodel MOF. The MDA universe is partitioned by a set of meta. Each of these meta is dedicated to a particular stage of MDA (requirements, analysis and design, code). From a purely theoretical point of view, MDA imposes no constraints on the use of a particular metamodel for each of these steps. This does not hold for the realization, since MDA currently advocates the use of the UML metamodel for the analysis stage and abstract design and advises to use UML profiles to develop code templates and practical design from UML models.

UML Meta model defines the structure that must have any UML model. It specifies, for example, a UML class may have attributes and operations. From a conceptual perspective, the UML metamodel allows to develop models describing object applications. UML defines several diagrams to describe the different parts of an application object. For example, class diagrams for describing the static part of the application while the sequence diagrams or activities used to define the dynamic part.

UML models are independent of execution platforms. Several market tools offer code generators to the various programming languages.

For all these reasons, it is clear that the UML metamodel is an ideal metamodel for the development of PIM (Platform Independent Model). Recall that PIM is a model of analysis and design of an application and it must be independent of platform performance. (See Figure 6)

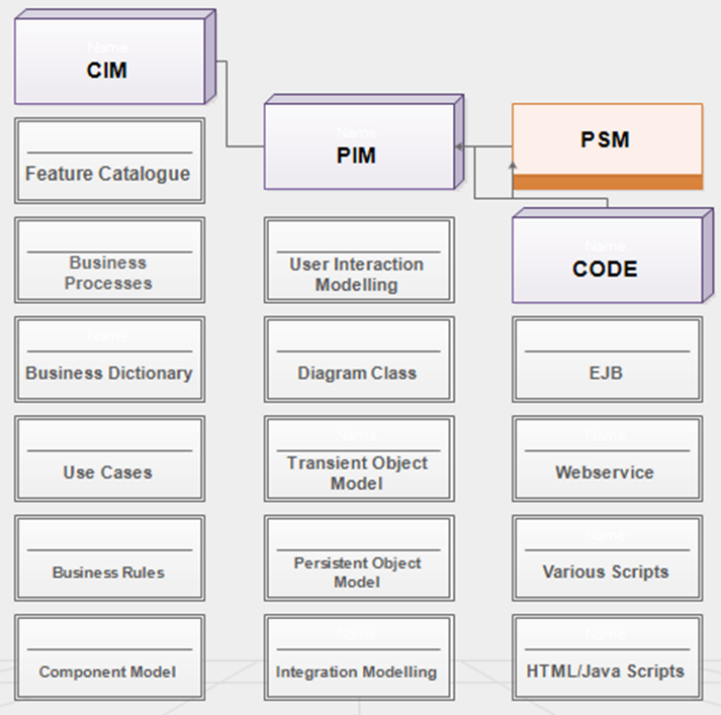


Figure 6 MDA layers [MDA09]

II.3. Two Hemisphere Model Driven Approach

II.3.1. The title of the proposed strategy

The title is derived from cognitive psychology. Human brain consists of two hemispheres: one is responsible for logic and another one for concepts. Harmonic interrelated functioning of both hemispheres is a precondition of an adequate human behavior. A metaphor of two hemispheres may be applied to software development process because this process is based on investigation of two fundamental things: business and application domain logic (processes) and business and application domain concepts and relations between them. [NKA+15]

II.3.2. Elements of two hemisphere model

Two-hemisphere model driven (2HMD) approach [NKA+15] proposes the application of business process modeling and concept modeling to represent systems in the platform independent manner and describes how to transform business process models into UML models. For the first time the strategy was proposed in [NKA+15], where the general framework for object-oriented software development was presented and the idea of the usage of two interrelated models for software system development was stated and discussed. The strategy supports gradual model transformation from problem domain models into program components, where problem domain models reflect two fundamental things: system functioning (processes) and structure (concepts and their relations).

• Business process diagram/ Process – business process usually means a chain of tasks that produce a result which is valuable to some hypothetical customer. A business process is a gradually refined description of a business activity (task). Task is an atomic business process unit, which actually describes some step or function and is done by a Performer. (See Figure 7, 8 and 9)

• Concept model/Concept – conceptual classes that are software (analysis) class candidates in essence. A conceptual class is an idea, thing, or object. A conceptual class may be considered in terms of its symbols – words or images, intensions – definitions, and extensions.

• Class diagram/Class – a class is the descriptor for a set of objects with similar structure, behavior, and relationships.

So that why it is necessary to find the way how source model elements can be transformed into target model elements according to the definition of transformations in the framework of MDA.

II.3.3. Description of possible transformations

Analysis of two hemisphere model and application of two hemisphere model for knowledge architecture development in the task of study program development makes to think that notational conventions of UML communication diagram is more suitable for definitions of formal transformations of two hemisphere model into object interaction and then into class diagram, than using of UML sequence diagram. Although the aspect of time sequence, which is a component of UML sequence diagram and is not shown in communication diagram, is missed in this case. We are

investigating the possibility to save time aspect in transition from two hemisphere model into class diagram through the defined transformations.

Intermediate model is used to simplify the transition between business process model and model of object interaction, which is presented in the form of UML communication diagram.

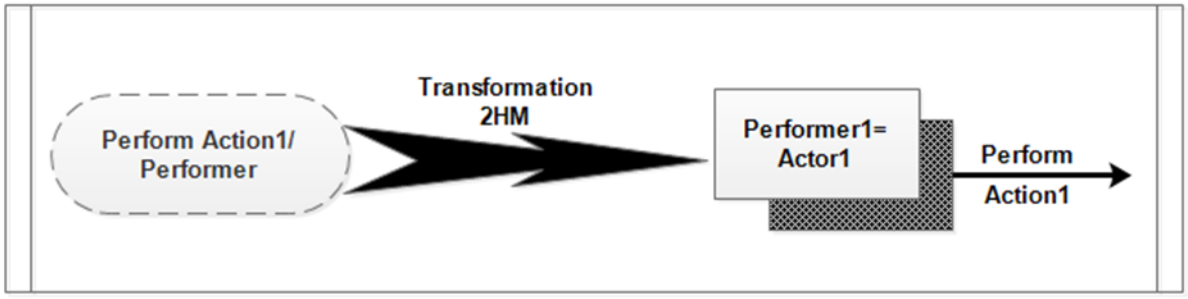


Figure 7 Transformation from business process model into intermediate model

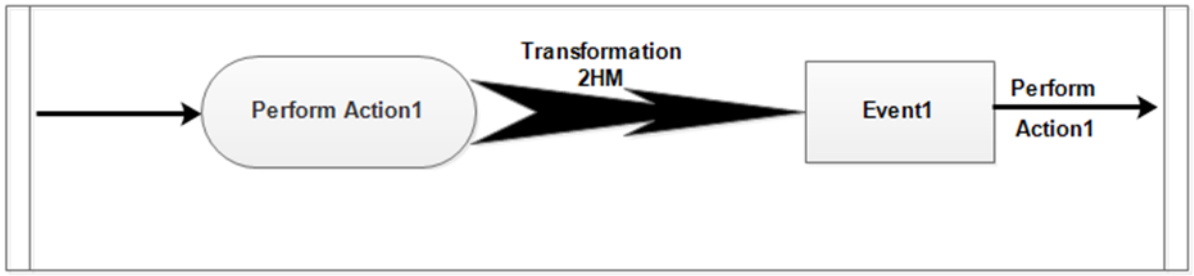


Figure 8 Transformation from business process model into intermediate model

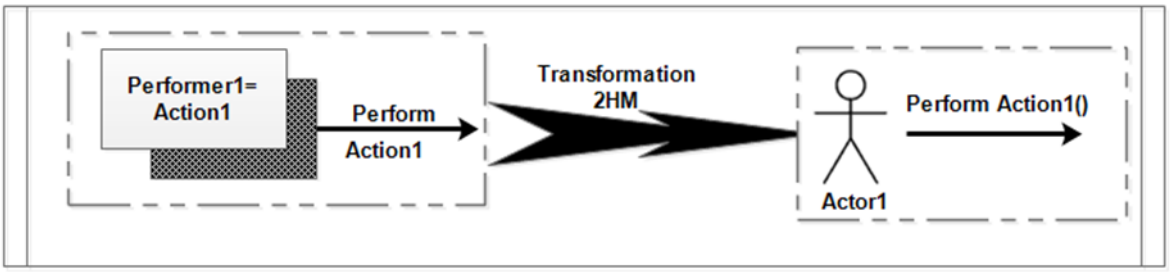


Figure 9 Transformation from intermediate model and concept model into object communication diagram

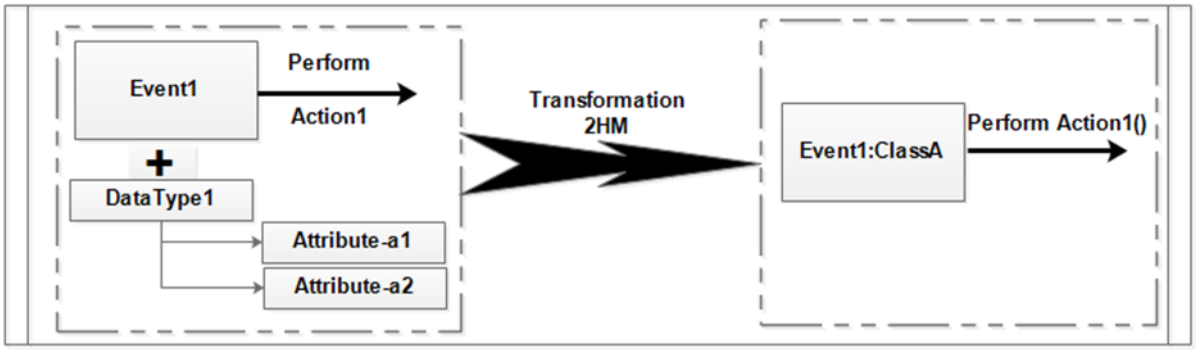


Figure 10 Transformation from intermediate model and concept model into object communication diagram

II.4. Two Hemisphere Model and Model Composition

II.4.1. Composition and Decomposition

The model composition is a new research topic in the MDA. The work is ongoing development and evolution. So there is still no mature foundation to date for this. Our goal through this part is to study existing models of composition approaches by analyzing and identifying 1) what are the elements involved in the composition process, and 2) how the model composition is made in these approaches. The ultimate goal is to arrive at an understanding of what is done for model composition in these approaches.

"Model composition is an operation that combines two or more models into a single one."

"Model composition in its simplest form refers to the mechanism of combining two models into a new one." [Vil03] [Szy98] [Boo87]

According to this, it can be said that the composition model is a process that takes two or more input models, integrates them through an operation and composition to produce a composite output model.

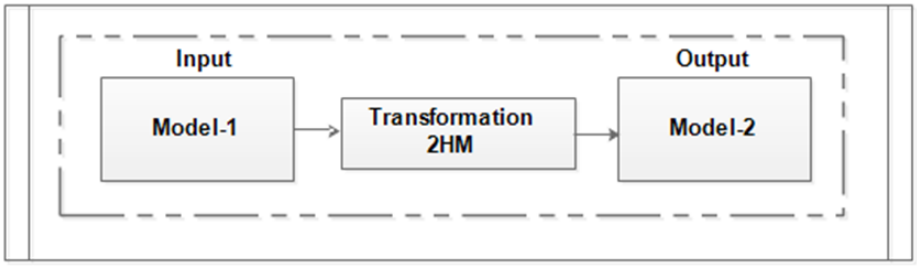
However, this scheme is very abstract. No assumptions about the input models, output, or on the compositing operation is expressed. In practice, each approach must specify these assumptions for its work context. These also include the differences to classify approaches.

Composition Language: The composition of elements need formalisms to express them. These formalisms are very diverse because each approach has its own elements of composition. They can be a weaving language, a metamodel of composition rules, and a UML profile for model composition. Despite their diversity, they can usually assess a compositional formalism on two points: the composition that provides abstractions and scalability.

Mechanism of composition: melting, replacing the union, weaving etc.

Element composition: what are the additional elements involved in the composition. There are two classification axes: the type and formality of these.

Composition Language: The composition of elements need formalisms to express them. These formalisms are very diverse because each approach has its own elements of composition. They can be a weaving language, a metamodel of composition rules, and a UML profile for model composition. Despite their diversity, they can usually assess a compositional formalism on two points: the composition that provides abstractions and scalability.



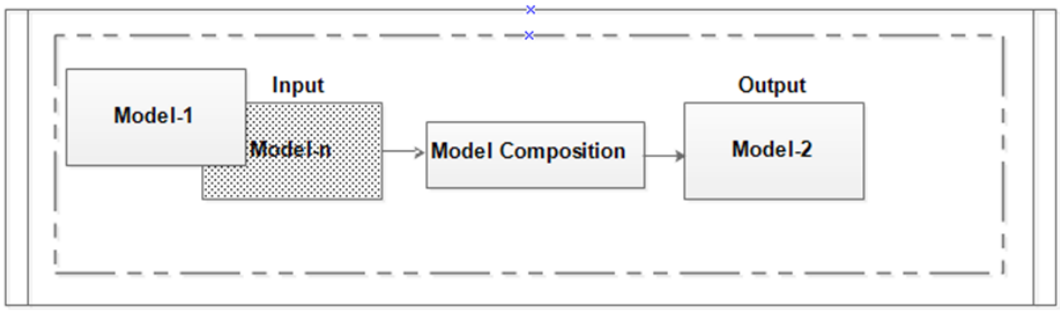


Figure 11 the two Hemisphere model driven architecture in the context of model composition

The idea of decomposition methodology for classification tasks is to break down a complex classification task into several simpler and more manageable sub-tasks that are solvable by using existing induction methods, then joining their solutions together in order to solve the original problem. Decomposition methodology can be considered as an effective strategy for changing the representation of a classification problem. Indeed, [Vil03][Le04][Veg05] considers decomposition as the most useful form of transformation of data sets". (See Figure 12)

II.4.2. Investigations based on two hemisphere model approach

A transformation tool takes a PIM and transforms it into a PSM. A second (or the same) transformation tool transforms the PSM to code. These transformations are essential in the MDA development process. The transformation tool takes one model as input and produces a second model as its output. There is a distinction between the transformation itself, which is the process of generating a new model from another model, and the transformation definition. The transformation tool uses the same transformation definition for each transformation of any input model. A transformation is defined in [Bézivin et al., 05] as the automatic generation of a target model from a source model, according to a transformation definition. (See Figure 11)

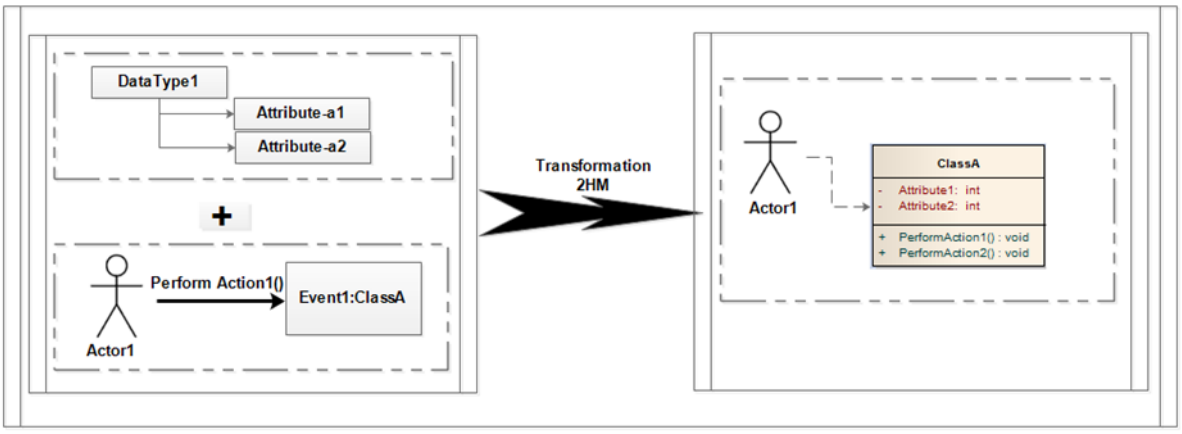


Figure 12 Transformation from intermediate model and object communication diagram into Class diagram

A transformation is defined as a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. (See Figure 13)

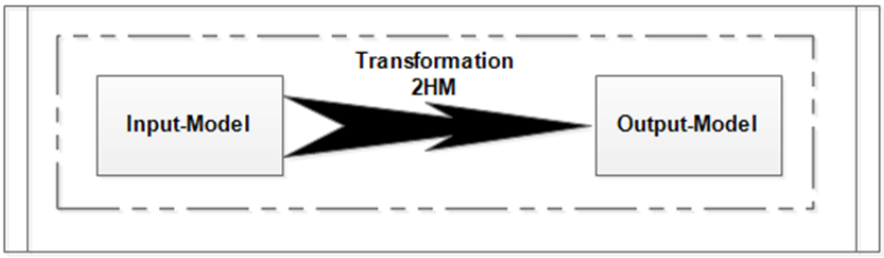


Figure 13 Two Hemisphere model transformation

A transformation tool or approach takes a model on input, so called source model, and creates another model, so called target model, on output. The two hemisphere model has been marked as input with mapping rules, the class diagram and transformation trace has been received on output. Transformation trace shows the plan how an element of the two hemisphere model is transformed into the corresponding element of the class diagram, and which parts of the mapping are used for transformation of every part of the two hemisphere model.

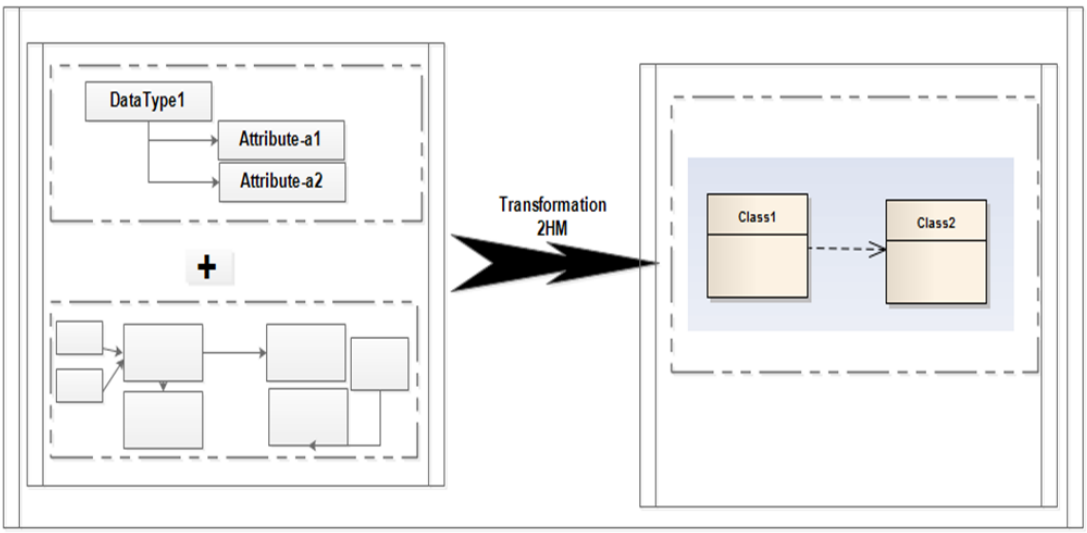


Figure 14 Transformation from two hemisphere model into class diagram

The Figure 14 shows how a transformation tool takes input –the two hemisphere model and receives output – the class diagram. Therefore implementation of model transformation (in our case transformation from two hemisphere model into class diagram) needs well-defined set of notational elements of source model, well-defined set of notational elements of target model and definition for transformation of elements of one model into elements of another one.

So the two hemisphere approach addresses the construction of information about problem domain by use of two interrelated models at problem domain level, namely, the process model and the conceptual model. The conceptual model is used in parallel with process model to cross-examine software developers understanding of procedural and semantic aspects of problem domain.

So it's clear here that moving from CIM to PIM is not an easy task. MDA itself say a little about this transformation. Therefore our case now is when a team members works on models of different parts of the project at the same time. This topic suggests a decomposition of the application into different parts that correspond to the layers in an overall layering diagram. So the question is how we can compose it? The diagram of the Figure 15 illustrates this idea:

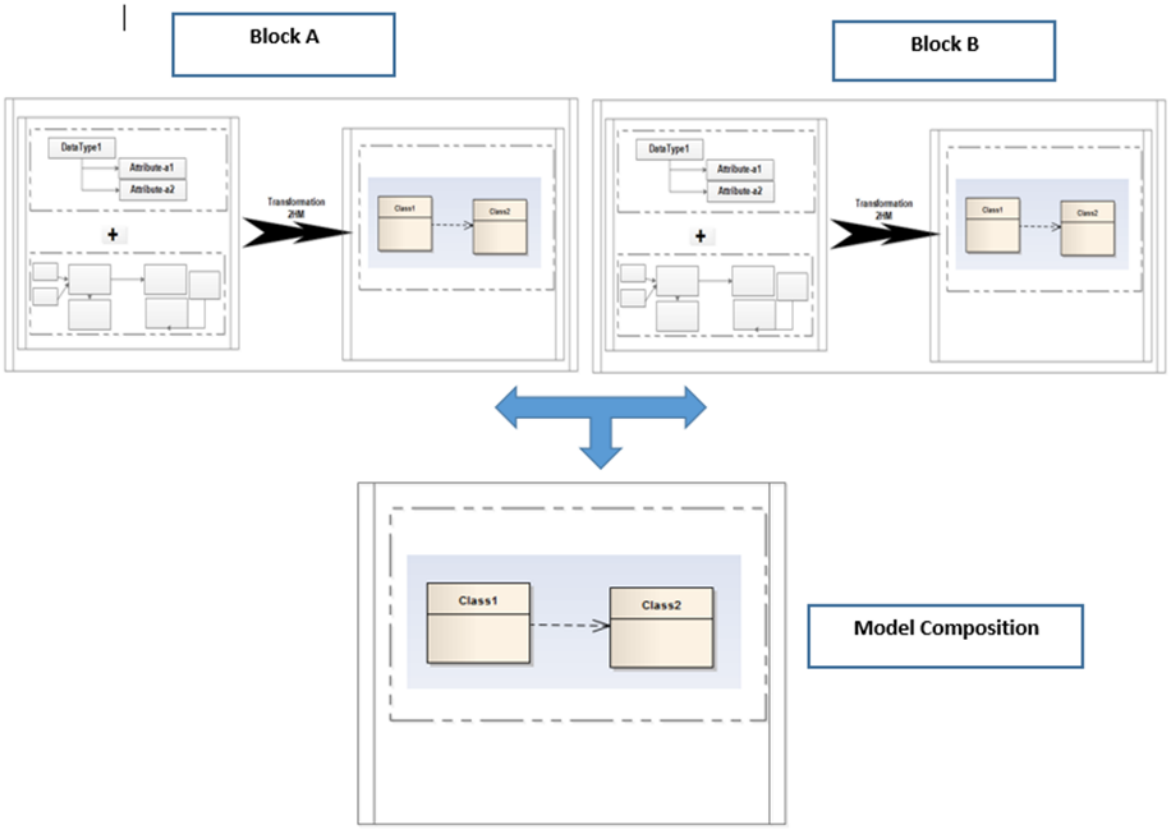


Figure 15 a model composition process based on the two hemisphere model methodology

II.5. Conclusion

This part shows that the MDA provides a framework for the operationalization of models or the integration of modeling languages. The use of an IDM approach in a particular area requires knowledge of a number of technologies.

Despite this drawback, the MDA provides an elegant framework for modeling and integrating the different expertise involved in a development process.

In particular, the languages used by an activity or a set of activities can be expressed using a metamodel, that is to say models which no longer describe a system but modeling concepts. These meta-models can then be associated with concrete graphic and or textual syntaxes.

The meta-tools offered by the MDA can then be:

1. Configured with these meta-models.

2. Used as modeling tools for specific areas.

Likewise, transformations can be specified to forge links between different meta-models, and thus switch from one modeling language to another. However, much work still needs to be done to show how to capitalize on MDA skills to improve development processes, in particular by reducing costs (for example by facilitating the transition from one activity to another) while allowing have quality products.

We presented the OMG's efforts for model composition in UML and provided an overview of the two hemisphere model driven architecture approach, and possible transformations in the frame of model composition develop an extension of this approach.

CHAPTER III. TOWARDS AN ENGINEERING OF SPECIFIC SOFTWARE PROCESSES DRIVEN BY MODELS

III.1. Introduction

In Software Engineering as in all other engineering fields, the product to be build is divided into sub-parts that are independently constructed and subsequently assembled. This procedure reduces the complexity and improves the reuse of the developed products.

MDA (Model Driven Engineering) is a recent software engineering discipline that focuses this approach. It has allowed several significant improvements in the development of complex software systems by providing the means that enable to switch from one abstraction level into another or from one modeling space into another.

However, models management may be tedious and costly. Thus, it is necessary to provide some flexible and reliable tools for automatic management of models and some techniques for their transformations in order to live up to user expectations.

In this context, model composition has become an important artifact in the MDA domain that allows constructing and composing an efficient assembly process. In this chapter, we present the state of the art of recent works in the model composition methods and techniques by focusing on the various parameters that govern and characterize their behavior.

The main contribution of this part is to classify ,analyze and compare existing composition model by presenting the composition approaches that allow reuse which imposes that the assembly process can be performed without having to modify the parts concerned or their production or execution environment. In our context, this means that composition model must be possible without modifying the concerned models and their specific domain of development.

III.2. Multi Modeling paradigm

The modeling of software systems is an area of research in which the approaches based on the separation of concerns have much interest. Indeed, the current requirements modeling systems become more complex, the construction of a comprehensive model taking in consideration all needs simultaneously is extremely difficult, even impossible in some cases.

Users of a system (for example: customers, staff, administrators) are often multiple with special needs (profiles) and specific expectations for a system.

III.2.1.Complexity

When discussing software and systems engineering, it is only a matter of time before the topic of managing complexity arises. The desire to manage complexity was the driving force behind the emergence of the aforementioned disciplines, and despite many valiant attempts to master it; the problem is still with us today. However, we believe that the nature of actual systems is quite different to those developed when those disciplines emerged, and in turn the developers of today's systems face different challenges to those in previous decades. In particular, it is no longer sufficient to manage complexity alone. Instead, we believe that most of today's development challenges boil down to a combination of three important factors: complexity, diversity and change.

III.2.2.Challenge of Diversity

The challenge of diversity reflects how developers have to manage in a non-homogenous environment. Life would be much easier if there was only one programming language and one deployment platform, but of course this is not the case, and for very good reasons. Diversity is not really a single challenge, but a category of challenges, outlined below. .

With this increased complexity of software systems, it is required to manage heterogeneous components, each addressing many at once. An obvious consequence of this complexity is the loss of modularity in the developed systems, which hampers their development and reuse. This problem is characterized by symptoms known under the names of dispersion (scattering) and coupling features (tangling) (identified for the first time by [Kiczales, 97] and used as justification for the programming approach aspects).

• Scattering: this is the case where a concern (functional or transversal) is distributed throughout the system, and not placed in a clearly identified unit. For example, if a feature is distributed over several components, the cost of an upgrade of this feature can be considerable.

• Tangling: this is the case where a unit contains several elements from different concerns. In this case, we have in the same component an inter-connection between multiple features, and therefore the management cost of the various interactions between these features can be important

To remedy the situation, several approaches adopting the principle of separation of concerns have been proposed. They allow a decomposition of modeling, especially in the design phase where several models can be developed separately to represent a particular perspective of the system. The developed partial models must then be compounds to produce the final model of the system.

III.2.3.Decomposition meaning

The idea of decomposition methodology is to break down a complex task into several smaller, less complex and more manageable, sub-tasks that are solvable by using existing tools, then joining their solutions together in order to solve the original problem. 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.

Managing and pruning complexity application development are recurrent problems in any scientific reasoning. The activities of decomposing problems into more manageable sub-problems and propose abstract representations to hide unnecessary details are the keys to properly understand the situation and to successfully provide solutions.

The degree to which these sub-problems may be separated and recombined is the principle of modularity. That why one of the explicit challenges in model driven engineering is to develop methods that will be feasible for complicated real-world problems. In many disciplines, when a problem becomes more complex, there is a natural tendency to try to break it down into smaller, distinct but connected pieces. The concept of breaking down a system into smaller pieces is generally referred to as decomposition. The purpose of decomposition methodology is to break down a complex problem into smaller, less complex and more manageable, sub-problems that are solvable by using existing tools, then joining them together to solve the initial problem. Decomposition methodology can be considered as an effective strategy for changing the representation of a classification problem. Indeed, considers decomposition as the "most useful form of transformation of data sets".

The activity of decomposing problems needs a step of composition 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. That why composition Model is a challenging topic of interest in which the definition of new approaches should benefit from existing composition model techniques.

When should we prefer one decomposition method over the other? Is it possible to solve a given problem using a hybridization of several decomposition methods?

III.3. Multi-Model Approaches

When tackling the complexity of large software systems, separation of concerns is essential for keeping the development process, the produced models and the code manageable. The separation of concerns can be done in different ways, but the objectives are always the same: being able to identify relatively independent "parts", so that they can be distributed among different actors of the process, be designed and built independently and, at the end, be integrated with the least possible effort and in a way which allows for future maintenance and evolution. Despite the evolution of analytical / design techniques in the field of software engineering, the construction of computer systems remains a difficult task. In reality, several partial models are developed separately and coexist with the associated risks of inconsistency, ie the global model must be frequently challenged when needs change. The object approach and related concepts (encapsulation, inheritance, and polymorphism) have been an important advance for the design of software systems, including modularity and reuse. But this approach has limitations when it comes to dealing with complex systems, multidimensional (i.e. multilevel systems: functional, business, technological, etc.) and highly parallel. To control this complexity, we use more and more so-called multi-model modeling approaches. Indeed, multi-modeling allows a decomposition of facet modeling, especially in the design phase where several models can be developed separately to represent a particular perspective of the system. The partial models developed are then compounded (woven) to produce the final model of the system. Multi-modeling can be designed in different ways: subject programming, role programming, aspect programming and point programming from view]. In this section, we review these approaches.

In this section, we present the four major multi-modeling approaches: Views modeling, Aspects Modeling, Subject Modeling and Role modeling.

III.3.1. Views Modeling

A point of view is defined in the ROBERT dictionary by: "a place where one must place oneself to see an object as best as possible or as a particular way a question can be considered". The terms closest to the definition of the point of view are: appearance, optics, perspective and view. In computer science, this notion of point of view has a multitude of meanings that diverge according to the work and the domains. The concepts of view and point of view have been studied in several areas related to information processing: databases, knowledge representation, analysis and design, programming languages, software engineering tools, etc. In the field of databases, the notion of view is exploited by the interrogation languages as a selection function on the data. In the representation of knowledge, views are used to represent classificatory reasoning and the taxonomic representation of knowledge. A point of view determines a set of characteristics related to a concept or a family of objects. A concept can be observed from different points of view. Our research team has been working for several years on the integration of the notion of point of view in the analysis / design of software systems. The work of Nassar [Nassar, 05] led to the establishment of a UML profile called View Based UML (VUML), which makes it possible to analyze / design a software system through an approach combining objects and points of view.

In the field of databases, the notion of view is operated by the query languages as a selection function on the data. In knowledge representation, the views are used to represent the taxonomic classificatory reasoning and knowledge representation. A view determines a set of characteristics associated with a concept or an object family.

In the views approach, the decomposition level is different from that adopted by the aspects approach. This is decomposition according to the views of actors of the system. The views are developed independently of each other and without making any distinction between the basic functionality and cross-functionality. The result of this type of decomposition is a set of entities described by the subjective views of the actors of the system.

III.3.2. Aspects Modeling

The modeling aspects (Aspect-Oriented Modeling and AOM) is an approach of multi modeling based on the separation between functional concerns and preoccupations called "cross" in the software development. The idea of modeling aspects results from the AOP approach and proposes to consider aspects in models. With the emergence of the IDM, the appearance paradigm was extended to upstream phases of software development, it means the design phase or to the requirements analysis phase.

The aspects approach decomposes the system into functional units and non-functional units. It thus separates the core functionality (or trades) of an application transversal extrinsic features to business requirements.

Aspect-Oriented Modeling (AOM) is a multimodeling approach based on the separation of functional concerns from so-called "cross-cutting" concerns in software development. The idea of

aspect modeling stems from aspect programming and proposes to consider aspects in models [Klein, 06]. Faced with the emergence of the IDM, the aspect paradigm has been extended to the upstream phases of software development, ie to the design phase [Clarke, 01] or to the requirements analysis phase. [Jacobson et al., 04]. The term aspect is associated with features that cannot be efficiently gathered in a single module (for example: security, performance, persistence, etc.). The paradigm of aspect-based modeling considers that the basic services of an application (the functional concerns) and its transversal properties or functionalities must be independent and decoupled from each other as well in the design as in the implementation. For example, Credit Card Abstraction admits a functional concern representing the payment process. It also accepts other (cross-cutting) technical concerns such as authentication management when connecting, the integrity of the payment transaction, and so on. The approach followed for aspect-based modeling consists of decomposing the application into aspects and basic units. A base unit represents a feature - or part of a feature - of a system. An aspect represents a transverse property that affects the basic functionality in a systematic way. These properties can be designed and analyzed separately from the basic functions of the application. It is thus possible to define on a basic model, mixing different cross-cutting concerns, as many aspects as necessary to represent these various concerns.

III.3.3.Subject Modeling

Programming subjects or SOP (Subject Oriented Programming) is another concern of separation technique introduced by [Harrison et al., 93]. This approach is based on a multidimensional separation of concerns, to cover different types of concerns (business, technology, business rules, etc.). It identifies a set of specifications and behaviors that reflect the perception of the real world corresponding to a generic vision of an actor.

The subject approach extended by MDSoC approach (Multidimensional Separation of Concerns), its offers a decomposition of the system into more arbitrary dimensions, where each dimension is a collection of particular concerns. We talk about concern in the broad sense, without differentiating between basic concerns or crosscutting concern.

Subject Oriented Programming (SOP) is another problem separation technique introduced by [Harrison et al., 93]. This approach is based on a multidimensional separation of concerns, covering different types of concerns (business, technology, management rules, etc.). It identifies a set of specifications and behaviors reflecting the perception of the real world corresponding to a generic vision of an actor. For example, a training administrator has his own characteristics for a course: a price and management methods. These features can be applied to any object for sale: product, real estate, etc. These characteristics are then called extrinsic to the object "Course". They are part of the generic subjective view of the administrator. A subject does not correspond to a class but it is a class hierarchy specification. This hierarchy denotes the set of object descriptions for a particular view. This specification does not describe a structure for object instances but it is a schematic description that can be applied to a particular domain of objects [Tarr et al., 99].

It is the instances of a subject, called subject activations that actually contain the data of a system. A subject can be activated for several domains and, for its part, an object can activate several subjects. The link between the different activations is realized through the notion of object identity.

The integration of all subjects is determined by a set of so-called composition rules [Ossher et al., 95]. A composition rule involves two or more topics to be integrated. There are two main categories of composition rules: matching rules and combination rules. The matching rules deal with syntactic and semantic links between classes (respectively class properties) with identical or different names, and belonging to different subjects to be integrated. Combination rules specify how classes (respectively class properties) of subjects should be composed. The definition of the composition rules is encapsulated in one or more composition modules, which involve several subjects to integrate [Ossher et al., 01].

III.3.4.Role modeling

In a complex system, an object interacts with the world subjectively from the perspective of the user's context. The emergence of the concept of role comes from the need that extrinsic properties of an object can change over time. Indeed, an object can be subject to multiple classifications during its life cycle, and each time it plays a role suited to a particular situation. Kristensen defines a role as a temporary viewpoint. Riehle refines this definition of role by associating a type that describes the view has an overlooked subject for another and specifying that an object can play different roles at a given time. In summary, the role concept addresses three main issues that arise when we want to model the dynamic aspects of the entities with traditional object models.

In role approach, it is to represent an entity of the model through multiple objects. Each object models a particular role played by an entity. Unlike modeling by views, roles are objects resulting from local entities subjective views without being linked to actors of the system. The different subjective views of the system represent all of the application concerns, and each concern is represented by the different roles and their interactions.

In a complex system, an object interacts with the world in a subjective way from the perspective of the user's context [Harrison et al., 93]. The appearance of the notion of role comes from the fact that the extrinsic properties of an object can change over time [Pernici, 90]. Indeed, an object can be subject to multiple classifications during its life cycle, and each time it plays a role adapted to a particular situation. Kristensen defines a role as a temporary point of view [Kristensen et al., 96]. Riehle refines this role definition by associating it with a type that describes the view that one object has vis-à-vis another and specifying that an object can play different roles at a given moment [Riehle et al., 98] .

In summary, the role concept addresses three main issues that arise when modeling the dynamic aspect of entities with traditional object models [Dahchour et al., 06].

Dynamic class change: objects change classification. For example, in time, a person ceases to be a student, and becomes a laureate. Two different cases occur depending on whether the object being transitioned is kept as an instance of the source class or not:

- An extension describes the case where the object remains an instance of the source class.
- An evolution describes the case where the object is no longer an instance of the source class.

The multiple instantiation of the same class: an object can be instance more than once of the same class. For example, a student may be enrolled in two different universities. He will therefore play two different roles.

Contextual access: the ability to see a multi-faceted object in a particular perspective. For example, the person can be seen separately as an employee or as a student.

III.4. Composition Concepts

All these definitions are based on the specification of ISO RM-ODP (Reference Model of Open distributed processing).

Composition: In RM-ODP specification, the composition is defined as an operation that creates a new object from a combination of two or more objects. The characteristics of this new object depend on the features of each handset objects and how these objects are combined.

Behavior: in RM-ODP specification, the behavior is defined as a collection of actions with a set of constraints that can occur.

Behavioral composition: in RM-ODP specification [[Ober et al., 08b], [OdO07] the composition of behavior is defined as an operation that creates a new behavior from a combination of two or more behaviors. The characteristics of this new behavior depend on the features of each handset behavior and the way how these behaviors are combined.

Configuration: in RM-ODP specification, the configuration of objects is a collection of objects that can interact with the interfaces. A configuration determines the set of objects involved in each interaction,

A specification of an object configuration can be static or dynamic with mechanisms that can change the configuration. The result of the subject composition is a configuration of objects.

The interactions: the services interact to provide a feature of their composition. Interactions materialize dependencies between services. [BB01]

From the data definitions, we can identify in the context of our work two types of dependency: Structural dependency and Behavioral dependency.

Structural dependency: from a structural point of view, an s1 service depends on s2 service when, in the design of the s1 service, the s1 service needs the functionality provided for the service s2 to operate: it is said that s1 service depends structurally on service s2. The service s1 defines explicitly the dependency to s2 service. A structural dependency materializes by structural type of interaction. [Sztipanovits et al., 97]

Example of structural dependency: as an example, we consider that a manager is composed by a service production reporting of operators and a graphics service, the Manager uses the Reporting Services for reporting the production of operators stored on a persistent medium. The service production reporting of operators uses the display function given by the graphics service to provide the user of the Manager reporting in different graph form. The service production reporting of operators is dependent on graphics service.

Behavioral dependency: from a behavioral point of view, a s1 service depends on the service s2 when the implementation of the service s1 can influence the implementation of the service s2. The s1 service has no structural dependence with s2. s1 does not need the functionality provided by s2 to operate, but it has a behavioral dependency with s2 service. [Sztipanovits et al., 97]

Behavioral dependency is materialized by a type of behavioral interaction.

Example of behavioral dependency: for example the Reporting Manager is linked to the security manager, for the execution of service production reporting of operators and graphic design comes into play in the implementation of authentication: the data used for authentication correspond to the data which are loaded.

The Reporting Manager does not need the functionality of the security manager and the security manager does not need the functionality of Reporting manager, but their performances are dependent on one another.

From the definitions of the composition and definitions of interactions, two types of composition can be identified: [Cer04]

- The structural composition.

- Behavioral composition.

We'll talk about structural composition when the composition mechanisms will focus on the structural dependency, so these mechanisms are used to define structural type interactions. We'll talk about behavioral composition when composition mechanisms will focus on behavioral dependency. These mechanisms therefore used to define interactions.

Example: the mechanisms used to design the Reporting Manager are from the structural composition (production reporting of operator service uses the graphics design service). The mechanisms used to design the configuration of Reporting and security managers are from the behavioral composition.

III.4.1.Context of software development

First, it is important to better situate the general context of any software development. A software system is the result of an engineering process using current technologies to meet specific customer requirements.

Three essential aspects, corresponding globally to three different professions, must be considered:

- The profession, which requires skills related to the application area of the software,

- Software engineering, which requires skills in software design,

- Management of the development process, which requires software engineering skills.

The profession or field of application requires the taking into account of professional knowledge or theoretical knowledge relating to a particular field of application. It determines the classes of applications and can have a significant impact on the specific aspect of a development process (integration of specific activities or tools, design or use of a business framework, etc.).

Software engineering includes all the techniques, technologies, software solutions or even development methods to be used to design the software. It has an impact on the specificity of the development process (choice of an object-oriented approach, or one based on components or agents, conformity to an architectural style, integration of technical frameworks, etc.).

Managing the process imposes a development cycle or a methodological approach that must be followed by the development team.

III.4.2.Coupling between processes and models

In order to be able to take advantage of model-driven engineering and allow a more tooled approach, it is necessary to couple more intimately the development process and the models produced as well as the necessary modeling languages. Starting from a classic breakdown of a process into activities, products and / or resources and considering that the models are products, a first approach consists in classifying the different entities involved in a development according to several categories.

The processes break down into activities. The activities represent the different tasks to be carried out to ensure a coherent development of the software. These activities are supported by common or specific (modeling) languages and provide the team with the modeling concepts necessary for development.

The different languages are integrated using model transformation activities. Finally, the languages are supported by tools, target platforms, frameworks, which will provide the technical and technological means to produce the software. [HBJ+08]

III.5. Composition approaches

We will go through 6 approaches that have had big impacts in Software Engineering to clarify some criteria. These criteria only concern the composition, they allow us to ignore the other less relevant details of the approach. The proposed criteria are the following:

Concept used: The composition is always inherent in the previous phase of decomposition. The decomposition to divide a system of the entities we call bean, component or bundle.

Coupling: This is the property measuring the degree of attachment of two units. Often we speak of strong coupling and weak coupling. In the strong coupling, it is difficult to understand the isolation units; change will force a unit to change all the associated units; reuse of these units is difficult. However, the weak coupling overcomes all these drawbacks. [HBJ+08]

The communication relates generally to the sending and receiving of data, events, or messages.

Customizing modules: that is to say to allow variations of an existing module.

Mechanism of composition: This is how to assemble the components.

III.5.1.Modular approach

This approach is based on the module concept that is defined as a task manager (responsibility assignment). The construction of a system is to set all modules that perform different tasks. A module is characterized by the following features:

A module is associated with a set of interfaces: interfaces expose the components (resources) provided and required by the module. These resources could be global variables or procedures with parameters and without the implementation.

A module has an implementation portion which is a set of sub programs and data structures that are accessible through the interfaces.

A module can be compiled separately: this allows work in parallel and allows easy replacement of a module with another in a system. Generally, a module communicates with another through the procedure calls and access to global variables declared in the interfaces of the other. The idea of the modular approach is the assembly of modules through their interfaces. This work resulted in the interconnection modules languages (MILs). [Par72]

III.5.2.Approach of Architectural Description

The Architectural Description approach is the successor of the modular approach. It focuses on the modeling of the architecture of software in terms of "... abstract system specification Consisting Primarily of functional components Described in terms of Their Behaviors and component-component interfaces and interconnections".

In the approach of Architectural description, architectures are expressed by the architecture description languages (ADL 10). ADL provides a formal notation for specifying architectural bricks. An architectural brick is a conceptual software unit which shows parts of a system regardless of their implementation. [Hayes-Roth, 1994]

The system is constructed by assembling these bricks. This will then enable the design of applications by detaching the details specific to the environment techniques.

The components are connected either by the connector or by the direct connection interface.

In the first case, the component provides and / or requires one or more ports. The connector connects the ports of components. [Hayes-Roth, 1994]

III.5.3.Software engineering component-based

The software engineering based on components (CBSE) is based on the construction of complex systems by integrating prefabricated software components. The principle of this approach is simple: "do not reinvent reuse purpose (the wheel)" [Szy98]. "Components are for composition. This approach is based on the concept of Component-namely that "A component (composition) is an artifact that allows you to group and isolate a graph of objects in the model, defining explicit responsibility and needs with respect to the rest of the application, allowing it to evolve independently.»

This depends on the component technology, for example, CORBA or DCOM RPC use, EJB uses RMI, Web Services uses SOAP RPC-28 etc.

The composition is made by connecting each component when analyzing (declarative), when designing (scripting and programming) and at runtime (visual). [Cer04]

III.5.4.Aspects Oriented Programming

In an object-oriented application, it is common for application features are scattered in different places and do not receive adequate encapsulation at both design models of programming languages. Such functionality is called a crosscutting concern (crosscutting concern) [KLM+][Wam03].

The cross features become scattered in the code, as and as the application evolves, difficult to identify, understand and evolve.

Aspects Oriented Programming aims to solve this problem by proposing to write the program into two parts: a functional part that encapsulates the core business application code, and a secondary part which includes cross-functionality disseminated. [KLM+][Wam03]

Communication is done by the event invoked implicitly calls between the core curriculum and aspects.

In this approach, the basic program code and code aspects are completely separate. A third language is used to establish relationships between them.

III.5.5.Reflexive approaches

Reflexivity is the ability of a system to reason and act on himself in its own execution, so to self-representation and self-edit. []

Formalize this concept. A reflective system is divided into two levels: a base level that corresponds to the functional application and a Meta level (meta-level) corresponding to the non-functional properties. Introspection is the ability of a program to observe its own state and therefore to reason about it. Intercession is the ability of a program to modify its own state of reification execution. Reification is the mechanism that gives the program the ability to act on its execution state. A Meta level can be seen as the interpreter running the baseline. A meta-level is indicated by the meta-objects that implement the functionality of the interpreter. The base level and Meta level are connected by a Meta link symbolizes the relationship between objects of the base level and meta-objects. This link is represented by the definition of a MOP (Meta-Object Protocol) []

There are two types of reification: the static and dynamic reification.

Static reification: the static reification is total and static reification of the base level before running it. Its internal data structures are reified. It is done at compile time (Compile-time) .The work in this field are related to open compilers (or meta-compilers) as OpenJava []. Generally these extension points are related with syntax tree entities of the source language: Assignment, method invocation, objects creation, etc. Meta-objects are extensions (specializations) compiler that specialize it to generate the correct code. These meta-object managing Meta information of the entities of language.

Dynamic reification: the dynamic reification is the dynamic of reification basic level, i.e. the reification of the entities that are involved in the behavior of this level.

It is done at runtime (run-time) .The work in this field are related to the execution environments (interpreters, virtual machine environments reflexive) such as Metaxa, Guarana, etc. The MOP correspond to points of extension of the execution environment provided by the various prototypes, meta-objects represent extensions of this environment and add dynamically (at runtime) behavior at the basic level executed by this environment.

Interaction composition: the interaction composition is based on the mechanisms of Behavioral fusion allows generating the behavior resulting from the interactions composition, the fusion occurs only where the interactions are enabled on a same trigger message and the fusion is a mechanism placed in the work to resolve the non-orthogonal paradigm.

Behavioral fusion is composed from rewriting rules applied on the patterns interactions and based on the semantics operators. For simple illustration, one of the rules is that a fusion may result in parallelization of calling services. The composition in the approach oriented interactions is behavioral. The rewrite rules define the interleaving of the execution interactions. The grain of the scheduling is performed at the level of the instructions defined in the pattern of interaction.

In the implementation of this model, these rules can be executed dynamically. The current prototype is limited in the sense that it does not offer the ability to define and apply (statically / dynamically) new fusion rules of sequential interactions; the result is a competitive runtime behavior interaction. But there is no mechanism to set this fusion; such as execute in sequential the interactions behavior (according to a defined order) and not concurrently.

The composition in the interaction approach is based on a composition of interactions (Called fusion) that allows the composition of the technical services. The interaction software approach differs from other approaches in that the fusion interaction allows interleaving their executions.

In this part, we made a classification of the approaches we have studied through six criteria. The Table 1 summarizes the characteristics of these approaches.

	Modular	Architectural	Software Engineering based On components	AOP	Reflexive	Interaction
Concept	Module	Brick	Component	Aspect	Meta-Object	Model object
Coupling	Low	Low	Low	Low	Low	Low
Communication	calling function and access of global variables	defined by the types of connector and / or links	defined by the types of connector and / or links	Calling event between the basic program and aspects	Meta link symbolizes the relationship between objects of the base level and meta-objects	calling services
Customizing modules	No	No	Yes	Yes	Yes	No
Mechanism of composition	By connection of the modules after the system design.	By connection of the components during the analysis of the system.	by connection of the components in the analysis (declarative), when designing (scripting language and programming), and at runtime (visual)	Weaving	static and dynamic reification	Fusion
Type of composition	Structural	Structural	Structural	Behavioral	-	Behavioral

Table 1 behavioral and structural composition approaches reviews

III.6. Model Composition Approches

Model composition approaches are widely applied in different domains and contexts, it plays a central role in several open issues in real world applications such as: model comparison, schema integration, similarity between semantic business process models, model transformation testing, migration, matching class diagram, and merging of architectural views. However many generic model composition algorithms and tools have been proposed so far in different application domains. In order to cover the works that are close to ours, we conducted a research to highlight the state of the art of existing model composition methods and techniques. Thus, previous research works have proposed many techniques to tackle the inherent problems related to matching, and achieved an automation degree in matching operation for specific application domains. With this in mind, an extensive investigation on related works is necessary before an own innovative approach is developed. They show many facets of model composition and therefore are capable of covering different application and user views. In what follows we make a comparative analysis of related work. First we give an overview of the approaches relevant to our work in adding flexibility to the model comparison process. Table 1 summarizes our finding. An ideal real world model comparison application would be a combination of the strengths of each approach, rather than one in particular.

III.6.1.EML (Epsilon Merging Language)

EML [KPP06a] is a rule-based language based on the Epsilon platform (Extensible Platform for Integrated Specification of Languages for model management) [Epsilon, 06], MIC [Sztipanovits et al., 97]. It allows the composition of models according to different metadata models. Epsilon is a basic platform: it is a core on which it is possible to define models management languages, focused on specific tasks (task-specific language) such as validation, transformation, generation, comparing and merging models.

The type of composition is fusion. EML is based on rules. An EML specification is a set of rules describing how to compose models. There are three types of rule in EML: compare, merge, and transform (predefined abstractions). A comparison rule declares a name and two parameters typed meta-classes of the instances to compare. The body of the rule is divided into two parts: a comparison part (compare), and a part to check the conformity (conform). These two parts contain the criteria that determine whether the corresponding instances are compliant. At runtime, the comparison rules are applied on all instances of two models, the comparison part is applied before the compliance part. A merge rule declares a name and two parameters representing the instances to merge.

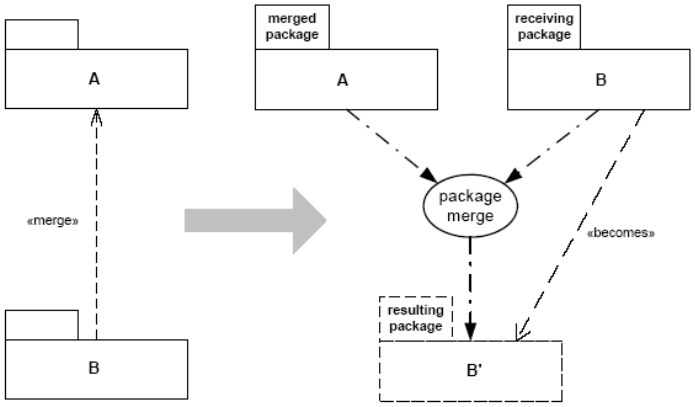
This Code shows an example rule in EML illustrating the comparison, merge and transformation on two classes (Left! Class and Right! Class). The comparison rule returns true if and only if the two classes are either concrete or abstract at the same time and their names and namespaces are the same. The merge rule creates, in the result composite model, a class whose name and namespace come from the left class, the properties are the union of the properties of two input classes. The transformation rule creates the destination object whose name, namespace, and properties come from the source object. (See Figure 16)

```
//The comparison rule returns true if and only if the two classes
//are either concrete or abstract at the same time and their names
rule MatchClasses()
    match l: Left!Class
    with r: Right!Class
    compare
        return l.name = r.name and l.namespace.matches(r.namespace);
    conform
    return l.isAbstract = r.isAbstract;
//The merge rule creates, in the result composite model, a class whose
//name and namespace come from the left class
rule MergeClasses()
    merge l: Left!Class
    with r: Right!Class
    into m: Merged!Class
        m.name := l.name;
        m.namespace := l.namespace.equivalent();
        m.feature := l.feature.includeAll(r.feature).equivalent();
//The transformation rule creates the destination object whose name,
// namespace, and properties come from the source object.
rule ClassToClass()
    transform source:Left!Class
    to target: Right!Class
    target.name := source.name;
    target.namespace := source.namespace.equivalent();
    target.feature := source.feature.equivalent();
```

Figure 16 Example rule in EML [KPP06a]

III.6.2.UML2/Package Merge

A UML2 Package Merge [Zito et al., 06] is presented as a composition mechanism of the contents of two packages with the notion of "Package Merge". UML 2 defines package fusion led as a relation between two packages: a package receiver (PR) and a package merged (PM). The aim of the package is the extension of the fusion functionality, "the possibility of extending the concepts defined in a package with the functionality of another". The PackageMerge [Zito et al., 06] relationship differs from the importing way and creating relationships between classes of the same name. For example, UML can define the concept of include relation at a generic level, and specialize it for different contexts of use while retaining its name. Conceptually, the effect of the PackageMerge relationship can be thought of as an operation which takes the contents of two packages and produces a new package combining the contents of the two packages. (See Figure 17)



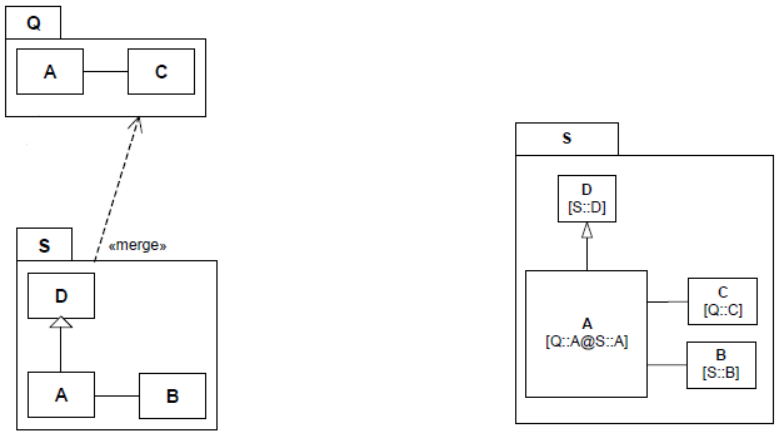


Figure 17 the PackageMerge relationship [Zito et al., 06]

The PackageMerge relationship involves a set of transformations. The source package of the PackageMerge relationship (package B) is called the sink package. This is the one whose content is extended after the merge operation. The destination package of the relation (package A) is called a merged package. This is the package that contains the additional items added to the receiving package. Figure 14 gives an illustration of this principle.

In the UML2 specification [OMG, 03a], the semantics of the PackageMerge relation is defined by a set of rules of transformations and constraints. Transformation rules ensure the matching of elements and the merging of the corresponding elements. These rules are defined for each type of metaclass. The constraints make it possible to ensure the validity of the merger. As with transformation rules, constraints are specified for each type of metaclass.

III.6.3.Kompose

Kompose is an open source tool, implemented in Kermeta [DFF+09], which supports the model of composition. It sets out an approach designed to oriented modeling aspects. The crosscutting concerns are defined in a model aspect AM (Aspect Model) while the basic features are the primary model in a PM (Primary Model).

Compound models are in Ecore. The type of composition is fusion. Note that Kompose historically started with the aspect-oriented modeling perspective, so he distinguished between primary models and aspect models. However, as its authors say in [FBFG07], this distinction has no special meaning except to have a good conceptual view of the separation of concerns. In practice, the handling supports on the two types are the same.

Kompose has a model composition language for describing composition specifications called composition directives. The central concept of this language is the Composer. A composer represents a composition operation performed on two metamodels. It is itself the root object that gathers all the composition directives. The structure of a composer includes: a primary input metamodel, an input aspect metamodel, the name of the output composite metamodel and all the composition directives.

Kompose provides two types of composition directives: pre-merge directive and pos-merge directive. Pre-merge directives are applied on the metamodels before composing them, post premege directives are applied on the composite metamodel before producing it. The former specify simple

modifications to the input models such as renaming, deleting, or adding elements in order to force or prevent merging.

The second reconciles the merged model so that it is reliable and consistent. Five types of actions can be defined in a directive to designate what it will do: Add, Remove, Create, and Set.

The Kompose composition language has a concrete textual syntax. An example of this syntax is in Figure 15. Its abstract syntax is in Ecore. You can "program" composition programs either in textual syntax using a Kompose textual editor, or directly in abstract syntax using the EMF editor. (See Figure 18)

```
    // Primary Model URI
PM "platform:/resource/KomposeSamples/bank/Bank.ecore"
// Aspect Model URI
AM "platform:/resource/KomposeSamples/bank/BLP.ecore"
// Composed Model URI
CM "platform:/resource/KomposeSamples/bank/BankBLP.ecore"
// predirectives for primary model
PMPre
// predirectives for aspect model
AMPre
// Rename package BLP to Bank
BLP.name = "Bank"
// postdirectives
Post
Bank::Controller.eOperations - Bank::Controller::transfer
Bank::Controller.eOperations - Bank::Controller::withdraw
```

Figure 18 The Kompose composition language [DFF+09]

III.6.4.AMW (Atlas Model Weaver)

AMW is an AMMA module (ATLAS Model Management Architecture) [Bézivin et al., 05b] - a generic model management platform. This module is targeted towards creating relationships between model elements or metamodel elements.

Compound models are structural, object oriented. AMW now uses EMF [Budinsky et al., 03] as the handler for its models. The type of composition is per operator.

Weaving Patterns and HOT Transformations use a language, called weaving language (formalism), which has a core part providing the basic generic concepts for creating structural links between patterns. These links are saved in weaving. AMW's basic weaving metamodel contains the basic weaving concepts. WElement is the basic element of all elements of the weaving metamodel; WModel represents the root of the weaving model. WLink represents links between the elements of the models. WLinkEnd indicates the type of elements that can be composed.

The links created by core concepts have no semantics. For the composition task, AMW provides the ability to extend the basic weaving metamodel in order to build new weaving concepts specific to the model composition domain (i.e. merge, replace, union, overload etc.) (Formalism extensibility). It is the WLink and WLinkEnd concepts that can be extended to add the new types of composition relation (for example merge, replacement etc.) and also to define the types of elements that the new relations can connect (for example to the instead of linking EObjects of EClass, it may be linking the

objects of the Person and Student user classes). This extension allows to have a weaving language dedicated to the composition of models. Similarly, by extending the basic weaving metamodel, one can also create a family of weaving metamodels for tracing the evolution, comparison, translation and even transformation of patterns.

III.6.5.EMF (Eclipse Modeling Framework)

Virtual EMF [Budinsky et al., 03] is an Eclipse plug-in built on top of EMF. A virtual model is a model that does not contain physical data: it is defined as a template that redirects all access and manipulation requests directly to all the basic models from which it was generated [Ecl06].

Compound models are in Ecore. The type of composition is by relations.

References (compositional elements) are created by EMF [Budinsky et al., 03] [Ecl06] editors generated from metamodels. These metamodels are defined by an extended metamodel of the MOF 5 standard, called Ecore (formalism). A metamodel (i.e. a model in Ecore) is described by EClass objects. These objects can have relationships (eReferences) and attributes (eAttributes).

It is possible to create inter-metamodel references, i.e. the references link two concepts from two different metamodels. These references are defined by the EReference concept in the Ecore metamodel. The instantiation of references at the level of metamodels makes it possible to create the links between the models (composition by relations). The models and the inter-model links together make the composite model.

Conceptually, an inter-metamodel reference resembles other references defined in the same metamodel. That is, it is also defined by the EReference type of the Ecore metamodel. It can be unidirectional or bidirectional, with cardinality 1 or multiple. It can also be of the type of containment or not. Conceptually, there is no difference between the types of references, whether inter-metamodel or not, except that inter-metamodel references link two concepts from two different metamodels while the others link concepts in the same metamodel.

From a technical point of view, inter-metamodel relationships are represented as proxies. The referenced metamodels will be loaded when requested (lazy loading).

The process of model composition by EMF editors can be summarized as follows:

First, we establish the references between the metamodels. To do this, you have to load the metamodels in an editor of Ecore. EMF [Budinsky et al., 03] manages metamodels as resources.

A resource is a persistent document containing the objects of the model. The Load Resource feature is provided to import metamodels. Once the metamodels are loaded in the same Ecore editor, we can compose them. The composition consists in creating references between the imported metamodels. The editor that composes is that of the composite metamodel.

Once the composite metamodel is obtained, we use EMF to generate the editors from this metamodel. We will generate a separate editor for each sub-metamodel. In these editors, we can build separate conforming models to these different compound metamodels. Then, we generate a third editor for the composite metamodel (in fact, it is the editor for the part of the references established between the metamodels which obviously are not known by the editors of the compound metamodels). Apparently, from an interface point of view, these editors are separate, but they can communicate

because they share and manipulate the same data model (i.e. the composite metamodel). In the composite editor just generated, using the same resource loading, we can reimport the models already created in different editors and instantiate the references between them. The result is a composite model.

III.6.6.AML (AtlanMod Matching Language)

AML AtlanMod Matching Language [GJCB09] is an extension of AMW for obtaining a matching pattern. It assembles different correspondence implementation strategies that are implemented as a series of model transformations. Each of these transformations takes a set of input and produces a correspondence output models.

The definition of an AML program is done in 3 steps: Import section: Allows you to reuse existing AML algorithms, Declaration of correspondence rules, Pattern Flow Block: Specifies how multiple matching techniques interact to produce the different match links.

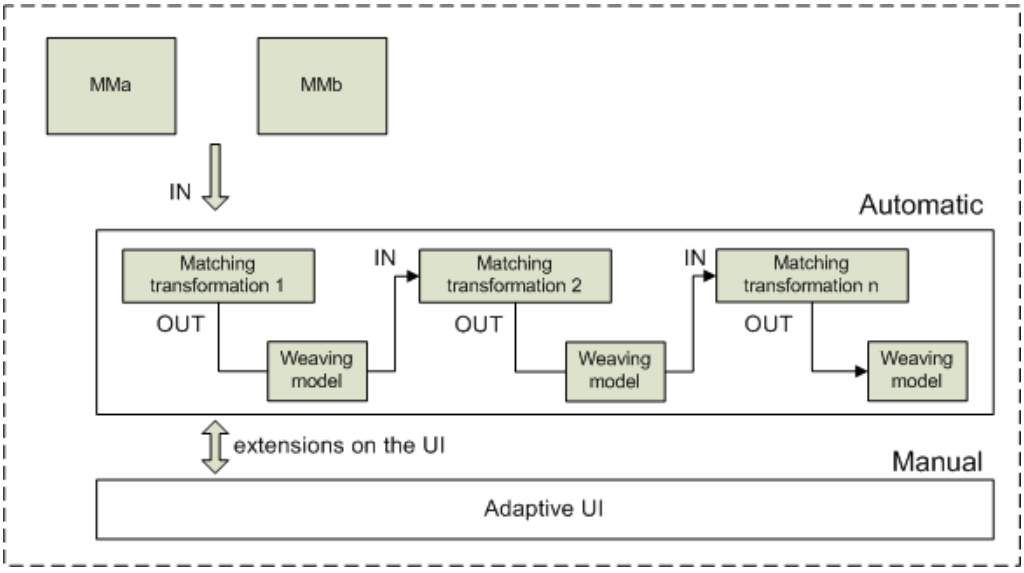


Figure 19 Correspondence implementation in AML approach [GJCB09]

As shown in Figure 19, the transformations implement different heuristics to produce the correspondence model. Initially, the transformation takes the two meta-models as input, and creates a link model. Subsequently, a transformation mechanism will add to the source models the model of links produced in the previous step in order to refine the correspondence model and to define more precise links.

III.6.7.MatchBox

MatchBox is a Framework that combines several mapping strategies in order to get the most refined correspondence links and subtle. The matching process (see Figure 20) consists of 3 steps [VIR10]

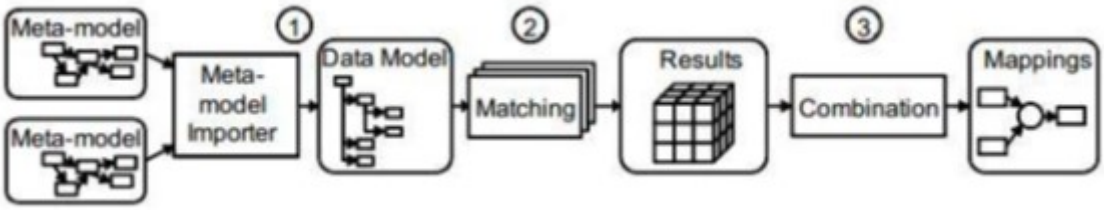


Figure 20 the matching process of Matchbox [GJCB09]

Step 1: Transformation of the input meta-models to the AMC (SAP Auto Mapping Core) data model, an internal model of a tree structure whose implementation is inspired by COMA ++ [Do06]. The reason for this transformation is to be able to benefit from different correspondence approaches based on existing diagrams and to exploit them on the AMC models.

Step 2: The MatchBox system allows the application of several matching algorithms (Matcher). It can be configured to allow the developer to choose the type of matching algorithm to use and to produce a matrix for each that contains similarity values between all of the model elements. The different matrices obtained are grouped together in the form of a cube (The X, Y and Z coordinates represent the source, target and type of correspondence respectively).

Step 3: The purpose of this step is to reduce the cube resulting from the previous step to a matrix. This matrix is obtained by first defining a similarity threshold and then filtering the inputs of the cube to keep only the values above the threshold.

III.6.8.ECL (Epsilon Comparison Language)

ECL [KPP06b] is a rule-based language for building links, based on the Epsilon platform [Epsilon, 06], MIC [Sztipanovits et al., 97]. Epsilon is a platform on which it is possible to define models for managing languages, focused on specific tasks (task-specific language), such as validation, transformation, generation, comparing and merging models.

An ECL rule [KPP06b] takes as input two parameters that refer to the model elements to compare. It is executed on all the pairs of meta-class instances which check these parameters. The body of an ECL rule is composed of three parts (highlighted elements in Figure 21): a comparison (compare), a compliance (conform) and an optional third part "guard".

The comparison determines whether two instances match based on a set of criteria. Compliance is a refinement of the comparison in the sense that it is performed only on items that have met the criteria defined in the comparison. Thus, for example, we can check whether two model elements have the same name. If this comparison returns a true value, we can check the conformity by comparing additional properties (type, cardinality…).

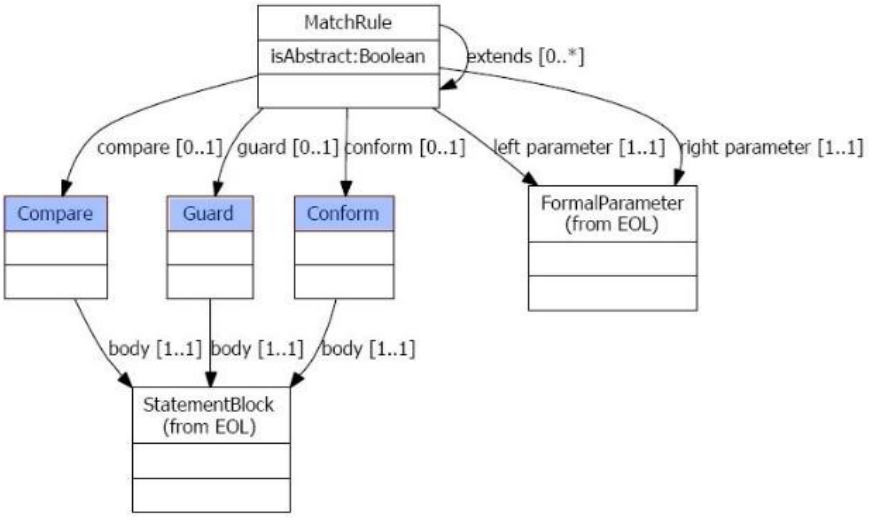


Figure 21 the body of an ECL rule

Regarding the optional "guard" part, we can limit the application of the comparison and compliance rules to a subset of defined elements. Thus the elements which are not declared in this part will not be affected by the execution of these rules.

The result of the execution of the ECL rules provides pairs of instances, saved in a "matching trace".

III.6.9. EMFCompare

EMFCompare is an approach applicable only in the field of meta-models. EMFCompare [BP08] is an Eclipse project that performs the EMF model mapping. This approach calculates the correspondence based on the principle of similarity. The matching engine based on heuristics and elements are compared following several metrics including: name similarity, type of content and relation. They return values ranging from 0 to 1, which will be aggregated to obtain overall similarity values.

III.6.10. XMF (eXecutable Metamodeling Facility)

XMF [CESW04] is a meta modeling framework invented by Xactium company that allows to easily design DSMLs(Domain Specific Modeling Language) executable. XMF manage synchronization and communication between two models. The vision of transformation is more related to the MDA approach. Compound models are created in languages which are modeled by the XMF meta-modeling language (remember that XMF is a meta-modeling approach).

- The XMF metamodeling language in fact consists of several sub-languages: [CESW04]
- XCore and XOCL provide the abstract syntax and semantics of the language.
- XEBNF provides the concrete textual syntax of the language.

- An own graphical language provides the concrete diagrammatic syntax of the language.

```
'context Root
@Operation sameName(c1,c2)
@XSync
@Scope
Setc1,c2
end
@Rule r1 1
x1 = Class[name = n1] when x1 = c1;
x2 = Class[name = n2] when x2 = c2 and n1 <> n2
do x1.name := x2.name
end
end
end'
```

Figure 22 Rule implementation of a XMF approach [CESW04]

XCore and XOCL XCore is the heart of XMF, it is a kernel language providing all the concepts and object-oriented meta-modeling primitives such as packages, classes, associations, objects necessary to design the structure of a language. XCore is an extension of MOF. However, unlike MOF, XCore is executable. The executability of XCore comes from two factors. On the one hand, XCore uses an extensible OCL constraint language, called XOCL, which allows to describe the behavioral semantics of the language; on the other hand XCore explicitly defines an Operation concept which is the means allowing to manipulate the state of the models; this is an advantage over MOF 17. XCore and XOCL allow to model the abstract syntax and the semantics of the XMF language. We add that when defining the abstract syntax, XOCL is also used to describe well-formed rules so that the language can eliminate invalid models.

XEBNF XMF uses an extension of the EBNF language, called XEBNF - a generic parser language, to define the concrete textual syntax of the language.

Note that based on standards such as QVT, OCL, MOF, XMF languages are independent of technical platforms, even the implementation of the language behavior is carried out in a platform independent language (i.e. XOCL). This gives great exibility to XMF. Exibility and executability are the main advantages of XMF.

Models composed in XMF are UML class diagrams. The type of composition is by relationships called mappings.

A mapping is a relationship or transformation between models or programs written in the same or different languages.

In this definition, we can see that there are two visions for mapping: a specific transformation vision of MDA; the other view is more generic. Generic in the sense that a mapping can be used for several different purposes, not just transformation.

For example, we can maintain consistency between two models, manage the event and data exchange, manage synchronization and communication between two models. Rather, the transformation vision is linked to the MDA approach. On the other hand, we are interested in the composition of models, so the generic vision fits well in our perspective.

Corresponding to the two visions above, XMF offers two types of mappings: unidirectional mapping and synchronized mapping. One-way mapping: it is based on the transformation vision.

Unidirectional mapping takes one (or a set of) model (s) as input and then generates a model as output. In order to describe this mapping, XMF provides XMap. XMap is a declarative, executable, pattern-matching, one-way mapping language similar to transformation languages like ATL 18 [JK05]. One-way mappings are often used in code generation, for example to transform a model to Java or C ++. From a composition perspective, we are not interested in this type of mapping.

Synchronized mapping: This is a mapping intended to manage the synchronization between two models. As we said above, synchronized mappings can be used for several purposes; consistency management is only one of several applications of this type of mapping. Other typical applications of synchronized mapping are the management of multiple models of a system, support for "round trip engineering" etc.

In order to describe synchronized mappings, XMF provides XSync. We take a simple example in [CESW04] to illustrate the nature of synchronized mapping and the syntax of XSync.

A synchronized mapping consists of a scope and a collection of synchronization rules. The mapping scope is a collection of items to which the mapping is applied. In this example, these are instances c1, c2. A rule describes the condition under which an action is performed. Actions are synchronizations that can be performed on both sides. A rule of the form: a pattern, a boolean when condition and a do action. This example is a synchronized mapping which takes any two classes and synchronizes the name of the first class with that of the second.

The first note on the example above is that XSync allows you to define operations for mappings. This means that in the case of XMF, the models are structurally and behaviorally composed.

III.6.11. Theme

Theme [Clarke, 02], [Baniassad et al., 04], aims to solve the problem of the dispersion of the requirements in the design patterns. A Theme is a design element for the encapsulation of a feature or a crosscutting concern. It is represented by a package stereotyped by "theme". It can be combined with other Themes using a surrogate relationship called 'bind' which expresses the composition between two Themes.

Theme aims to solve the problem of requirements dispersion in design models.

A Theme corresponds to a design element allowing the encapsulation of a functionality or a transversal concern. It is represented by a package stereotyped by "theme". It can be combined with other Themes using a substitution relation called 'bind' which expresses the composition between two Themes. A Theme is represented by a parameterized package. Each parameter corresponds to a class and to all the methods on which the functionality must be woven. The structural part of the system is often represented in Theme \ UML by class diagrams, while sequence diagrams are used to describe behavior.

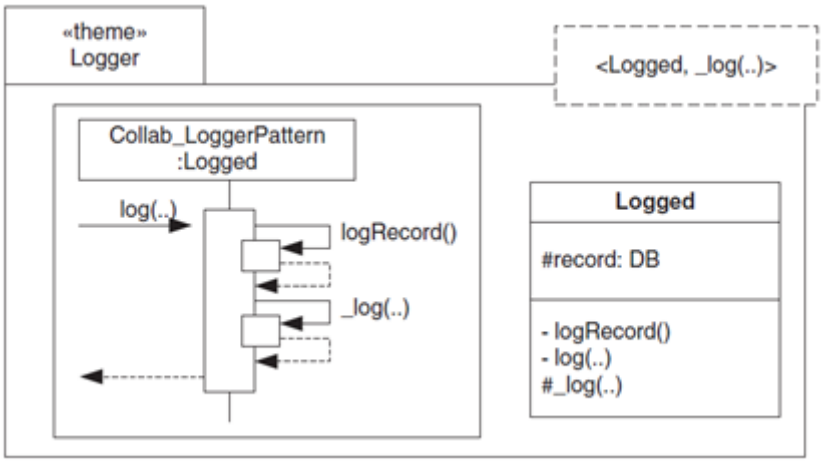


Figure 23 Theme Logger of Theme/UML approach [Clarke, 02], [Baniassad et al., 04],

Figure 23 represents the Theme Logger, a classic example of a transversal concern. Since all the operations of the system must be traced, all the operations of the Themes must be supplemented by an authentication behavior (logging). This aspect has been defined as a generic functionality to be able to design the logging behavior separately from the operations which require an authentication step. The Logger aspect has two template parameters: the Logged class and the _log method. The weaving of the basic CMS (Course Management System) Theme with the Theme Logger uses a 'bind' composition relationship. This relation makes it possible to substitute the Logged parameter by the Person, Student and Professor classes defined in CMS, and to substitute the _log method with their respective register, unregister and giveMark methods.

III.6.12. GME (Generic Modeling Environment)

GME [XS05] is a generic modeling environment that using modeling paradigms (modeling languages dedicated to DSML areas). A modeling paradigm formalizes a metamodel by defining the syntax, semantics and concrete presentation of DSML.

The specific modeling environments configured are then used to create the models. These models are saved in the database and then are used to automatically generate or synthesize the applications.

All models defined by GME, whatever the paradigm, share the same set of generic modeling concepts, independent of paradigms and shared by all configured GME environments. These concepts are called FCOs (First Class Objects) (formalism). [XS05]

They define models from a generic point of view so that a model is a set of composed parts without specifying the nature of the part. This nature will be specified by the business concepts of the paradigm. The concepts of FCOs are able to compose / decompose parts allowing to create and manage complex models.

GME is a metamodeling approach, it provides the techniques of composition of metamodels and also those of models.

Metamodels composition: GME offers three types of relationships for the composition of metamodels: equivalence, implementation inheritance and interface inheritance [LNK + 01].

Equivalence: This operator concerns the operation "union" of two concepts defined in two paradigms.

This union produces a unique new concept which includes all the attributes and associations of the two unified concepts.

Implementation Inheritance and Interface Inheritance: These two operators use typical UML class specification mechanisms. In implementation inheritance, the sub-concept can inherit all of the attributes of the super-concept and all of the containment relationships in which the super-concept acts as a container.

Interface inheritance, on the other hand, does not allow relationships to be inherited.

These operators are useful in the case where one wants to extend a metamodel (thanks to the mechanisms of class specification) or to produce a third metamodel unifying the two compound metamodels.

Model Composition: GME distinguishes two cases: the composition of models of the same metamodel and the composition of models of different metamodels.

The model composition of the same metamodel can be done by the reference concept of FCOs. On the other hand, GME is not able to compose the models created by different metamodels although it provides the ability to compose the metamodels. You cannot import existing models and reuse them by establishing relationships between them. Once you have a new composite paradigm, it is only possible to create new models.

III.6.13. MATA (Modeling Aspects using a Transformation Approach)

MATA [Whittle et al., 07a] [Whittle et al., 07b] is an approach for models aspect composition using the model transformation techniques. The composition procedure is asymmetric, because it distinguishes between a basic model and an aspect model. MATA defines an aspect model as a combination of two dependent parts: a pattern and a specification composition. The pattern is used to detect a location in the base model where compositional specifications will be applied. MATA [Whittle et al., 07a] [Whittle et al., 07b] defines three types of annotations represented by create, delete, and context stereotypes. The create stereotype is used to annotate the elements that will be added to the base model, while the elements marked with the delete stereotype will be deleted from the base model. The context stereotype is used to avoid applying a stereotype to several elements in the case where an element is annotated by one of these stereotypes and contains other elements. The composition process with MATA is done in two stages: first a pattern described by the pattern of the appearance is sought in the basic model, then one proceeds to the modification of this pattern according to the composition specification.

The MATA approach was initially developed in the context of aspect-oriented modeling [Kiczales et al., 97]. The composition technique can be generalized to the composition of several models, by considering a chain of transformation by successive application of appearance models. Although the approach only supports a priori the composition of UML class diagrams, sequence diagrams and state diagrams, it can be adapted to other UML models or other modeling languages described by a metamodel.

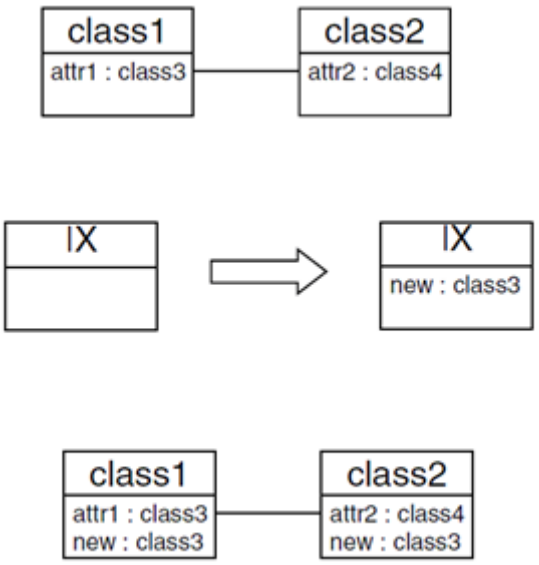


Figure 24 transformation rule in MATA aaproach [Whittle et al., 07a] [Whittle et al., 07b]

Figure 24 describes the transformation rule and presents the result of applying this rule on the initial model. The left part of the rule describes the pattern on which it is applied; here, the pattern corresponds to any UML class. The right part of the rule describes the elements that must be added or removed; in this case, the new attribute is added.

The peculiarity of this approach is that the graph transformation rules are defined using the concrete syntax of the modeling language. This property distinguishes this approach from the most well-known transformation approaches, by the fact that these approaches define the transformation at the level of the meta-model, using the abstract syntax of the modeling language. This can be a definite advantage for the designer familiar with aspect modeling [Kiczales et al., 97] who does not have a deep knowledge of the UML meta-model, when he has to describe transformations based on the abstract syntax of the modeling language.

III.6.14. Muller et al

Muller et al [Muller et al., 05] is an approach based on the component model concept to design a system by assembling a set of parameterized models designed around a model called basic model. This approach is part of the multi-model engineering. Methodologically, the approach proposed structuring the system functionality, this functional decomposition aims to control the complexity of a system and better manage its evolution.

Composition guidelines can be classified according to the level of granularity of the elements to which they apply. Thus, a distinction is made between element directives and template directives. Element directives are used to rename, create, delete, and add elements to a model. Pattern directives are used to determine the order in which the appearance models are composed with the base model, thereby defining a weave order relationship between the appearance models.

Behavioral properties are specified in terms of operations defined in meta-classes and by textual descriptions associated with those operations. This meta-model describes extensions made to the UML meta-model to support merging models using signatures and merge rules and also to support the use

of composition directives. This meta-model has been implemented to provide automated support for the composition of class models using the Kermeta language [Muller et al., 05].

III.7. Review process and potential proposal for a model composition prototype

The work presented in last section (III.6), discuss some similarities between model composition methods and techniques.

Approaches	Reusability	Conflict	Automation	Traceability	Synchronization	preservation	Conviviality	Composability	MDA
EML	+	+	+ +	+	-	+	+ + +	+	++
UML2/Package Merge	+ + +	+	+	-	-	-	-	-	+
Kompose	+	+ +	+ +	+	+ +	+	+ + +	+	+
AMW	+ + +	+ +	+ +	+ +	+	+ + +	+ + +	+	++
VEMF	+ +	+ +	+ +	+ + +	+ +	+ + +	+ + +	+	+
AML	+ + +	+ +	+ +	+ +	+	+ + +	+ +	+	+
MatchBox	+ +	-	+ +	+ +	-	+	+ +	-	+
ECL	-	-	-	-	-	+ +	+ +	-	+
EMFCompare	-	-	+ +	-	-	+ +	+ +	-	+
EMF Editor	+ +	-	+ +	+ +	+ +	+	+	+	+
XMF	+ +	-	+ + +	+	-	+ + +	+ + +	-	+
THEM/UML	+ + +	-	+	-	-	+ + +	+	-	+
GME	+ +	-	+ +	+	+ +	-	-	+	+
France et al	+ +	+ +	+ +	+	-	-	-	+	+
MATA	+	-	+	-	-	+ +	+ +	+ + +	+
Muller et al	+ +	-	+ +	+	+	+ +	+ +	+	+
+ + + : Strongly; + + : Averagely; + : Weakly; - : Non supported by literature									

Table 1 comparison of related approaches

However, all these approaches lack a repository to manage conflicts and analyze how the desired model compositions should be. Without a directive composition, we cannot be sure that the merge result correspond to what we need exactly.

III.7.1. Assessment Criteria

The approaches mentioned above are evaluated according to several assessment criteria, which are selected based on the need to promote the reusability and the automation of model elements, as well as building a generic composition operator. However, the assessment of how a composition approach proceeds to manage conflicts and ensure model consistency during the composition process is also an important coefficient.

The criteria are the following:

Reusability: Aspects that encourage the restructuring and reusing of composition elements and the proposed mechanisms. This criterion is analyzed on two different planes. The foreground is used

to study whether the approach provides ways for defining a generic part / variable in the modeling elements used. This property promotes reuse and structuring systems. This category include approaches based on configurable elements in consideration of appearance of models such as the approaches of France et al [Whittle et al., 07a ; Whittle et al., 07b], Muller et al [Muller et al., 05]., Theme[Clarke, 02], [Baniassad et al., 04], or MATA. The second plan involves the mechanisms offered by an approach to define generic operators' models of composition. In this category we find the AMW approach that offers a weaving generic metamodel that defines the composition of links to a higher level of abstraction, the extension of this metamodel for defining the semantics of links depending on the application domain. The EML [KPP06a] supports generics by defining reusable generic rule libraries, and lets you merge models that conform to different Meta. The composition mechanism MATA [Cottenier et al., 07] is specified in the appearance of the model itself. Therefore, MATA [Cottenier et al., 07] is based on the concrete syntax of the modeling language, which does not promote the reuse of transformations that are explicitly specified in the appearance models.

Conflict management: This criterion aims to ensure consistency of models handled during the composition process. This criterion is obviously essential to consider. Conflict can be managed at several levels or phases may be considered a pre-composition, which is to check consistency between the models to compose. Similarly, post composition is needed to resolve conflicts and inconsistencies that may occur in the compound model. The AAM approach [France et al., 04a ; Reddy et al., 06] intends to use-especially the composition of the elements of directives - to resolve conflicts at both levels; by against the identification of these conflicts remains the responsibility of the designer. The Theme approach manages this aspect by combining a set of reconciliation strategies in a merger type of composition relationship. Several types of reconciliation are offered (precedence, conversion, resizing, etc.). The EML approach does not offer a mechanism to manage conflicts between models; for that, prior harmonization of models is necessary before the phase of "matching". The AMW approach tolerates the presence of conflicts in the models. It offers two resolution strategies: first, automatic, based on the use of heuristics to identify matches; the other, interactive, allows manual refining correspondence links.

Degree of automation: This criterion characterizes the ability of the approach to compose models without human intervention. Our goal is to offer a more automated process possible, it seems important to consider the approaches studied under this criterion. The composition approaches studied are based on assumptions in the definition of an automatic solution composition. These assumptions are several levels or stages of the writing process. For example, the approaches of France et al Muller et al, EML, Theme / UML assume that elements models to merge must have the same signature; in other words, these elements represent consistent views of the same concept, which is not always true in reality. In this case, designers must use common namespace (namespaces) or naming conventions. Without these assumptions, the use of mappings (correspondence relationship) is required. The generation of these mappings can be partially automated using tools like AMW, or automatically by using transformations matching, the latter employing more sophisticated techniques such as thesauri or ontologies for the mapping of the elements.

Traceability: This criterion allows to determine the languages that support the traceability of changes, ie the possibility to keep track of each execution of a transformation rule

Synchronization: This criterion refers to the ability of the approach to spread the model made the changes in the models sources. The choice of this criterion is justified by the need to avoid rerunning the process of composition for each alteration of an input pattern element and impact of these changes directly on the model compound originally produced.

Conviviality: This criterion evaluates the degree of support for approaches by models of composition tools. The measurement of this criterion is done by identifying the mechanisms used: the model serialization, the type of control (rules, syntax, etc.), implementing features (debugging, virtual machine, etc.), or, verification the result of the composition. With the exception of the Theme approach that does not provide implementation of the composition of the proposed mechanisms, most of the approaches presented in this state of the art in bear much. Implementations are available generally in the form of Eclipse plug-ings, which allows to exploit the features of the EMF model transformation framework. These approaches also take advantage of solutions already implemented in the field of model transformation. Thus, in the AMW approach, composition links are translated into a set of executable transformations written in language processing ATL. The Runtime EML operates the EOL pivot language. The technique of composition proposed in AAM approach [France et al., 04a ; Reddy et al., 06] was implemented using the language Kermeta [Muller et al., 05].

Order of composability: The importance of the order in which the models are made depends on the application domain. The composition of the aspects is usually performed in strict order. For cons, the order of composition of business models matter. The study of approaches presented above shows that there are at least two distinct modes of composition. The first mode allows to design the system with dial strings. The approach of Muller et al. falls into this category, and offers a unitary composition operator called apply defined by a set of constraints guaranteeing the same result regardless of the order of compositions. In the second embodiment, the composition is carried out a set of the component models the same basic model. In this perspective, the approach of France et al. Explicitly Studied this problem by Proposing a set of guidelines for phrase providing the order of composition models. Other Approaches composition as MATA allow the user to define an explicit order of composition in the case where an appearance `shall consist before others. If an order is not set, the tool holder chooses the order of composition. This criterion is not taken into account in the Explicitly AMW and EML Approaches

Consideration of MDA concepts: This criterion allows to determine the approaches that exploit the central concepts of the IDM (CIM, PIM, PSM, Model, and Metamodel).

The focus on designing this catalogue of criteria was to provide a fine-grained catalogue of criteria which constitutes the prerequisite for an in-depth evaluation of existing approaches and thus allows as mentioned in the beginning to provide a new refined method in order to uncover other methods gaps. Furthermore, the main goal was to avoid blurred criteria by working out, as far as possible, unambiguous definitions and the criteria's values that are also measurable. Thus, each criterion is described by a set of properties— a name allowing to reference the criteria during evaluation of the approaches. A definition specifying the criterion as unambiguously as possible along with an optional discussion on difficulties in defining the criterion. An appropriate means of measurement, such as a list of possible values or a measurement scale, including not applicable as a default value for each criterion. A coefficient for every criterion in order to calculate how much the studied approaches are close to our work.

	Reusability	Conflict	Automation	Traceability	Synchro	Preservation	Conviviality	Composability	MDA
Coefficient	7	9	7	5	5	3	3	7	9
9 : Strongly; 7 : Averagely; 5 : Weakly; 3 : Non supported by literature									

Table 2 Assigning coefficients for criteria

III.7.2.Process results

The table II present a general classification of different approaches proposed by literature until now.

Classifying, identifying publication fora, and performing thematic analysis of the current model composition methods and techniques are in order to create an extensive and detailed understanding about this area through nine criteria. However, to make a real contribution in the model composition, authors have assigned to each criterion a coefficient (Table 3); more it takes a higher value, more it will be an important factor in designing our conceptual prototype; with this process we can easily select the nearest existing methods to our future approach and determining gaps by graphing and pinpointing for which study types we can involve and implement our method in different ways.

Our conceptual prototype will make use of existing approaches as the glue to unite their fundamental concepts. For this purpose we introduce the notion of reconciliation to calculate how much the proposed methods and techniques are close to our future prototype.

The reconciliation is formally defined as follows:

Reconciliation= Σ (C×CM)

C (Coefficient) =Represents the coefficient of each criterion

CM (Criterion Mark) =Represents the mark attribute to every approach regarding each criterion.

Approaches	Reusability	Conflict	Automation	Traceability	Synch	Models preservation	Conviviality	Composability	MDA Concepts	Reconciliation
EML	1×7=7	1×9=9	2×7=14	1×5=5	0×5=0	1×3=3	3×3=9	1×7=7	2×9=18	72
Package Merge	3×7=21	1×9=9	1×7=7	0×5=0	0×5=0	0×3=0	0×3=0	0×7=0	1×9=9	46
Kompose	1×7=1	2×9=18	2×7=14	1×5=5	2×5=10	1×3=3	3×3=9	1×7=7	1×9=9	76
AMW	3×7=21	2×9=18	2×7=14	2×5=10	1×5=5	3×3=9	3×3=9	1×7=7	1×9=9	102
VEMF	2×7=14	2×9=18	2×7=14	3 ×5=15	2×5=10	3×3=9	3×3=9	1×7=7	1×9=9	105
AML	3×7=21	2×9=18	2×7=14	2×5=10	1×5=5	3×3=9	2×3=6	1×7=7	1×9=9	99
MatchBox	2×7=14	0×9=0	2×7=14	2×5=10	0×5=0	1×3=3	2×3=6	0×7=0	1×9=9	56
ECL	0×7=0	0×9=0	0×7=0	0×5=0	0×5=0	2×3=6	2×3=6	0×7=0	1×9=9	21
EMFCompare	0×7=0	0×9=0	2×7=14	0×5=0	0×5=0	2×3=6	2×3=6	0×7=0	1×9=9	35
EMF Editor	2×7=14	0×9=0	2×7=14	2×5=10	2×5=10	1×3=3	1×3=3	1×7=7	1×9=9	70
XMF	2×7=14	0×9=0	3×7=21	1×5=5	0×5=0	3×3=9	3×3=9	0×7=0	1×9=9	67
THEM/UML	3×7=21	0×9=0	1×7=7	0×5=0	0×5=0	3×3=9	1×3=3	0×7=0	1×9=9	40
GME	2×7=14	0×9=0	2×7=14	1×5=5	2×5=10	0×3=0	0×3=0	1×7=7	1×9=9	59
France et al	2×7=14	2×9=18	2×7=14	1×5=5	0×5=0	0×3=0	0×3=0	1×7=7	1×9=9	67
MATA	1×7=7	0×9=0	1×7=7	0×5=0	0×5=0	2×3=6	2×3=6	3×7=21	1×9=9	56
Muller et al	2×7=14	0×9=0	2×7=14	1×5=5	1×5=5	2×3=6	2×3=6	1×7=7	1×9=9	66
The value corresponding to the first Table						**+ + + : 3; + + : 2; + : 1; - : 0**				

Table 3 process results

We conclude that many works are close to our proposed conceptual prototype (the nearest works in the Table 4 are marked in gray). The EML is similar to the approach AMW approach in the

sense that there is a pre-establishment phase of relationships (rules of correspondence / weaving model) and a use of additional programming languages. (See Figure 25)

There is also a large difference in maturity of these approaches. Some of them are equipped and therefore can be exploited while others offer only prototypes.

The Kompose approach is suitable only for structural models: currently this approach only supports class models. In addition the designer must set the matching process by defining signatures at meta-model level to define specific matching operators.

Compared to EML and AMW that are based on symmetrical models, where no distinction is made between source models, Kompose is in the line of asymmetric approaches with two types of models: a basic model that plays a key role in the composition, and the aspects models, which contain the cross-woven concerns of the basic model.

The package merge mechanism in UML defines how the content of a package is extended with the contents of another. Therefore, the composed model includes all the properties of the two models: no filter is applied to the content of the composed model.

This mechanism does not detail the progress of the mapping phase and has no solid theoretical foundation, since it is based only on a names-based matching and does not treat advanced semantic relationships.

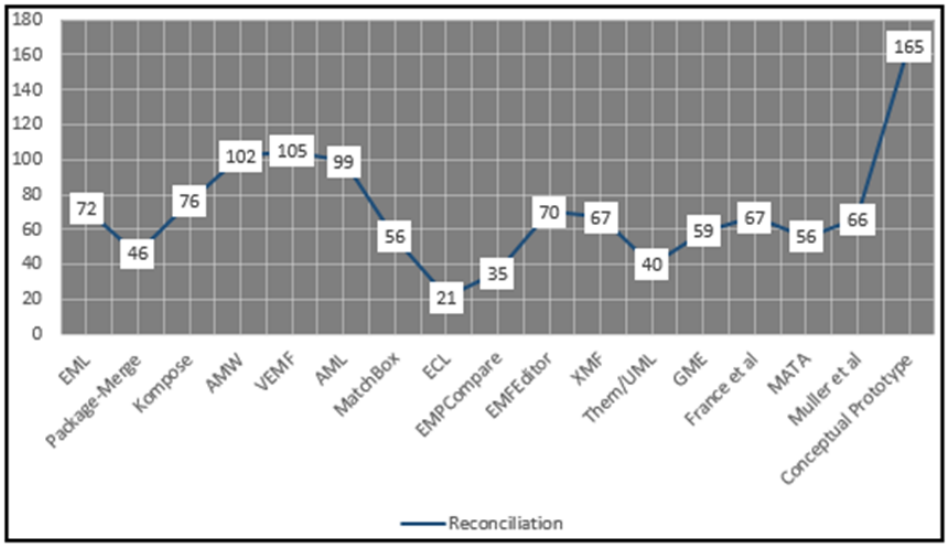


Figure 25 Reconciliation curve for each approach

III.8. Conclusion

The modular approach allows to build systems by assembling modules. The idea is to connect these modules through their interfaces.

The main interest of the modular approach is to facilitate the construction of systems by allowing 1) to write the module with little knowledge about the other code modules, and 2) to replace one or more modules without reassemble all the system. The construction of the system is more understandable, manageable, and maintainable.

One limitation of the approach is we don't have the possibility to personalize a module, it means to allow make many variations of an existing module.

The Architectural Description approach focuses on designing systems by assembling architectural bricks. Three concepts of architectural bricks are popularly accepted component, connector and configuration. Architectural bricks are modeling portions of an abstract system without defining their implementations, so we can use these portions when the system is in the analysis phase to allow create the architecture description.

This description can help later to make simulations on the system or allows other people to analyze the system. Therefore the variety of components created compatibility problems, portability, multiplicity, complexity and lack of standardization.

The software engineering based on components approach strongly promotes the concept of composition. The applications are built by assembling components through their interfaces. There are two types of assembly or by direct connections between the interfaces, or by connectors which link them indirectly.

The main interest of the approach component-based reuse is high. This can significantly reduce application development time.

The fact that the concept of component is not clearly defined has the effect of making vague the concept of composition.

This make it difficult to define composition standards effective mechanisms. Noted that the composition of the components is a fundamental and important activity in the component approach but there are not many research efforts for achieving the composition, compared to the efforts to define new models components. These are the factors that limit the productivity improvement and the reuse approach.

The AOP [GLF04] approach enables to encapsulate crosscutting code of an application in the separate software units called aspects and build these units within the core program. The weaving is carried out either at compile time, runtime or in both phases. Conflicts between woven aspects in one place can be detected and controlled by the weaving contracts and / or by the weaving order controls. The interest of the AOP approach is to avoid redundant codes that frequently appear in several places in the application, and thus increase the reuse of these codes by their composition in several different applications. A limit of AOP approach is the resulting code can be very complicated, difficult to understand, test and debug. However, the control mechanisms of the current weaving order still need lots of improvements to better manage discrepancies, conflicts during weaving.

We think that this study is a good way to find a new and more difficult effective way of action. In the light of these six composition techniques, we intuitively identify four similarities as follows:

- Every technique composes a pair of models.
- Every technique proposes a mechanism for detecting similar or equivalent model element.
- Every technique proposes a mechanism that uses matching for combining models.
- Every approach proposes a mechanism of composition based on the components detected in the beginning.

So, this study allows us to realize that there are two categories of composition: the white box composition which is involved in the internal structure of the components, and the black box composition which comprises components as they are without any change. We can find it even in the model composition in which there are also two types of composition: one allows to compose component as they are, and the other composes them but after that their structure is transformed.

As we mentioned before it is observed that some composition techniques already proposed various operations on a set of models. In special cases, reusing or adapting these techniques seems an interesting path to build a new composition models operations. In another side we suggest using this taxonomy to create a novel composer framework to automatically select the appropriate composition method for a given problem.

After analyzing the different approaches in the field of model composition, we have been identified some criteria that influence their implementation; our goal is develop these criteria to fill up existing approaches gaps. Indeed, some approaches deal only with matching problem, while others feel this issue pre-board as a step in a larger process.

The composition has an impact on three levels: Syntactic level: Matching input models expressions to produce a target model expression-Semantic level: assigning a semantic to the composed model, depending on the semantics of the associated source models.-Methodical level: The use of the composed model depends on the semantics of the associated source models.

We can notice that there is a correlation between the power of the approach and its complexity. However, approaches that the result is satisfactory are difficult to handle, carry and requires in most cases a human intervention, the simplest approaches to implement do not produce accurate results.

Regarding conflict management and model comparison—the proposed resolution of inconsistencies are limited and do not match the needs of the user, at the composition level there is a shortage at the reusability of composed models. Indeed, the models evolve over time and the change in one of them may cause the inconsistency of the correspondence model, hence the need to reflect the changes, or at least to identify items models that will be impacted by the changes.

Concurrently, the identified approaches do not fully address this aspect of the evolution system. This is because they do not exploit the links established beforehand to provide a synchronization mechanism that ensures the consistency of the overall system.

Several techniques of composition method have been suggested in the literature. However, there is no work that considers the coexistence of these different composition methods in order to answer practical questions such as: when should we prefer one composition method over the other? Is it possible to solve a given problem of a several composition methods?

CHAPTER IV. A CONCEPTUAL PROTOTYPE FOR MODEL COMPOSITION BASED ON THE TWO HEMISPHERE MODEL DRIVEN ARCHITECTURE

IV.1. Introduction

According to the formalization of model composition proposed in Chapter III, we present in this chapter a conceptual prototype for model composition that supports the definition an intuitive process based on the two hemisphere model driven architecture.

Our primary motivation for a conceptual prototype compromised of the two-hemisphere model driven approach, which proposes the use of business process modelling and concept modelling to represent the systems in platform independent manner and describes how to apply transformations from business process model into UML models.

The strategy use an intermediate model, which is received in a direct transformation way from process model, then extract appropriate interacting objects from concept mode to produce a class diagram based on concept model and formed according the information about object interaction. In general, the two-hemisphere model driven approach starts with the CIM phase by presenting the specification of the system at problem domain level in the form of two diagrams (business process model and concept model) and then uses many transformations in the context of PIM phase to provide a UML diagrams.

The business process model (graph G1 in Figure 26) is interrelated with the concept model (graph G2 in Figure 26) as shown below. Certain concept in the concept model defines the data type for one or several data flows between business processes. [NKA+15]

The business process model is transformed into an intermediate model (graph G3 in Figure 1), where the edges (i.e. data flows) of the business process model become nodes of an intermediate model, and nodes of the business process model (i.e. processes) become edges of an intermediate model. Semantics of the nodes and edges of the intermediate model is the same as of the UML communication diagram (graph G4 in Figure 26), where nodes of the intermediate model are the edges (i.e. objects) of communication diagram, and nodes of the intermediate model are the edges (i.e. messages to perform the operation) of the communication diagram.

The essence of the transformation is illustrated in Figure 1. Two hemisphere model driven approach is here to answer one of the most important and problematic stages in MDA realization, that is the derivation of PIM elements from a problem domain, and PIM construction in the form that is suitable for the PSM. In chapter, we are trying to carry out the concept of two hemisphere model driven

approach on the model composition by proposing a conceptual prototype based on the composition of several models coming from different source. The process of two hemisphere model driven approach as shown in Figure 26.

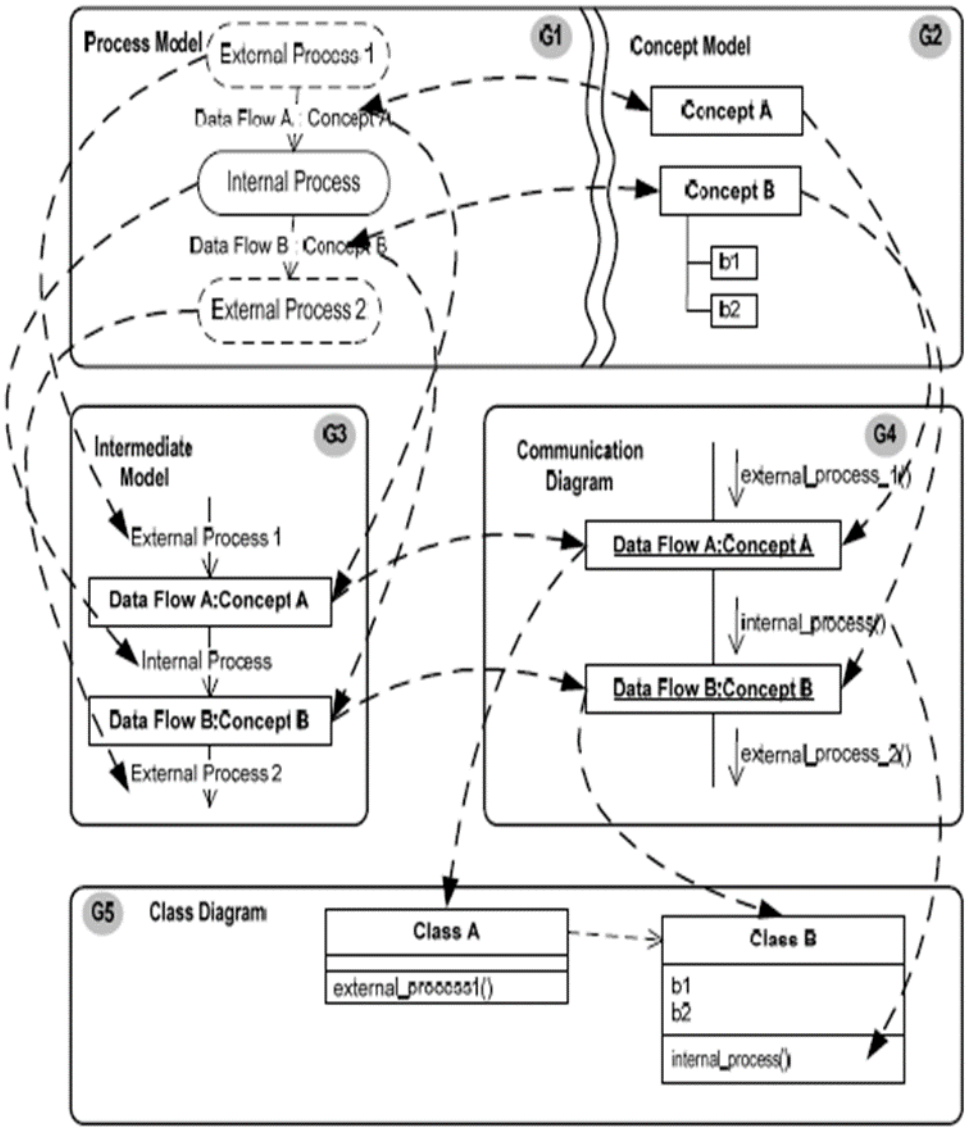


Figure 26 TWO-HEMISPHERE MODEL TRANSFORMATION INTO CLASS MODEL [NKA+15]

IV.2. Model composition within the MDA

Models offer many advantages. The most important benefit they provide is to specify different levels of abstraction and helping manage the inherent complexity of applications.

In this context, the variety of modeling capabilities and the ability to express links traceability are decisive assets to manage complexity. Another clear advantage of models is that they can be presented in size graphic, thereby facilitating communication between the actors of IT projects.

The graphical models used are among the most relational models, which allow to specify the structure of the databases. The graphical representation of these models offers a significant productivity

gain. The gossips say that model is the best way to lose time because, ultimately, we need to write code anyway.

Similarly, the famous saying stating that a good diagram is worth a thousand words, sometimes we hear replicate that a scheme can match more than a thousand speeches, depending on how we the interpreter. These criticisms are aimed right in the absence of knowledge of good modeling practices, that is to say, the model engineering. This is why it is essential to acquire good modeling practices to determine how, when, what and why model and to fully exploit the advantages of the models.

The OMG has defined MDA for this purpose. The approach MDA recommends the widespread use of models and offers the first answers how, when, what and why model. Without claiming to be a Bible modeling, listing all the good practices, it aims to highlight the qualities intrinsic models, such as sustainability, productivity and integration of platforms execution. The principle key of MDA is the use of models for different phases of the application development cycle. Specifically, MDA advocates the development of requirements models (CIM) analysis and design (PIM) and code (PSM). [MDA06]

CIM specify client requirements in order to create a model of the future application requirements. This model should represent the application in its environment to define the services offered by the application. Creating a requirements model is an important step, thereby it helps to articulate the traceability links with models that will be built in other phases of the application development cycle, such as analysis and design models.

PIM provides formal specification of the system structure and functions that abstracts from technical details, and thus presents solution aspects of the system to be developed.

Once the analysis and design models produced, the code generation PSM can begin in order to facilitate code generation from an analysis and design model. This step provides all the information necessary to operate a platform for execution, such as information for manipulating file systems or authentication systems.

An MDA idea is promising—raising up the level of abstraction, on which systems are developed, we could develop more complex systems more qualitatively. In this context, model composition is an essential activity and a new research topic in the MDA development process. In general a model composition tool combines two or more models into a single one to provide a general representation of the system.

However, there is no any model composition solution, where complete transformations CIM to PSM including composition paradigm in the PIM phase has been defined in the literature.

IV.3. Model composition process

A model composition should provide means to support common features for building a sophisticate composition operator.

The survey presented in the chapter III summarizes a core set of minimal requirements for a model composition framework.

Indeed, to allow automating the model composition in the MDA framework, authors propose a composition process which is made on three main steps called comparison, Weaving and verification, thus a repository to manage conflict is implemented separately, however every module or phase might be defined in a way such that they interact with this repository.

IV.3.1. Overview

Our conceptual prototype is based on the two-hemisphere model driven approach to design a system by assembling a set of class diagrams coming from different sources.

This approach is part of the multi-model engineering. Methodologically, the approach begins with a breakdown of the studied system to master and manage better its complexity and evolution.

The major goal of our conceptual prototype is the focus on MDA (Model Driven Architecture) approach as a whole concept by building a global class diagram representing the entire system by assembling a set of class diagrams presented in PIM phase as the business process model and the concept model according to the two-hemisphere model driven Approach.

The composition operation will be performed by following three steps—comparison, Weaving and verification of the resulting model; These phases have a direct link with a set of constraints called conflict management repository, to ensure conformity between models made and guarantee the consistency of the assembly throughout the composition process as presented in the Figure 27.

The comparison phase begins when we receives the two input models. The activity performed in this phase is the identification and analysis of the input models.

The goal of the comparison phase is to define what input model elements are similar by identifying the set of links between models to be composed. It is initially governed by comparison rules which implement the comparison strategy presented in [28].The comparison of elements is divided into three parts—Lexical comparison (Identify lexically equivalent concept), structural comparison and Syntactic comparison. Then the weave operation, with the aid of a set of previously comparison strategies, match the input models elements referring to the type of relationship raised and matching rules.

The phase is finished as soon as the matching models, no matching models and matching description are specified. The final stage is the verification of the produced composed models by specifying properties to verify [31] [32].

While a rule is violated an error is detected and it creates a problem model which conforms to a problem metamodel.

The problem model can be analysed referring to the conflict management repository dedicated to resolve problems.

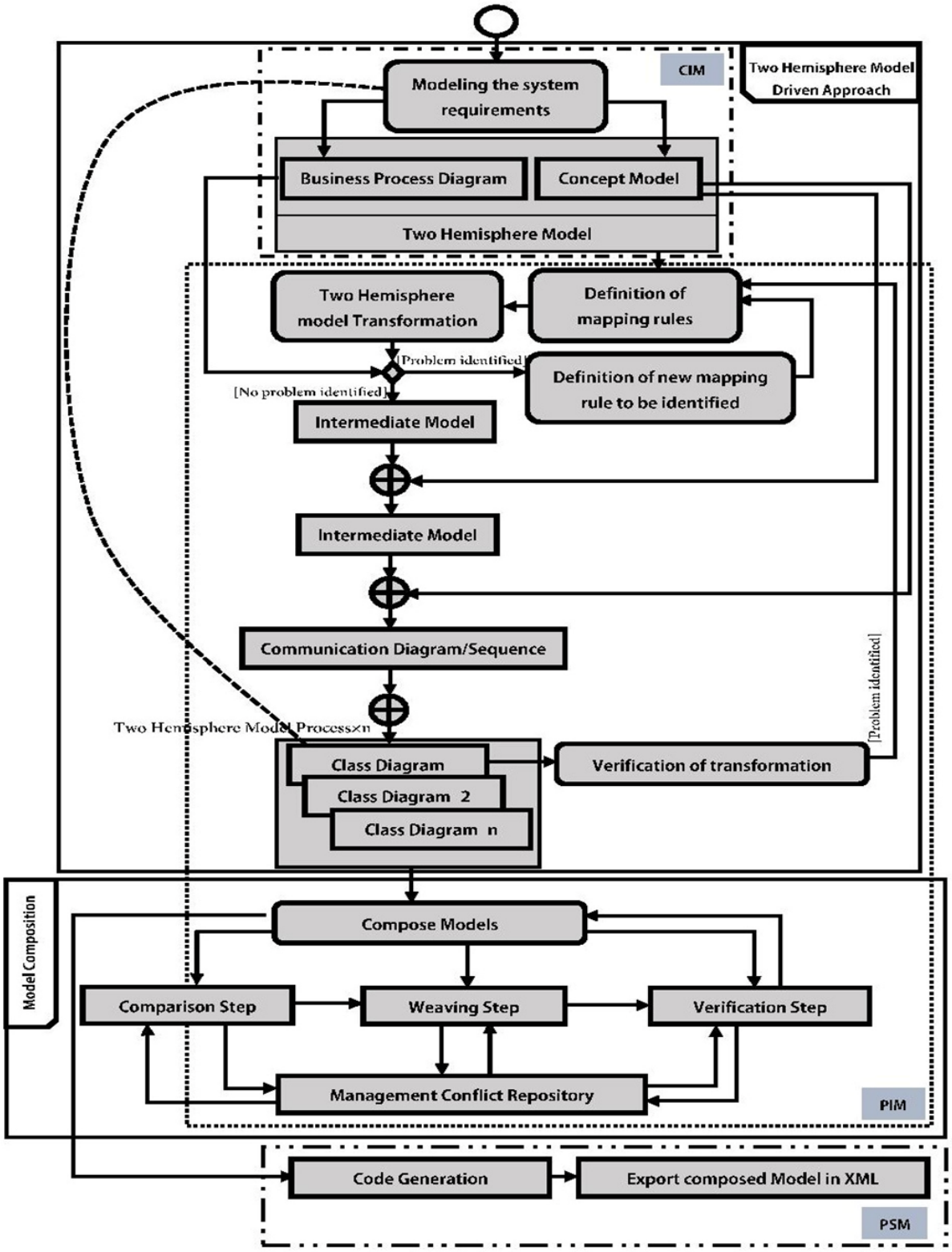


Figure 27 THE CONCEPTUAL PROTOTYPE PROCESS

The composition mechanism implemented is structured in three major steps: (See Figure 28)

Comparison:

This phase contrasts the input models in order to calculate their differences [33]. The information obtain in this phase is taken into consideration to help inferring the operations performed in each module.

This phase is critical to the final result of the merge process because the other phases are based on information collected by it.

The main goal of this phase can be summarized in identifying model elements that describes the same concepts in the different models that have to be merged. During the analysis of conflicts in this phase through our conflict repository, if elements are equal or equivalent then there are no conflicts. In this case, the weaving phase can be performed. Otherwise, the existing conflicts must be resolved (Table.5) before continuing the weave process. After resolving the conflicts, the weaving phase can be performed.

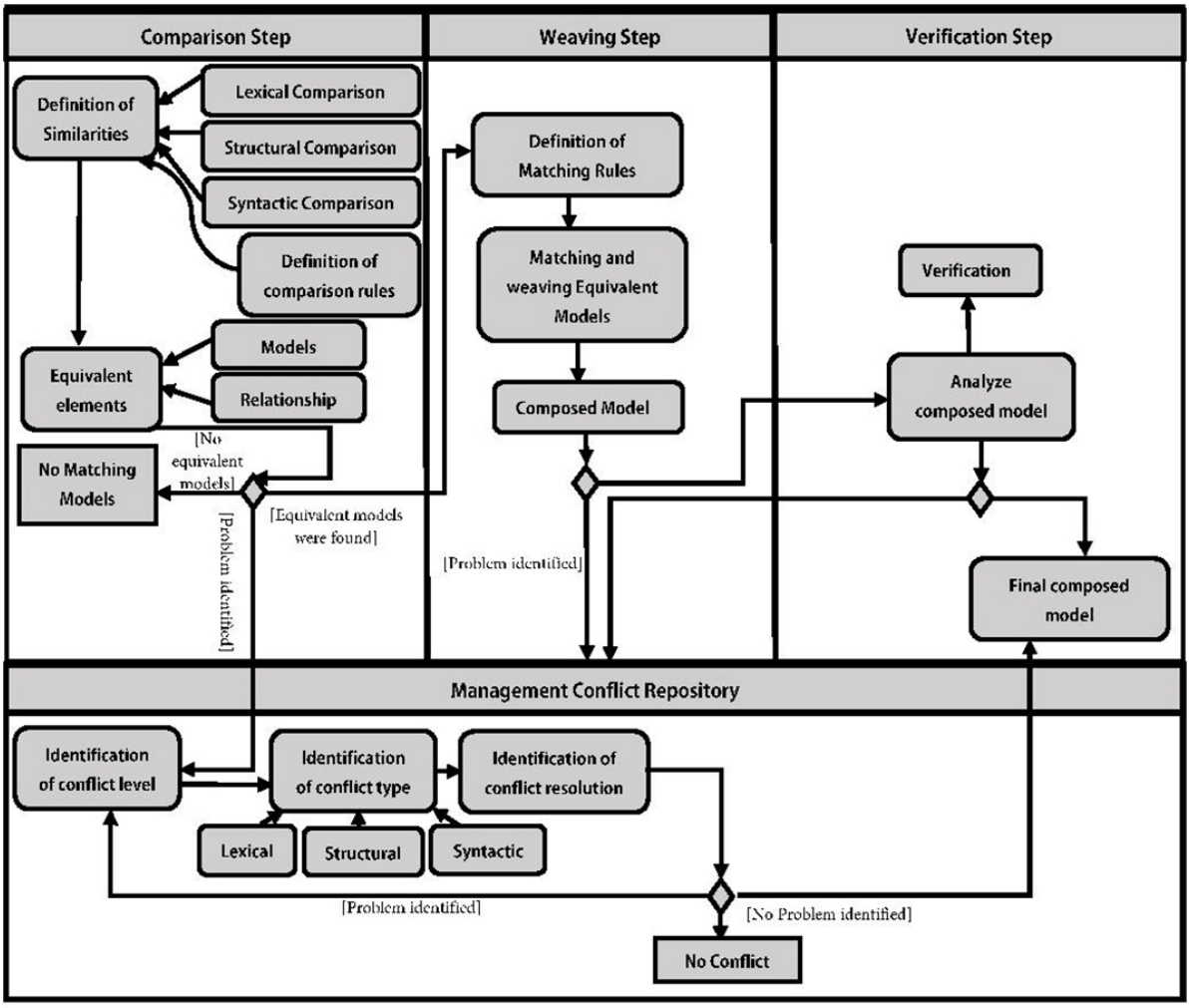


Figure 28 conceptual prototype for model composition

Weaving:

In this phase comparison model elements are merged to create new model elements that represent an integrated view of the concepts.

The merging operator builds a new model from two models. It merges equivalent elements according to the comparison operator and creates new elements in the target composed model. This operator is independent of a specific domain.

It consists in going through the set of elements that match in both input models and if they can be merged the operator creates a new model element in the output model. If the elements cannot be merged, a conflict is detected that has to be solved according to the conflict repository.

This happens when compared elements based on a subset of their properties (e.g., when merging two class diagrams, classes with same names) cannot be merged because of other properties that do not match.

The knowledge for detecting model elements that describe the same concept is based on information dependent on the meaning of the model. Thus, the weaving operation has to be specialized for each modelling language. However, in order to interact correctly with the merging operator, the comparison operator has to have a clear interface.

The weaving metamodel described in this section implements the behaviour of the merging operator and offers a precise interface for its. The prototype can then be specialized by providing specific matching operators through this interface.

Verification:

This phase has as a main goal to evaluate the target model in terms of the completeness and quality regarding the result provided by a manual specification.

We can also formally analysed the composed model [32] against desired properties to uncover design error by identifying badly formed models using the well-formedness rules. In some cases, the uncovered problems of verification can be resolved using the repository of managing conflicts.

In other cases, more substantial changes may be required. This chapter focuses on the global process of composition shown in Figure 27. Activities related to the verification of models is not within the scope of our work.

Management conflict repository

However, to uncover any conflict during the composition process, we have also defined a repository to manage conflicts, which will be represented separately as a generic module that communicate in real time will all others modules. The repository operates as follow:

Management conflict repository: it can be considered as conflict detection phase that verifies whether the differences contain conflicts. There are various types of conflicts such as lexical, syntactic and semantic conflicts.

Indeed, conflict is a set of contradictory changes where at least one operation performed by a developer does not comply with at least one operation performed by another developer [UAW06].

A good conflict detection method should minimize the occurrence of false positive and false negative conflicts. The former are non-conflicting changes marked by the detection method as conflicts [UAW06]. The latter are conflicting changes not marked by the detection method as conflicts. False

positives conflicts reduce the productivity of developers since the time taken to analyse them could be used for other tasks. False negatives conflicts produce even more harmful consequences, as they may be ignored by the development team, leading to inappropriate merges.

When the conflict detection method considers only the syntax of diagrams, it is restricted to the detection of syntactic conflicts. These conflicts are detected by structural comparison between the versions of diagrams [Uchitel, 03].

As the method does not recognize the semantics of diagram, equivalent representations can be diagnosed as conflicting (false positives conflicts). Understanding the semantics of the diagram allows the identification of different representations that have the same meaning.

Thus, the method becomes able to ascertain whether syntactically different versions are semantically equivalent. This helps in reducing false positives conflicts.

Moreover, understanding the semantics of diagram allows the detection of semantic conflicts. A semantic conflict occurs when changes performed in parallel not only interfere with the modified element, but also in others. For example, semantic conflicts may occur when a developer modifies an element that depends on another element modified by other developers [Rumbaugh et al., 96]. As semantic conflicts are more difficult to detect, they can generate false negative conflicts.

Therefore, understanding the semantics of the diagrams can also help reducing false negative conflicts. In what follows we detail each phase metamodel and related activities of the proposed model composition conceptual prototype.

IV.3.2. Comparison phase

IV.3.2.1. Comparison Metamodel

The composition metamodel describes how the composition prototype can be accomplished. For this purpose, we have extended the core comparison presented in [KPP06a] [MT00] to support composition requirements and to handle new relationship types. The metamodel shown in Figure 29 describes the generic comparison process to construct a model composition prototype. The key elements of the comparison metamodel are:

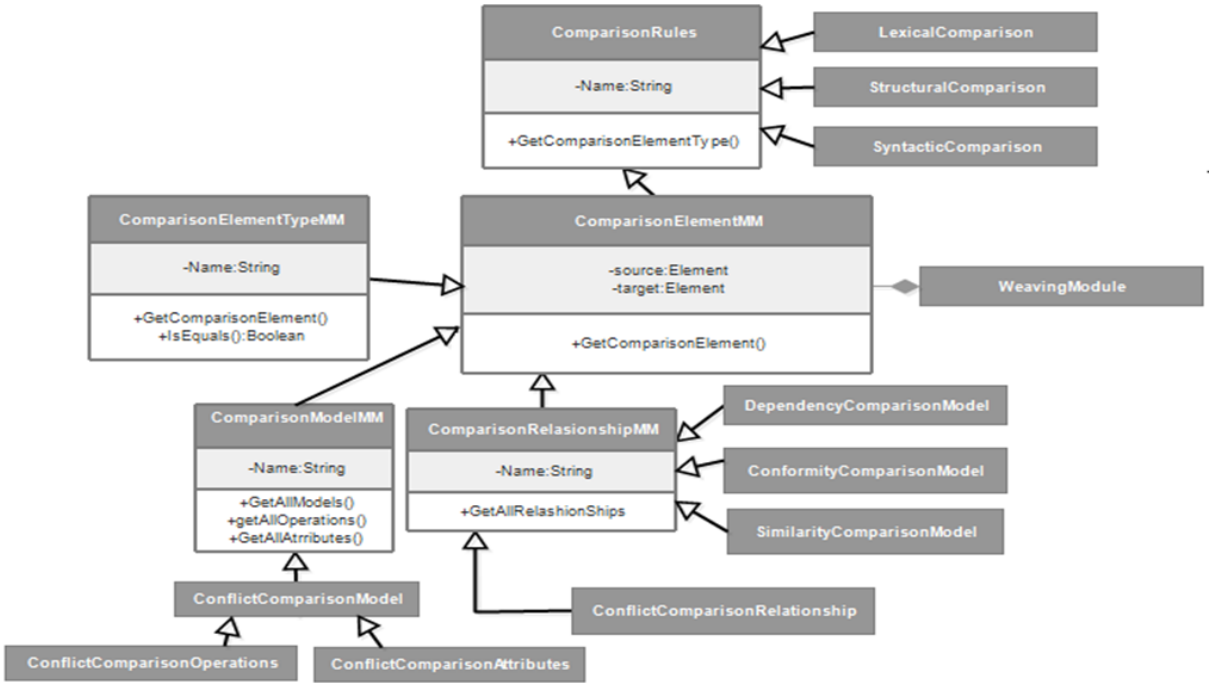


Figure 29 Comparison phase metamodel

ComparisonEelementMM:

Compare two class diagram by checking that their names, attributes, operations and relationships match. Moreover, to conform, their types must also match. A type of comparison element has to be specialized by ComparisonEelementTypeMM to define the algorithm for weaving elements. The getComparisonElementType operation defines the types of the model elements.

This type is compared with the type of other model elements to check if these elements have to be matched. A type of comparison element has to be specialized by ComparisonEelementTypeMM to define the algorithm for weaving elements. The getAllComparisonElementType operation defines the types of the model elements. This type is compared with the type of other model elements to check if these elements have to be matched.

For example, two methods in a class diagram can match because they have the same name or because they have the same name and the same parameters. In the first phase of comparison, the getComparisonElementType operation will return the name of the methods and the list of parameters and it is necessary to start the weaving phase. The match operation in class WeavingElement implements the generic algorithm for merging two

Model elements. The complete algorithm is defined in [MT00]. If two model elements match according to their signature in our case their types, this operation tries to match them into a new model elements.

The algorithm compares the values of each property of the elements to merge in order to detect possible conflict. If no conflict is detected the new model element is created, otherwise the conflict must be solved using the repository of managing composition conflict.

ComparisonRules and ComparisonEelementType:

The comparison process is governed by a set of comparison rules applied in the comparison phase to achieve consistent model composition lays in the ability to compare input model elements. Before composing elements, it is necessary to verify the existence of lexical, structural and syntactical overlaps.

Overlaps between input models are undesired as they can lead semantic conflicts, misinterpretation and problems in the transformation process. For example, according to the UML metamodel specification [29] there should not exist two (or more) model elements, such as two UML classes, with equal names in a same namespace.

Therefore, a model composition mechanism attempting to compose UML Class Diagrams should take into account such constraints and avoid creating output models with conflicting names and/or elements with the same semantic value. The comparison rules are responsible for identifying constraints and defining the correspondences among input model elements, thus making similarity and overlapping explicit. UML models are often used to describe things that exist the real world is.

They are useful to graphically depict a system's structure and behaviour from different viewpoints and at various levels of abstraction. A set of UML models can be used to better manage the description of a system, where each model in the set captures a different aspect of the solution [KPP06a].

Performing sophisticate comparison rules requires a clear understanding of the UML metamodel specification and semantics rules. (As mentioned in the Table 2).

Number	Rule
1	Since interfaces are declarations, they are not instantiable. Instead, an interface specification is implemented by an instance of an instantiable classifier, which means that the instantiable classifier presents a public facade that conforms to the interface specification.
2	An abstract aggregation does not provide a complete declaration and can typically not be instantiated. An abstract aggregation is intended to be used by other aggregations.
3	In a mapping relationship such as Derivation, it is usually formal and unidirectional. In other cases, such as Trace, it is usually informal and bidirectional. The mapping expression is optional and may be omitted if the precise relationship between the elements is not specified.
4	In a generalization relationship, for each attribute in the superclass, an equivalent attribute is inferred for the subclass unless the attribute in question has a private visibility.
5	In a generalization relationship, for each operation in the superclass, an equivalent operation is inferred for the subclass unless the operation in question has a private visibility.
6	In a generalization relationship, for each association in the superclass, an equivalent association is inferred for the subclass.
7	In a realization relationship, for each attribute of the interface, an equivalent attribute is inferred for the class.

8	In a realization relationship, for each operation of the interface, an equivalent operation is inferred for the class.
9	If a class is abstract and all its methods are abstract then, if there is an interface that has the same name and the same elements, they are semantically equivalent.

Table 2 Uml semantics rules [UML-OMG]

IsEquals Operation:

A default comparison is implemented in the IsEquals operation to check if the elements type is equal to the type of another elements.

ComparisonRelationshipMM:

The generic comparison relationship metaclass must be extended to create different relationship types. Defining extensions is a practical solution to represent a specific semantic for each comparison relationship. For instance, SmilarityComparison indicates that the linked elements represent the same concept but differ in some properties (e.g. two classes with the same name and different sets of attributes), the DependencyComparison denotes dependency information between two elements, and the ConformityComparison defines the link type between elements.

IV.3.2.2. Unique comparison algorithm

We propose a unique algorithm for comparing model elements. The comparison 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.

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. The result of the comparison function allows merging model elements that are identified as equivalent. (See Figure 30)

```
429  function match($model1, $model2){
430      // compare the equivalence between propreties
431      $model1 = new object();
432      $model2 = new object();
433      foreach($model1 as $m1){
434          foreach($model1 as $m2){
435              //compare propreties
436              if($m1===$m2){
437                  return true;
438              }else{
439                  return false;}
440          }
441      }
442  }
```

Figure 30 A unique algorithm for comparison phase

IV.4. Weaving Phase

IV.4.1.1. Weaving metamodel

Our weaving process is also governed by a set of weaving rules. Indeed, the composition operation can be specified using the model comparison principles.

The model comparison enables us to establish correspondences between input models, and then to weave the global model by merging the models resulting from the comparison phase.

However, due to conflicts and dependencies between models, it was necessary to define a repository to manage these conflicts, every phase has a link with our repository and every elements that generate a problem are transformed according to several rules.

Elements having the same signature are merged to form a single element. To support the alignment of elements in our conceptual prototype, each element type is identified by a type which contains the class name, attributes names and the arguments lists.

The element type is a set of syntactic properties and it includes values related to its properties. For example, if a UML class named A, includes properties 'price' and is 'float', the type of this element is defined by the vector {name = A Price: float}.

If two models each have a class with the same name and with the same value for the property, the two classes will be merged to compose a single class in the global system. The merged class contains the union of attributes and operations defined in the inputs classes (operations and attributes that are syntactically equivalent will be included once in the merged class).

The repository use proposed algorithms [KLM+][JZM08][NSC+19] and resolve several conflicts based on a list of refactoring for class level, method level, field level and relationship level as represented in the Table.5.

The weaving metamodel contains the date structures and utility methods that are used to match compared elements based on their types in order to create the merged elements and keep a traceability information between the input models and the composed model.

However, modellers can specify weaving process that are used during composition to force matches, disallow merges, and to override default match rules. The weaving phase specify simple model modifications that should be made before the models are matched.

These changes are with a direct link with our management conflict repository in order to specify renaming model elements, removing or adding elements.

– Source and target models: the composition operator combines source models in order to produce the target ones. Hence, the composition specification has to be aware of the relevant information (models name, their metamodel.).

– Matching rules: applied in the weaving phase of a composition process in order to identify which concrete elements to match given as input the composable elements for the composition, they also describe the behavior needed to combine two elements that match with respect to some correspondences criteria. These rules have a name, a statements block which specifies the matching mechanism, and a set of parameters referencing the contributing elements.

– Merge: with the merge strategy two or more corresponding composable elements has been identified by the applied matching rules.

– Element: A language comprises a set of elements, like e.g. Class, relation, package, which allow the modeller to express certain concepts.

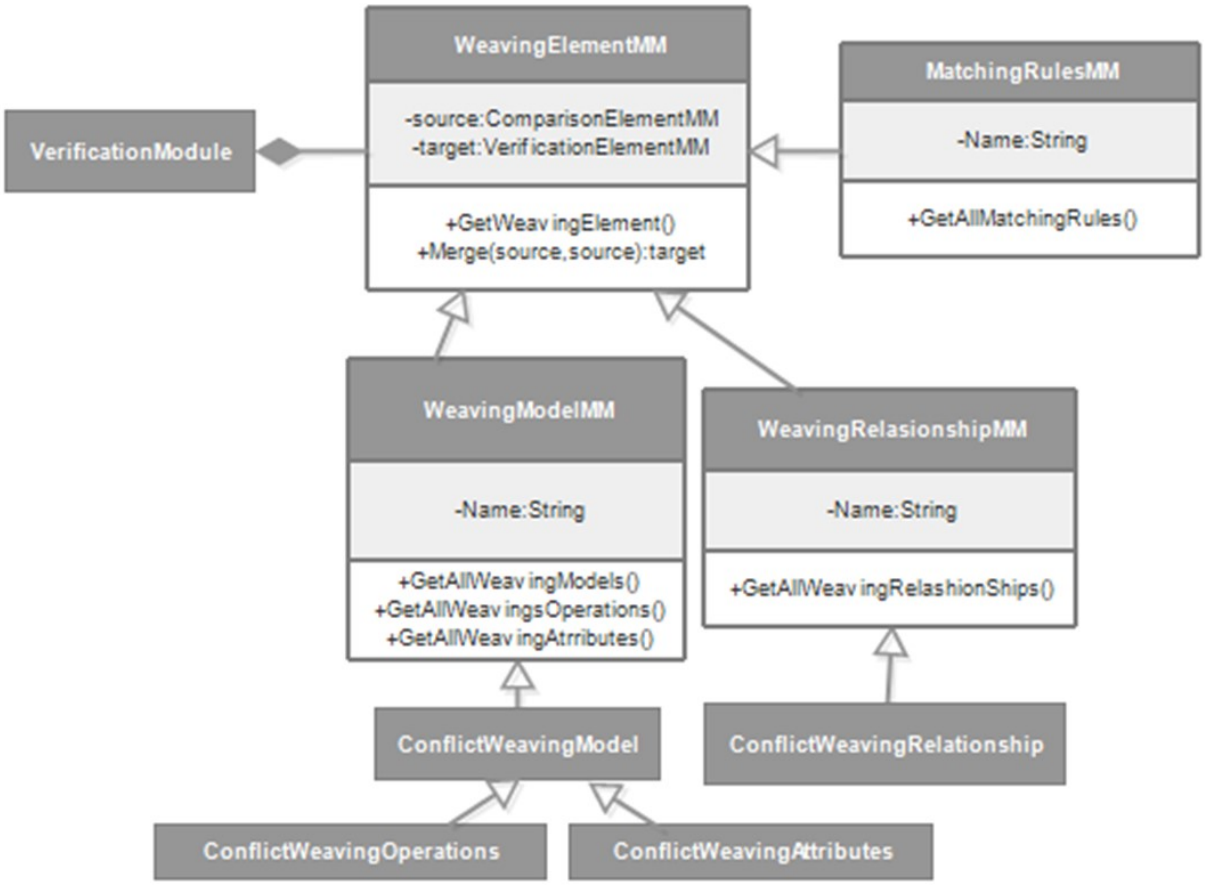


Figure 31 Weaving Phase Metamodel

IV.4.1.2. A Generic Weaving Algorithm

We propose a generic algorithm for deeply weaving two model elements. The sum() operation takes two elements as parameters and creates a new element as a result. The sum() function calls the match() function presented in the comparison phase in order to compute matches. Figure 30

```
function sum($model1, $model2){
    // compare the equivalence between propreties
    $model1 = new object();
    $model2 = new object();
    if(match($model1, $model2)==true){
    foreach($model1 as $m1){
        foreach($model1 as $m2){
            //compare propreties
            if(match($m1, $m2)==true){
                //if matching, sum propreties
               $resultat = $m1 + $m2;}
            if(match($m1, $m2)==false){
                //if not matching, create a new proprety (array) with the propreties
                $resultat = [];
                $resultat['m1'] = $m1;
                $resultat['m2'] = $m2;
            }else{
                //merge values and resolve conflicts if any
                $resultat = resolveConflicts($m1, $m2);
                }
            }
        }
    }
    return $resultat;
}
```

Figure 32 A Generic Weaving Algorithm

The concrete implementation of the weaving operator satisfies the following properties – Completeness: The implementation of the weave 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 weave 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 weave 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. 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.

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.

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.

```
public function merge($A, $B){
    //create objects
    $A= new object();
    $B = new object();
    $A = $B;
    $C = new A();
    //union of propreties
    foreach ($A as $p1) {
        foreach ($B as $p2) {
            if($p1 === $p2){
                //merge propreties
                $merged_value = resolveConflicts($p1,$p2);
            }
        }

    }

}
```

Figure 33 A Generic merging Algorithm

IV.4.2. Conflict composition repository metamodel

IV.4.2.1. Overview

The composition of separate concerns is a cornstone for the construction of complex software, however a large number of composition concepts have evolved and been successfully put into practice, but their abilities to cope with composition conflicts are mostly limited, that why defining a repository to manage conflicts seems a major task in large model composition projects.

Indeed, in this section we enhance the concept of managing composition conflict through a generic repository dedicated to the resolution of potential composition conflict in order to uncover composition errors and ensure that a composed model is produced with a level of credibility.

The main new idea is to propose a general repository for model composition activities, which can be applied to conflicting model composition errors in particular. As the first step toward a global repository, our research will be based on a set of actions for detecting and correcting composition conflicts.

Several studies [KLM+][NSC+19] have demonstrated that managing conflicts and dependencies can be done by identifying possible conflicts and give them a solution or a transformation rule. So in this phase of harmonization, authors have identified three possible conflicts—Lexical conflicts, structural conflicts and semantic conflicts.

Indeed, our goal is to we enhance the concept of managing composition conflict through a generic repository dedicated to the resolution of potential composition conflict in order to uncover composition errors and ensure that a composed model is produced with a level of credibility.

The main new idea of the research is to propose a general repository for model composition activities, which can be applied to conflicting model composition errors in particular. As the first step

toward a global repository, our research will be based on a set of actions for detecting and correcting composition conflicts.

Lexical conflicts: this kind of conflict is caused by the confusion of identifying lexically equivalent concepts; multiple meanings and synonyms of classes introduced in the design of inputs models coming from different teams of designers make the mission more difficult.

Indeed, a synonym dictionary is used in order to identify mappings among domain concepts that have equal semantic value. The great benefit of using synonym dictionaries is to pave the way for the domain specialists to explicitly apply their domain expertise in the matching process.

Structural conflicts: Here we distinguish between 2 types of conflicts—those related to the type of association between two classes, and those related to the inheritance hierarchy. We intend using the TreeDiff implementation available at [GJ06]. Our choice was based on its ability to identify structural similarities between trees in reasonable time. The result of the Tree Diff algorithm is the detection of concept equivalence groups.

They are represented as subtrees of the enriched ontologies. Concepts that belong to such groups are compared in order to identify if lexically equivalent pairs can also be identified among the ancestors and descendants of the original pair.

Syntactic conflicts: includes class invariants, constraints, and operations specifications formalized by pre- and post-conditions. As defined in [GJ06], here we can use the Typographic Similarity—where a syntactic property of a mode element defines its structure.

The signature is a collection of values assigned to a subset of syntactic properties in a model elements metamodel class. If an instance of a Classis an abstract class then isAbstract = true for the class, otherwise the instance is a concrete class, isAbstract= false. The set of syntactic properties used to determine a profile elements signature called a signature type [Reddy et al., 06].

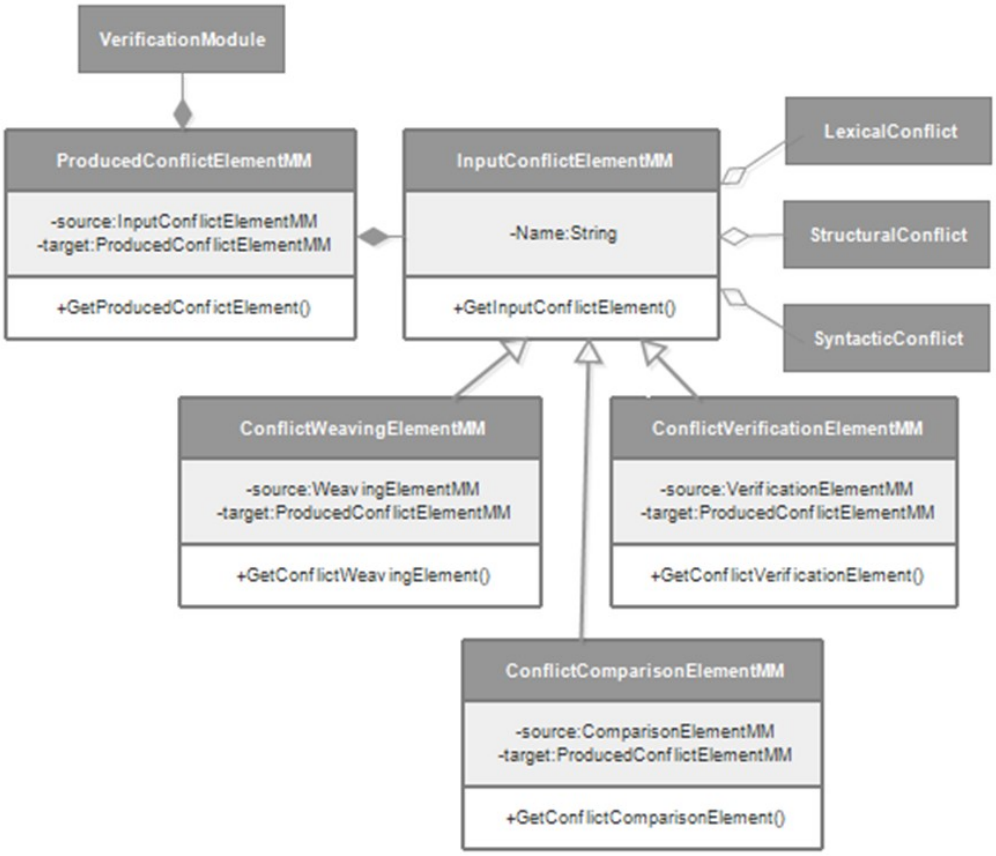


Figure 34 Conflict composition repository metamodel

Defining a conflict composition repository to manage conflict seems an important step in the implementation of our conceptual prototype. Every module might be defined in a way such that they interact with each other.

Our analysis of detecting and correcting composition conflicts will be based on a list of actions [34] that should be captured by the repository.

Moreover, understanding of the UML rules of class diagram allows for an efficient detection of conflicts. In general a conflict occurs when changes performed in parallel not only interfere with the modified element, but also with others.

For example, semantic conflicts may occur when a developer modifies an element that depends on another element modified by other developers [GKP07]. As semantic conflicts are more difficult to detect, they can generate false negative conflicts. Therefore, understanding of the rules related to the diagrams can also help reduce false negative conflicts.

IV.4.2.2. A Repository for Composition Conflict

In this section, descriptions of the repository are followed by illustrated examples. The set of resolution rules defined in this section is not intended to be a complete set, but serves as a starting point for the eventual definition of a complete set of managing model composition conflict.

There are several approaches to detect conflicts related to model composition. In this context we reason about managing conflicts as a separated module presented in the form of a repository which

communicate in real time with a composition framework or consistency analyser and can be used in every phase during the composition process as presented in Figure 35.

Some approaches pursue a more global way of reasoning by looking at the composition conflict as a secondary module included in the implementation of others modules.

To illustrate this, we take another look at this paradigm by trying to determine all kinds of conflicts interacting at several levels (Class, relationship, operations).

The authors are convinced that it would not be possible to detect conflict by looking only at the whole composition process: a separate context is needed to determine that the problem exists and then to propose a solution.

Indeed, our primary goal is to define a generic module presented in the form of a repository for the composition phases (comparison, weaving, verification), to precisely describe, detect and resolve composition conflicts.

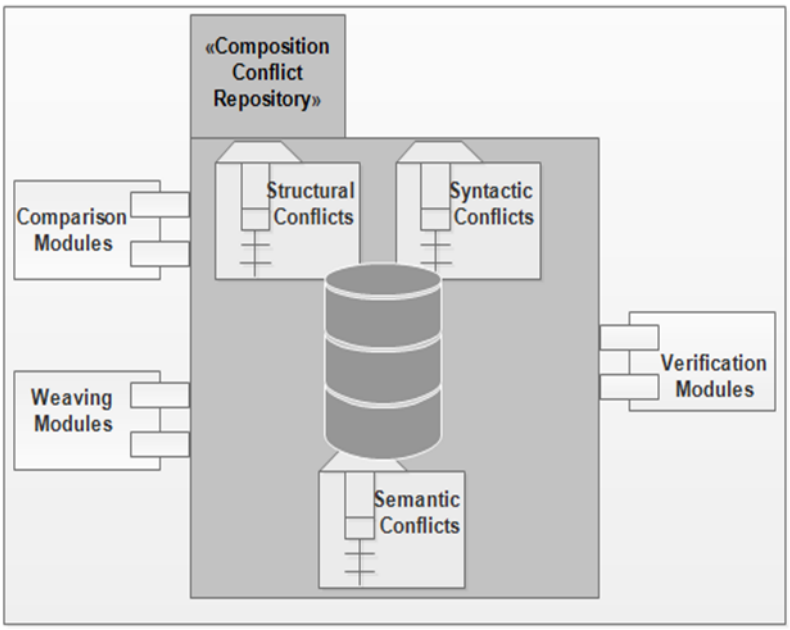


Figure 35 A graphical concept of the composition conflict repository

IV.4.2.3. Composition Conflict Categories

The main goal of our repository is to model, detect and resolve composition conflicts related to the model composition process. Within this context, we identify several categories of composition conflicts.

To analyse the causes of these conflicts precisely, we first identify probable causes. Next, we define explicit actions to resolve composition conflicts related to composition.

We built a methodology based on UML rules (Table. 1) that detects and analyses composition conflicts. As presented in the Fig. 2 we distinguish four categories of composition conflicts that can

occur when a composition process occurs: syntactic conflicts, structural conflicts and semantic conflicts.

The first type of conflict is the conflict caused by the multiple meanings and synonyms of models introduced as input models (classes, operations or attributes).

The second type relates to structural conflicts between classes. Here we distinguish between those who are related to the type of association between two classes, those related to the inheritance hierarchy.

The hierarchy of conflicts include inheritance cycles (which can appear when you want to merge the hierarchies from input models) and the level of conflict. The third type of conflict involves semantic conflicts that concern modeling elements. (See Table.3).

Number	Rule
1	In a mapping relationship such as Derivation, it is usually formal and unidirectional. In other cases, such as Trace, it is usually informal and bidirectional. The mapping expression is optional and may be omitted if the precise relationship between the elements is not specified.
2	In a generalization relationship, for each attribute in the superclass, an equivalent attribute is inferred for the subclass unless the attribute in question has a private visibility.
3	In a generalisation relationship, for each operation in the superclass, an equivalent operation is inferred for the subclass unless the operation in question has a private visibility.
4	In a generalisation relationship, for each association in the superclass, an equivalent association is inferred for the subclass.
5	In a realisation relationship, for each attribute of the interface, an equivalent attribute is inferred for the class.
6	In a realisation relationship, for each operation of the interface, an equivalent operation is inferred for the class.
7	If a class is abstract and all its methods are abstract then, if there is an interface that has the same name and the same elements, they are semantically equivalent.

Table 3 UML Semantics Rules [UML09]

Conflict Category	Conflict Types	Modeling element involved
Syntactic	Synonymy	Class, operation, attribute

	Polysemy	
Structural	Inheritance cycle	Class Generalisation Association
	Different level in hierarchy	
	Different types of associations	
Semantic	Contradictory semantic assertions	Class, operation, attribute

Table 4 the Different Conflict Categories to Deal with in the Composition Process

To resolve the conflicts presented in the previous part, we propose an interactive approach in three steps to address probable composition errors. The first step solves conflicts of polysemy and synonymy class hierarchy and conflicts. The second step can solve hierarchy conflicts using the same process. The third step solves conflicts over the semantic assertions.

Treatment of Polysemy and Synonymy Models

In general, the classes of a system are supposed to have different names. But as the input models can be made by various teams' designers, the risk of having identical names for classes with different roles is strong. Hence, there is a need for treatment polysemy in order to achieve consistent models. The resolution strategy follows these instructions:

• Identify models that have the same name in the input models.

• Determine if they have the same responsibilities.

• Solve otherwise the conflict by renaming models.

• Choose new names for these models.

Check that these new names will not make a new problems of multiple meanings in the various dictionaries of classes; otherwise, we should choose other names until there are no more problems.

Figure 3 below summarizes the key steps in the process of dealing with polysemy conflicts between input models.

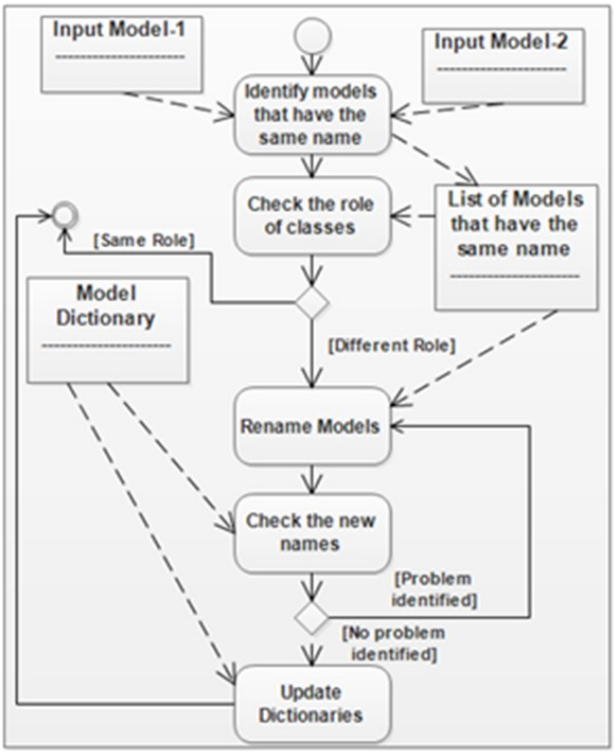


Figure 36 A graphical concept of the composition conflict repository

Treatment of synonymy is also an important step in the composition process to avoid duplication and ensure consistency between models. This treatment consists of:

• Identifying on the input models if there are any models that have the same role and different names;

• Renaming synonyms models that have the same name;

• Updating class dictionaries affected by this renaming.

Step 2: Treatment of Structural Inconsistency Conflicts

The resolution of this type of conflict aims to eliminate the structural inconsistencies between input models. This category includes conflicts of cycle' inheritance. These are problems of parent classes at different hierarchical levels, and inconsistencies between the types of association.

In case of inconsistency of the types of association between classes, we developed the following priorities based on the rules presented in Table 5 and some previous research [Gottlob et al., 96] [Gro09]:

• Association vs Navigable Association: In this case the association predominates, because otherwise we risk losing information. Thereby a navigable association is bidirectional by default. This means that if an association exists between two classes, then both objects know about each other. If "A" is the source class and "B" is the target class, the arrowhead would be placed on the "B" side of the association. A navigable association of this type means that at runtime object "A" knows about object "B", but object "B" has no knowledge of or visibility into object "A".

• Navigable associations in opposite direction: The relationship between the two models is transformed into association (without navigation) that provides access to the source information.

• Association vs Composition: The relationship becomes an aggregation; thus, to guarantee the concept of compound.

• Association vs. Aggregation: Aggregation predominate because it conveys structural information that we should maintain.

Step 3: Treatment of Semantic Conflicts

To automate the processing of this type of conflict, we must call a "semantic" tool in order to analyse expressions, detect conflicts and redundancies between semantic assertions established for an element defined in multiple input models. An example of resolving this kind of problem will be presented in the next section.

Conflict Identification Example: University Artifact.

To show the different composition conflicts in a real case we will refer to a modeling system of the university artifacts (Figure 37) made by different teams' designer, this example will illustrate the key elements of our solution.

We focus on the composition of two class diagrams, which correspond to different team's point of view.

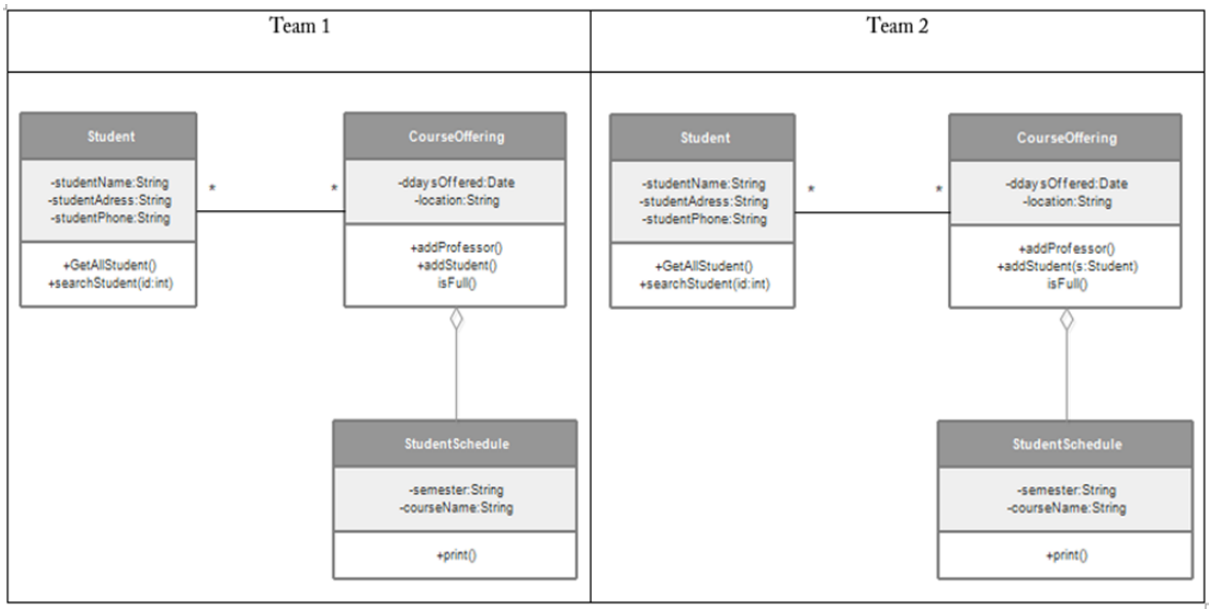


Figure 37 Class diagrams made by two different teams: University artifacts

Referring to Figure 37, we consider operations addStudent() and addStudent(s:Student) in a class CourseOffering that is part of a class diagram for the university artifact, the operation addUser() defined by the first team in a class named CourseOffering adds a student to a collection of student.

The addStudent operation in the second context of modelling calls the addStudent(s:Student) operation if and only if the user calling the operation is recognized to add a student.

The addStudent(s:Student) operation adds a user to the list. This, the composition of these two different operations produces a conflict because the two operations have different framing and involved in different context.

This is an example of a property conflict – a property conflict occurs when two matching elements (elements with the same name and syntactic type) are associated with conflicting properties.

In this example, the intention is to merge the addStudent() operation made by a team with the addStudent(s: Student) operation produced by another team.

To resolve this conflict, and based on some composition directives [HKR+07], we should rename the addStudent(s:Student) operation (First Team) to checkAndAddStudent, and keep the addStudent() operation in the same name. Renaming elements is not always a good solution to resolve this kind of conflict.

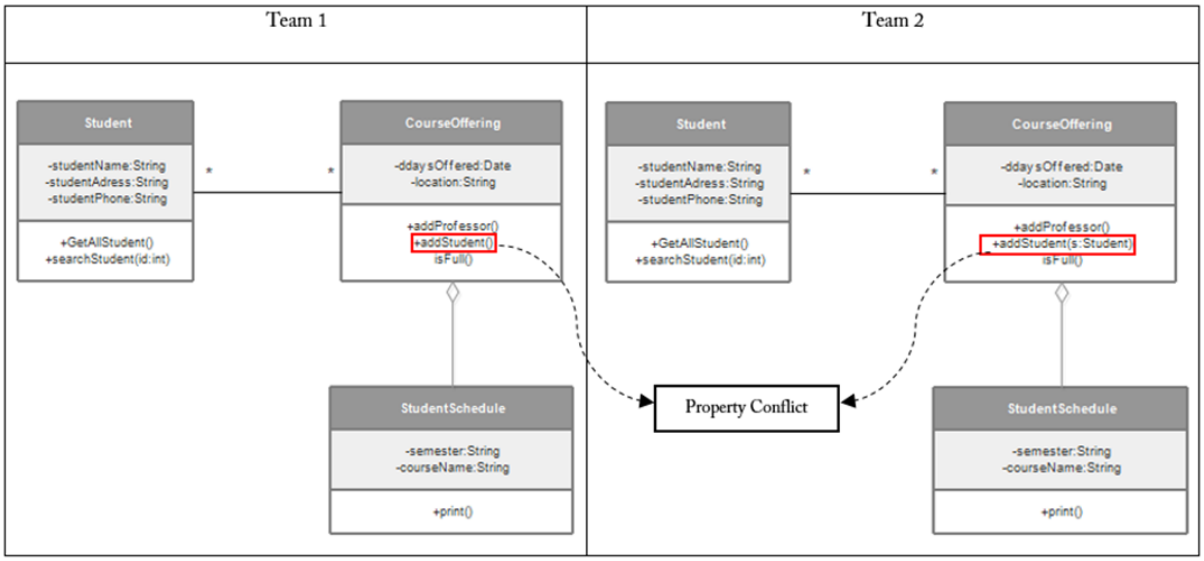


Figure 38 An example of property conflict during the composition process of two class diagram

As presented in Figure 38, the first class diagram contains a class CourseCapacity with an attribute maxStudent: int, which has the constraint {maxStudent = 10}. The second class diagram with the same name (CourseCapacity) and the same attribute (maxStudent), but different constraint {maxStudent = 12}.Indeed, applying a merge operation to the two class diagrams will provide a property conflict, particularly in matching attributes because their constraint ({maxStudent = 10 and maxStudent = 2}) is incoherent.

Yet, specifying an operation that will replace the other one can be a good directive to resolve this conflict. In this case, the properties operation with the high level of priority crushed the properties of the overloaded elements.

This solution has also a poor side, because overriding relations can provide a cycle of conflict, especially when the two elements have the same level of priority and are in direct link with dominant properties. This to say that some time, it is necessary during the composition process to apply others operations (add, delete) to the elements in the goal to ensure producing a refined result. [Liang et al06]

Our repository can be used to apply any kind of possible resolution during the composition process. If we take the example of an association, to guarantee and facilitate aces to an element, an association may be added or deleted (depending on the type of conflict) in order to avoid a security risk.

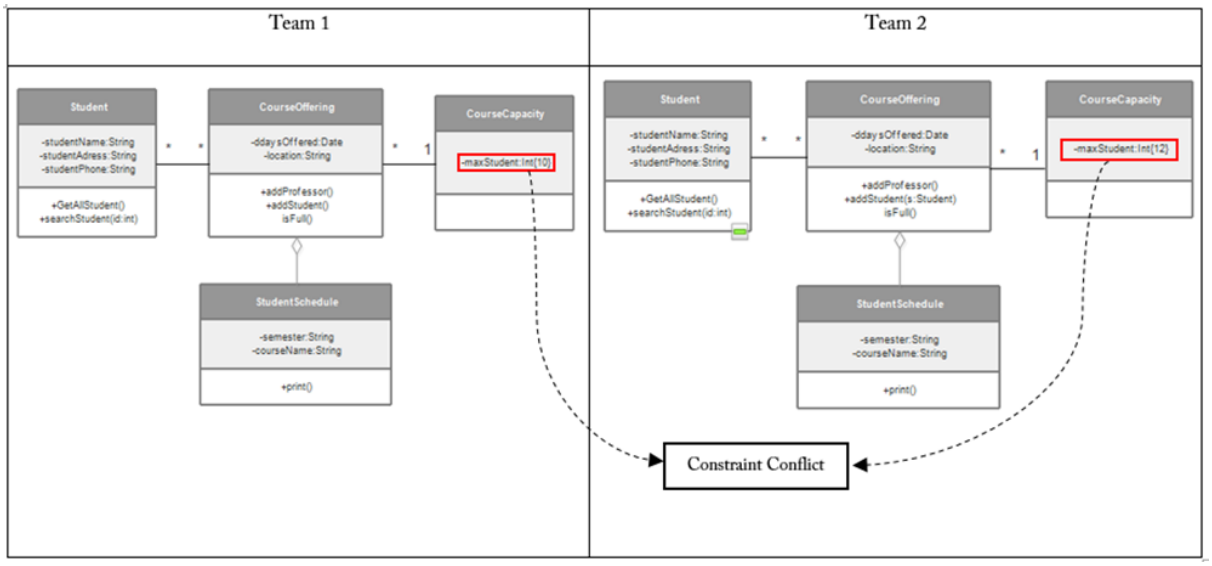


Figure 39 Example of constraint conflict during the composition process of two class diagram

IV.4.3. Conflict Resolution

In this section we present many tables(Table 5, Table 6, Table 7, Table 8) that summarize actions for resolving model composition conflict at several levels based on some previous research [25] [29].

Operation	Description	Conflict Type
Class-Level		
Reappoint Class	change the name of the class and update the dictionary renaming	Lexical and syntactical
Displace Class	Remove the current class and move all its properties to another class.	Lexical and syntactical
Take out Class	Move the property from the current class to a new one.	Lexical and syntactical

Table 5 Model Composition Conflict Resolution at Class Level

Relationship-Level		
Take out Hierarchy	During an inheritance hierarchy, we add a new class to a subclass.	Structural
Take out Subclass	Move the property from the current subclass to a new one.	Structural
Take out Superclass	Move the property from the current superclass to a new one.	Structural
Crash Hierarchy	Delete a class from an inheritance hierarchy.	Structural

Table 6 Model Composition Conflict Resolution at Method Level

Method-Level		
Step Down Method	Displace a method from a subclass, in order to put it in a class that require it more.	Structural
Step Up Method	Displace a method from a class to a superclass.	Structural
Rename Method	change the name of the method and update the dictionary renaming	Lexical and syntactical
Reappoint Method	change the name and the accessibility of the class and update the dictionary renaming	Syntactical
Raise Method	copy the same method in several classes	Syntactical
Displace Method	Remove the method from the current class and move all its features to another class.	Syntactical

Table 7 Model Composition Conflict Resolution at Method Level

Field-Level		
Step Down Field	Displace a field from a subclass, in order to put it in a class that require it more.	Structural
Step Up Field	Displace a field from a class to a superclass.	Structural
Displace Field	Remove the field from the current class and move all its features to another class.	Structural
Reappoint Field	change the name of the field and update the dictionary renaming	Lexical and syntactical
Decline Field	Remove the field from the current class and move all its features to another class.	Syntactical
Raise Field	copy the same field in several classes	Syntactic
Encapsulate Field	Increase accessibility to a field through the creation of getter and setter.	Syntactical

Table 8 Model Composition Conflict Resolution at Field Level

IV.5. Conclusion

The model composition is the central concept in model driven architecture for maximizing return on investment, dealing with complexity and maintainability. But still "complete implementation of model composition capabilities" are not supported by model driven architecture. This chapter discuss

abilities on adopting a new methodology presented in the form of a conceptual prototype to automatically compose models defined in terms of class diagrams in order to build a global view of the system under construction.

We have presented the progress process of an environment for the formal implementation of different MDA aspects, using model composition and based on the two hemisphere model driven approach introduced in [NKA+15]. For an effective implementation of MDA, several features should be supported and integrated into appropriate tools starting from CIM to PSM.

Our conceptual prototype aims to automate all the MDA process by focusing on Model composition paradigm as a crucial activity. The proposed process is applied on two hemisphere model driven approach, which is an approach that aims to automate the process of class diagram development from correct and precise two hemisphere and enables knowledge representation in a form understandable for both business users and system analyst.

The central hypothesis of this approach is to apply many transformations in the framework of MDA, where the source model is defined in terms of a business process model, associated with a concept model, and the target model is defined in terms of class diagram.

In this chapter we tried to investigate the possibility to continue from the point which the two hemisphere model finish by using model composition to compose produced models in order to automate the whole MDA process and help in mitigating ever-growing complexity of modern software system.

The models are small enough and developed by a single or a couple of designers, that why they should be composed manually. However, in most cases, the models are too large to be composed manually and it's necessary to develop an automatic composition method to ensure that all the elements in the model are handled.

Indeed, our conceptual prototype can be used to build the model for a large system, where the modelers identify different class diagram. It models each piece separately to deal with complexity. Once all these models have been correctly built in isolation, it is necessary to compose them, however four main ideas for composition are identified:

-Better understand the interactions between the elements to compose: Model comparison.

-Match equivalent: weaving.

-Analyze interactions to identify conflicts and undesirable emergent behaviors: Repository for conflicts management.

-Check the global consistency of the system's model.

The model composition conflict presented in a form of repository that facilitates the customization of model composition.

This repository can form the basis for the development of tools to support the model composition process from the comparison phase to the verification phase. The repository also provides a common vocabulary for describing composition conflict actions. Illustrated examples demonstrate the use of each action.

Among the main challenges of the present chapter is to contribute to the understanding of composition conflicts, in particular within the scope of structural composition. To this extent we propose and illustrate a systematic approach to analyze such composition conflicts in a precise and

concrete manner. We propose several actions to express conflict detection and resolution rules. These actions have been introduced to deliver a precise explanation why and when some forms of composition cause a conflict, and to ensure that the categories are not overlapping. Also, the precise formulation makes it possible to perform the conflict detection fully automatically, for example, as a separate module which communicate in real time with a composition framework or consistency analyser.

The actions defined by our repository are expressive in the sense that they can specify common composition conflict actions such as renaming, replacing models, and at the same time they can be used to specify creation and removal of model elements, making it possible to significantly alter how models are composed.

Empirical evaluation is needed to validate the repository in real world design settings. Specifically the amount of effort required to specify the kinds of resolution that are required in real world designs needs to be empirically evaluated; the development of a tractable method of identifying conflicts in the composed model needs to be investigated; and the currently defined repository needs to be evaluated for its ability to support the kinds of composition conflict that actually occur.

This evaluation could result in the specification of some common detection conflict strategies to manage the complexity of specifying compositions and could be an area of future research. We are also exploring how to express the applicability and consequences of using a generic repository in terms that it can interact in any step of the composition process.

CHAPTER V. DEVELOPMENT AND EXPERIMENTATION OF A WEBSERVICES PLUGIN BASED ON MODEL COMPOSITION

V.1. Introduction

Model Composition is a powerful approach that requires efficient implementation to be useful in practice. To prove the feasibility of the concepts and validate the approach introduced in the previous chapters, we implement an operational prototype in the form of a plugin, which embodies the principle of model composition based on webservices. This chapter presents the essence of the plugin through several user stories, describes the outlook plugin, its general architecture, the implementation of this plugin, and finally the web services module and how the communication of this plugin with several applications.

For the implementation of the plugin, we decided to use several programming languages:

C#: we choose C# in order to implement the plugin installer, because this programming language can be used to write almost any type of software, including console apps, mobile apps, Windows apps, websites and blockchains. PHP to implement the web services.

PHP 7: we choose PHP in order to implement our web services, one of the most significant strengths of PHP is that it supports a lot of databases. Writing a web page using a database becomes awfully simple, using one of the database specific extensions (ie, for mysql), or using an abstraction class, or a connection to any database supporting the standard connection. Other databases can use the cURL extension or sockets like CouchDB. PHP supports many protocols like LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (on Windows), also supports Java objects, and uses them transparently as built-in objects.

Angular 6: we choose this framework in order to implement our user interfaces, is a powerfull JavaScript framework for building web applications and apps in JavaScript, html, and TypeScript, which is a superset of JavaScript. Angular provides built-in features for animation, http service, and materials which in turn has features such as auto-complete, navigation, toolbar, menus, etc. The code is written in TypeScript, which compiles to JavaScript and displays the same in the browser.

V.2. Plugin essence

V.2.1. Plugin for Outlook

Microsoft Outlook is a personal information manager software system from Microsoft, available as a part of the Microsoft Office suite. Though primarily an email client, Outlook also includes such functions as calendaring, task managing, contact managing, note-taking, journal logging, and web browsing.

Individuals can use Outlook as a stand-alone application; organizations can deploy it as multi-user software (through Microsoft Exchange Server or SharePoint) for such shared functions as mailboxes, calendars, folders, data aggregation (i.e., SharePoint lists), and appointment scheduling. Microsoft has released apps for most mobile platforms, including iOS and Android.

In addition, Windows Phone devices can synchronize almost all Outlook data to Outlook Mobile. Using Microsoft Visual Studio, developers can also build their own custom software that works with Outlook and Office components.

In this context we have designed a plugin for Outlook, this plugin is composed of several modules, an installer that allows to install the plugin on the machine, a web services that allow Outlook to communicate with other applications in order to upload user activities (conferences, absences, meeting) in the target applications and allow permanent communication, finally a web interface that represents the progress and behavior of the plugin in real time, as well as the generation of crash reports.

Our plugin is configurable, it allows to communicate with any web application through a configuration page, it can be installed on several types of machine and several operating systems.

V.2.2. Backlog and user stories

Through several brainstorming sessions and online forms, we have identified a set of needs to implement through our plugin and which allow us to meet a real need in the context of model composition, in order to meet the expectations of several users.

The tool must be able to synchronize the following activities:

- Conference (SKYPE or LYNC)
- Remote Work
- Miscellaneous Absences.
- Training
- Courses
- Meeting (on SITE)

Outlook-Plugin synchronization is done through categories of the same name as the target application activities, is done by setting the plugin, which ensures great flexibility and reusability.

Example: entering an Outlook appointment in the "Conference" category will place it in the other application in the "Conference" activity. In Outlook, a “Skype meeting” will automatically

categorize itself as “Conference call”. If the category is not entered, the Outlook appointment will automatically categorize itself as "Meeting". Entering a new category must replace the previous one. A deleted category will be recreated. A private meeting will not be taken into account by the tool.

The user must be able to adjust in the parameters section:

- Exit the synchronization of keywords or categories.
- Synchronization range (number of days before and after).
- The synchronization application should:
- Generate a log of errors present on the workstation
- Report anomalies to the server via a WS.

It is necessary to give the user a view of the synchronization process:

- The synchronization button must give more visibility,
- A short user synchronization log must be present.
- In the event of a network problem, a clear message must be displayed, and the module must be able to try synchronization again at a given rate (with a user message).
- The application must be able to install easily.
- The update should also be transparent to the user.

We present in this table the various points of the backlog which were carried out in this version. We identified these needs following several online forms in order to identify the real need from the user's point of view.

Category management
The plugin must provide the user a predefined list of categories with the form *category_name*.
Whether automatically (plugin) or manually (user), a single category will imperatively be associated with a meeting.
By default, a meeting will automatically be tagged "Meeting".
If the plugin detects that the meeting is of type "Skype", it will be automatically tagged "Conference call".
The user can change the target application category automatically assigned by the plugin at any time.
The user can assign many categories to a meeting (in addition to a single category).
Meeting Planning and Synchronization
A “private” meeting is ignored by the plugin.
Immediate synchronization should be performed if the meeting dates change.
Immediate synchronization must be performed if the associated category changes.
Immediate synchronization must be performed in the event of an update of the category blacklist.

Immediate synchronization must be carried out if the synchronization range changes.
Immediate synchronization should be performed when Outlook starts up.
Set up a synchronization indicator (OK / KO) in the toolbar: - Synchronization icon OK (indicate when hovering the sync time) - KO synchronization icon (indicate the pattern when hovering)
Set up automatic resynchronization in the event of an error as follows: - Add a "Timer" type icon indicating the time before a new automatic resynchronization attempt if the detected error is different from a problem of network connectivity on the user station side. - After 3 unsuccessful attempts, alert the user (popup) that the synchronization is not working and that he must wait. Continue to retry automatically. Warn again (popup) if the sync is operational again.
Plugin configuration and saving information
Provide a button in the toolbar to launch a plugin configuration window specific to the logged in user.
The user must be able to edit a blacklist of all their personal categories that they do not wish to synchronize with the outlook plugin.
The user must be able to choose a synchronization range (for technical reasons) defined as follows: - 1 field for the number of days to synchronize before the current date (min 1, max 30) - 1 field for the number of days to synchronize after the current date (min 1, max 365)
The user configuration must be backed up so that it can be restored to the same on another computer to which he has connected.
The user must be able to deactivate the plugin directly from the configuration window.
Error management and reporting
Setting up a webservice (temporarily hosted on Propulse) to collect error logs that will be sent by all plugins in the computer park. The details of the logs remain to be defined.
The plugin must be able to archive a "memory dump" on the user workstation in the event of a system error. The dump must be sent to the webservice to allow a complete analysis.
The plugin should automatically deactivate (from a user point of view) when an internal error occurs. Deactivation closes the configuration window (if open) and deactivates the associated button in the toolbar.
Implementation of a log web service specific to installations, deletions and updates of installers and plugins.
What about the benefit of setting up user action logging on the user workstation?

Implementation of a local log
Installation and Updates
The installation must be deployed using a packager (type .exe or equivalent).
The installer must be able to detect the user environment and the presence of an older version of the plugin.
When launching the installer, it checks whether it needs to be updated with regard to the user configuration and installs itself if necessary.
At the end of the installer update or if it is already up to date
When launching the installer, the user will be able to choose between: - An update of the plugin (if a new version is detected). - Reinstallation of the plugin in a compatible version and by default the pread recently. - Removal of the plugin (if installed).
When launching the installer, if no version of the outlook plugin is detected, it will automatically install it without asking the user for confirmation.
Implementation of a versioning webservice for the installer as well as for the plugin.
Taking into account the versions of Windows + version of Outlook + architecture 32/64.
Memorize on the user workstation the choice he made regarding the version of the plugin installed in order to avoid asking him to update his plugin each time Outlook is opened. After 7 days, the update request through the installer will be effective again.

V.3. OutlookPlugin Architecture

Outlook is the go-to tool in most companies for entering events and meetings. In order to enter meetings in other web applications, it is necessary to copy Outlook tasks.

Based on this need, we had the idea to implement an Outlook plugin that will be able to communicate with several applications with a simple configuration in order to simplify the process and avoid double entry.

The solution is structured as follows

AddInManager: library responsible for reading and writing registry keys for managing Outlook add-ons.

HttpClient: extension of System.Net.Http.HttpClient adding detailed logs, performing automatic attempts in case of connection failure, as well as HTTP Basic authentication.

Install: the OutlookPGP installer.

OutlookPGP: Outlook add-on.

PluginClient: overlay of HttpClient used to consume the PGP WebService.

SharedConstants: constants shared by all the projects in the solution.

WebServiceClient: overlay of HttpClient used to consume the OutlookPGP WebService.

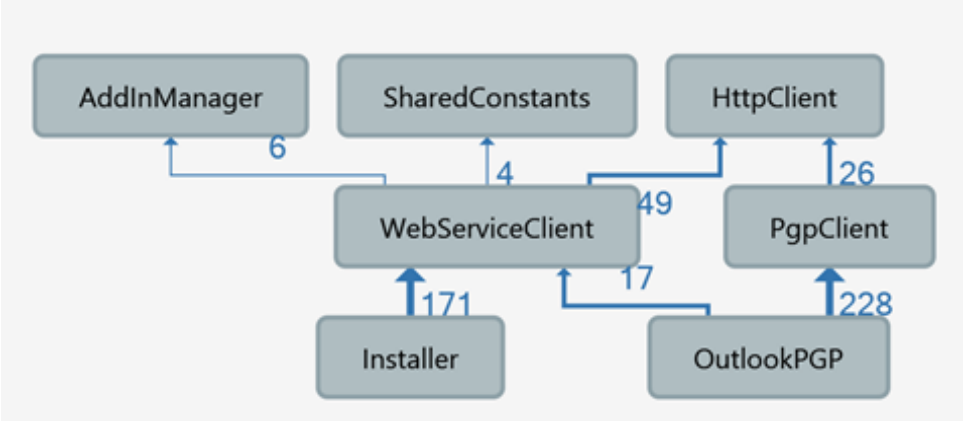


Figure 40 Solution structure

OutlookPlugin project dependencies. The numbers in figure 40 indicate the number of references detected (methods, classes etc.).

The whole project supports two main configurations:

Debug: for the developpement version.

Release: for the production version.

A project compiled in Debug mode will use developpement enviornomment, OutlookPGP-WebService developpement and use data from devellepement database.

A project compiled in Release mode will be more optimized and will use production environnement, OutlookPGP-WebService production and use the data from production database.

AddInManager: The library can be used to know the list of installed complementary modules, to install new ones and to uninstall some. The entry point of the library is the static AddInManager class. The library uses the Windows registry keys to perform the installation and uninstallation. (See Figure 41)

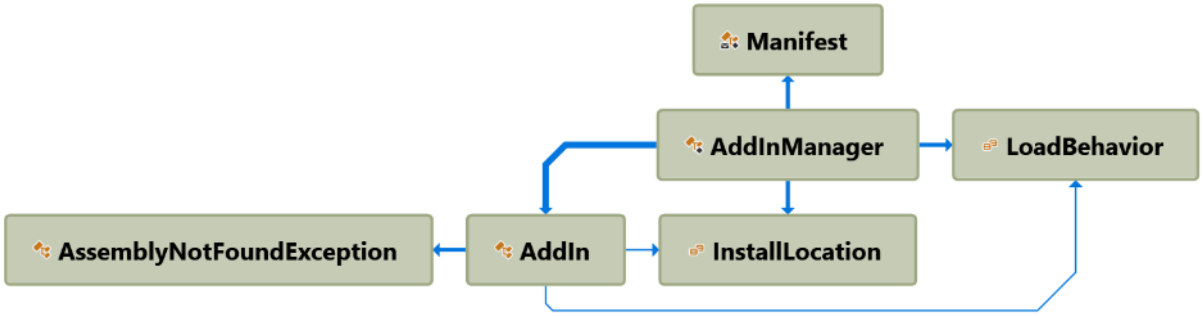


Figure 41 AddInManager dependencies graphic

HttpClient: The library can be used by other projects as a REST client. Its entry point is the RestWebClient class.

The code handling the general behavior of the class is located in its private SendWithExceptionsAsync method. The class adds the following features:

- Management of QueryParams.
- Throwing exceptions when receiving server error codes.
- Detailed debugging and logs.
- Multiple tests in case of network errors (configurable).
- Validation or inhibition of SSL certificates.

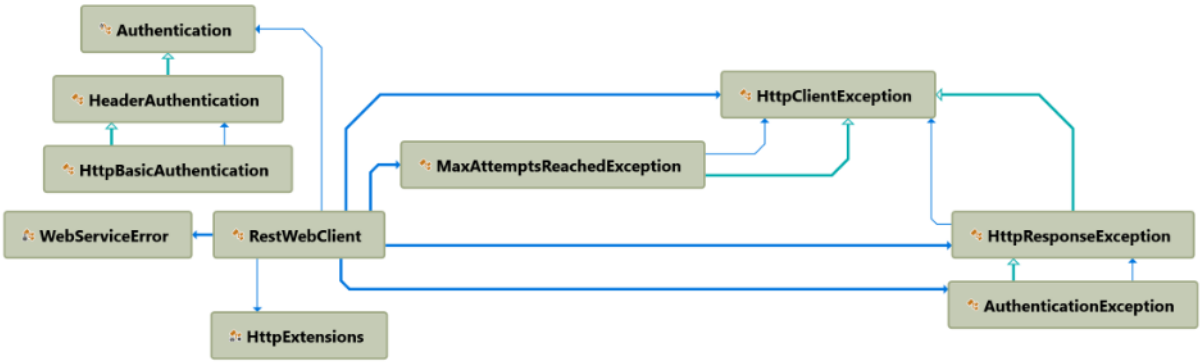


Figure 42 HttpClient dependecies graphic

V.3.1. Installer

The installer was designed to work in command line and graphical interface.

The most complex example is updating the installer:

```
// This step is an empty step marking the end of the process.
// It is used as a "goto" (or "favorite"), allowing to be the target of step 3.
var s10 = new NoOpStep ();

// The first step is to get the list of installer files.
// The first dependency observed is that of the OutlookPGP-WebService API.
var s1 = new InstallerFileFetchStep (VersionApiClient);

// Step 8 is to clean up unused files.
// This step simply makes a difference between the files present on the server and those
//present on the client.
// This step needs step 1 which knows the files installed on the server. She also needs to
//know where the installer is installed.
// Delete the folder and then copy the files is not an option here because if the copy operation
//fails, you can't go back to recover the deleted files.
// If, on the other hand, the copy fails we keep the original files.
// Then we delete the extra files, and if this operation fails, it is not "serious".
var s8 = new UnusedFilesCleanupStep (s1, Constants.InstallerInstallLocation);
```

```
// The file comparison step allows you to check the checksum of each file.
// If a file is different on the server, we download it.
// Otherwise, we leave it as is.
// The dependency on step 1 comes from the fact that the server files must serve as a basis
//for comparison that only step 1 knows.
var s3 = new ExistingFileComparisonFileFetchStep (s1, Constants.InstallerInstallLocation,
new CustomStepBehavior (s10));

// Step of creating a temporary directory into which the new installer will be downloaded.
var s4 = new DirectoryCreationStep ();

// Step of deleting the temporary directory (we mark the deletion immediately to prevent
//the failure of a step from preventing cleaning).
// If we don't do this, we run the risk that a user whose installation failed every day for
//1 month ends up with a PC full of installers.
var s5 = new DirectoryRemovalStep (() => s4.Directory, true);

// Step of downloading the files.
// The lambda (second argument) is used to select a destination directory.
// Without lambda, the file from step 4 would be immediately fetched even before the step was executed,
//and we would have a null.
var s6 = new FileDownloadStep (s1, () => s4.Directory);
```

```
var s6 = new FileDownloadStep (s1, () => s4.Directory);

// We restart the newly downloaded installer from the temporary directory.
var s7 = new InstallerRestartStep (() => Path.Combine (s4.Directory, $ "{Constants.InstallerAssemblyName} .exe"),
args);

var s9 = new DirectoryCopyStep (Constants.AssemblyDirectory, Constants.InstallerInstallLocation);

// We finally create the process with all the steps, in the right order.
return new MultiStepProcess (s1, s2, s3, s4, s5, s6, s7, s8, s9, s10);
```

Figure 43 Commented code snippet from ProcessFactory.cs

With this method, instead of modifying a large chunk of 1000 lines of code with several conditions and nested loops, we have steps that are all interchangeable, weakly dependent and more easily debuggable.

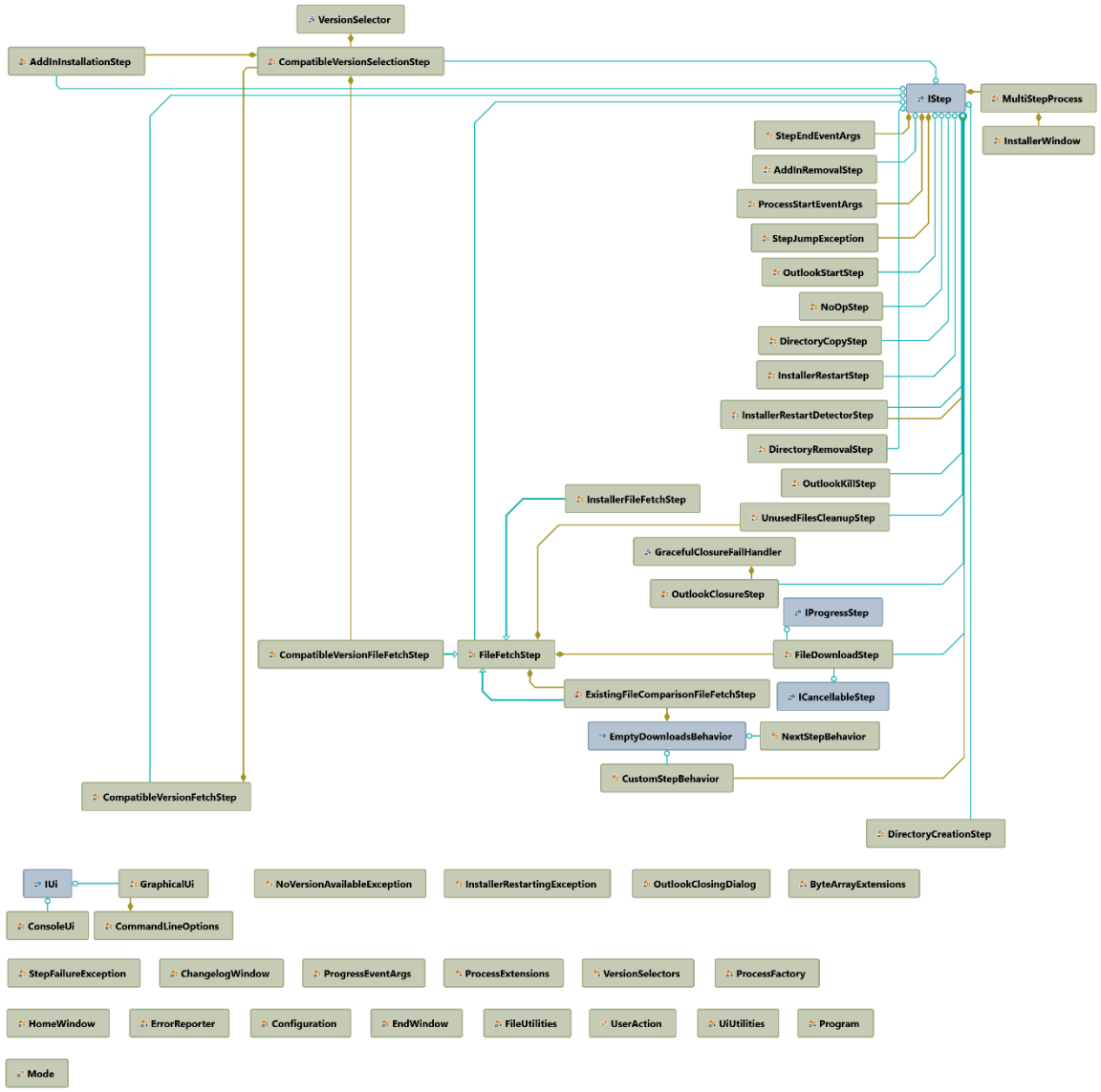


Figure 44 Strong dependencies (constructor arguments) and inheritance of project classes.

Command line use

The installer can be used from the command line if the --no-gui argument is specified. Using this mode results in a more verbose rendering and additional details on the flow of certain steps. In the event of an error, the call stack will still be displayed, even if no crash report is sent.

Here is the list of available commands:

- opinstall.exe --help: displays command line help.
- opinstall.exe AutoUpdate --no-gui: Performs an update of the installer and add-on installed in% USERPROFILE% \ .. \ Public \ applications \ OutlookPGP.
- opinstall.exe SelfInstall --no-gui: Downloads and installs the installer without the add-on.
- opinstall.exe SelfUninstall --no-gui: Uninstall the installer without uninstalling the add-on.

- opinstall.exe AddInInstall --no-gui: Downloads and installs the add-on without installing the installer.
- opinstall.exe AddInInstall <version> [--keep] --no-gui: Downloads and installs the given add-on version.
- Adding --keep allows you to force the installer to keep a version of OutlookPGP until an update follows.
- For example, if the latest version is 1.0.0.0 and the opinstall.exe AddInInstall 0.9.0.0 --keep --no-gui command is run, version 0.9.0.0 will be kept until a 1.1.0.0 (or other) is output.
- opinstall.exe AddInUninstall --no-gui: uninstalls the add-on without uninstalling the installer.
- opinstall.exe KillOutlook --no-gui: kills the Outlook processes found (avoids going through the task manager).

The configuration principle is defined in the outlookPlugin.csproj file in order to allow great compatibility of this plugin with several applications, several operating systems.

We have even set up a user interface that allows you to do this without touching the code.

The figure 45 defines project-level properties. AssemblyName, Name of the output assembly, Configuration, Specifies a default value for debug. OutputType, Must be "Library" for VSTO, Platform, Specifies what CPU the output of this project can run on, NoStandardLibraries, Set to "false" for VSTO, RootNamespace

In C#, this specifies the namespace given to new files. In VB, all objects are wrapped in this namespace at runtime.

```
<PropertyGroup>
  <ProjectTypeGuids>{BAA0C2D2-18E2-41B9-852F-F413020CAA33};{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}</ProjectTypeGuids>
  <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
  <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
  <ProjectGuid>{1A7B9AB0-631D-466A-B54A-5D938589C8C4}</ProjectGuid>
  <OutputType>Library</OutputType>
  <NoStandardLibraries>false</NoStandardLibraries>
  <RootNamespace>OutlookPGP</RootNamespace>
  <AssemblyName>opaddin</AssemblyName>
  <TargetFrameworkVersion>v4.6</TargetFrameworkVersion>
  <DefineConstants>VSTO40</DefineConstants>
  <TargetFrameworkProfile />
  <IsWebBootstrapper>False</IsWebBootstrapper>
  <BootstrapperEnabled>true</BootstrapperEnabled>
  <PublishUrl>publish\</PublishUrl>
  <InstallUrl />
  <TargetCulture>fr</TargetCulture>
  <ApplicationVersion>1.0.3.0</ApplicationVersion>
  <AutoIncrementApplicationRevision>true</AutoIncrementApplicationRevision>
  <UpdateEnabled>true</UpdateEnabled>
  <UpdateInterval>7</UpdateInterval>
  <UpdateIntervalUnits>days</UpdateIntervalUnits>
  <ProductName>OutlookPlugin</ProductName>
  <PublisherName />
  <SupportUrl />
  <FriendlyName>OutlookPlugin</FriendlyName>
  <OfficeApplicationDescription />
  <LoadBehavior>3</LoadBehavior>
</PropertyGroup>
```

Figure 45 code snippet from the plugin configuration file representing the project-level propreties

The figure 55 presents also a section from properties that are set when the "Debug" configuration is selected.

DebugSymbols (If "true", create symbols (.pdb). If "false", do not create symbols)

DefineConstants, Constants defined for the pre-processor, EnableUnmanagedDebugging (If "true", starting the debugger will attach both managed and unmanaged debuggers), Optimize (If "true", optimize the build output. If "false", do not optimize), OutputPath, Output path of project relative to the project file, WarningLevel, Warning level for the compiler.

```
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|x64' And '$(Ide)' == 'VisualStudioCode' ">
  <PlatformTarget>x64</PlatformTarget>
  <DebugSymbols>true</DebugSymbols>
  <DebugType>portable</DebugType>
  <Optimize>false</Optimize>
  <OutputPath>bin\Debug\</OutputPath>
  <DefineConstants>$(DefineConstants);DEBUG;TRACE</DefineConstants>
  <ErrorReport>prompt</ErrorReport>
  <WarningLevel>4</WarningLevel>
  <LangVersion>7.2</LangVersion>
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
  <DebugSymbols>true</DebugSymbols>
  <DebugType>full</DebugType>
  <Optimize>false</Optimize>
  <OutputPath>bin\Debug\</OutputPath>
  <EnableUnmanagedDebugging>false</EnableUnmanagedDebugging>
  <DefineConstants>$(DefineConstants);DEBUG;TRACE</DefineConstants>
  <WarningLevel>4</WarningLevel>
  <LangVersion>7.2</LangVersion>
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
  <DebugType>pdbonly</DebugType>
  <Optimize>true</Optimize>
  <OutputPath>bin\Release\</OutputPath>
  <EnableUnmanagedDebugging>false</EnableUnmanagedDebugging>
  <DefineConstants>$(DefineConstants);TRACE</DefineConstants>
  <WarningLevel>4</WarningLevel>
  <LangVersion>7.2</LangVersion>
</PropertyGroup>
```

Figure 46 code snippet from the plugin configuration file representing debug propreties

Our project in visual studio looks like in the figure 47, the installer and the web services that provide clients with target applications.

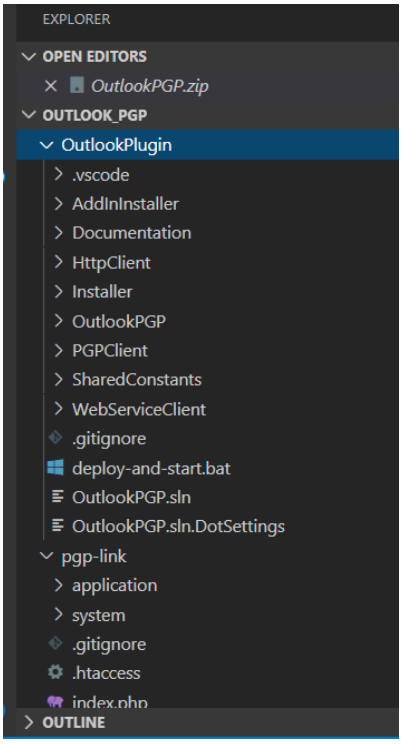


Figure 47 the project structure

Installation: installing the plugin is easy and guided by windows. It also allows us to uninstall the plugin if it already exists or to force Outlook to close. (See Figure 48)

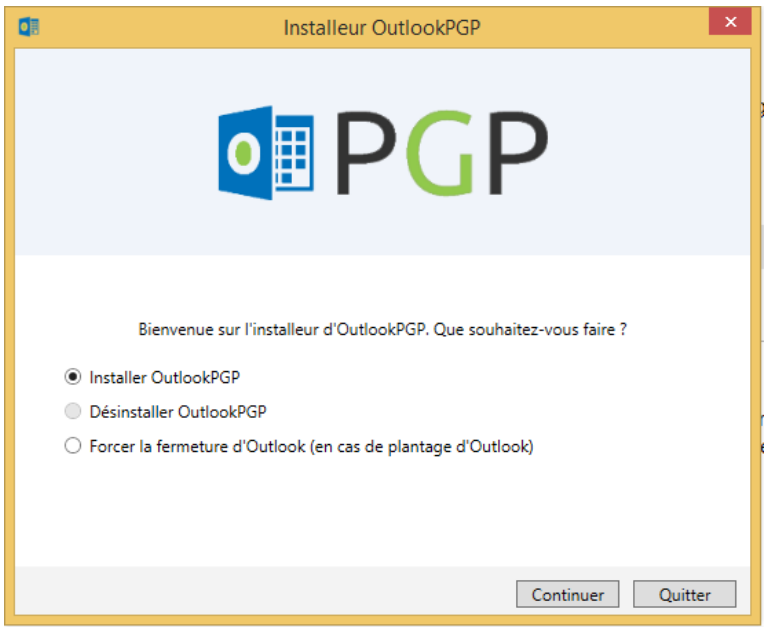


Figure 48 Installation Window

Following this installation a module is added to outlook, in this module we find the possibility of synchronizing the Outlook calendar with the target application and the configuration of the module. (See Figure 49)

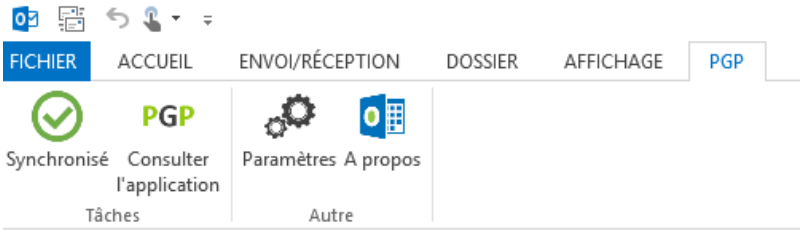


Figure 49 Outlook module added after installing the plugin

From this configuration, we have the possibility to exclude meetings, choose the synchronization period and activate or deactivate synchronization. (See Figure 50)

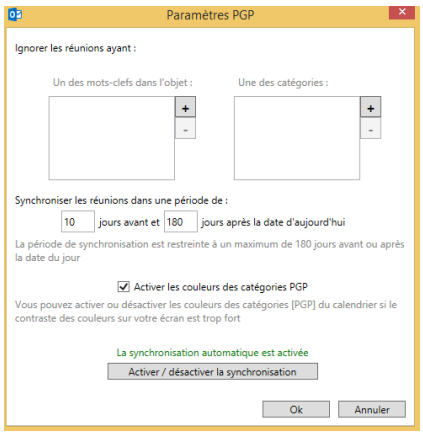


Figure 50 Configuration Windows

In addition to this, the types of meetings and activities defined in the plugin are added to outlook in order to allow the user to classify activities. We have prefixed them with [PGP] (See Figure 51)

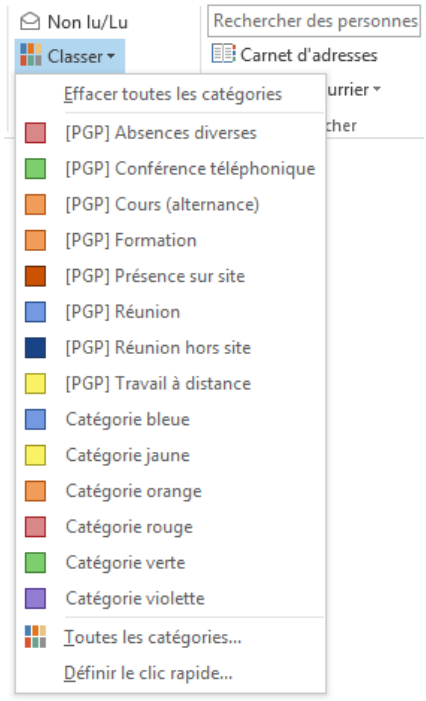


Figure 51 Plugin activities injected in Outlook

V.3.2. User Interface

An interface is represented by the IUi interface. All operation has been broken down into a process. We find for example the installation process of the add-on, and the update process. A process is represented by the MultiStepProcess class. The processes are broken down into stages. Each step represents a non-breaking operation (and potentially dependent on another).

A step is represented by the IStep interface. Each concrete class can also implement IProgressStep (which makes it possible to communicate a percentage of progress to the interface) or even ICancellableStep (which makes it possible to make the step cancelable). The ProcessFactory class was created in order to create all the triggerable processes within the installer. The dependencies between the stages are formulated in several different ways.

V.3.3. Plugin deployment

In order to deploy the project, we have tto follow some steps:

- Zip the contents of the build (Installer \ bin \ <Debug or Release> \ *)
- Send the archive via the OutlookPGP-WebService graphical interface
- In "Deployment" menu, "Installer" tab. we "Send file" and the deployment is now effective.

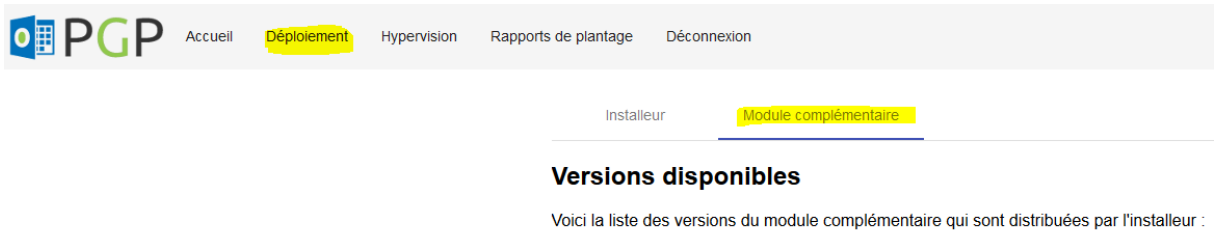


Figure 52 process to deploy the plugin

- The plugin is initialized using dependency injection.
- Each service is injected into scopes (representing the lifetime of a service).
- By default there is an "App" scope which lasts the entire execution time of the module, defined in the ThisAddIn class. All services must be registered there.

CrashReportService: is responsible for reporting all errors via the WebService.

OutlookAppointmentService: is responsible for providing the list of synchronizable meetings.

OutlookAccountService: is responsible for providing the EOutlook account associated with the application.

OutlookCategoryService: is responsible for providing the list of categories [PGP], creating them and updating the colors.

OutlookTaggingService: service in charge of tagging (pasting categories) each synchronizable meeting.

SynchronizationService: service in charge of performing synchronization.

UpdateService: performs updates and launches the installer.

NotificationService: displays graphical notifications to the user.

UsageReportService: reports usage to the WebService.

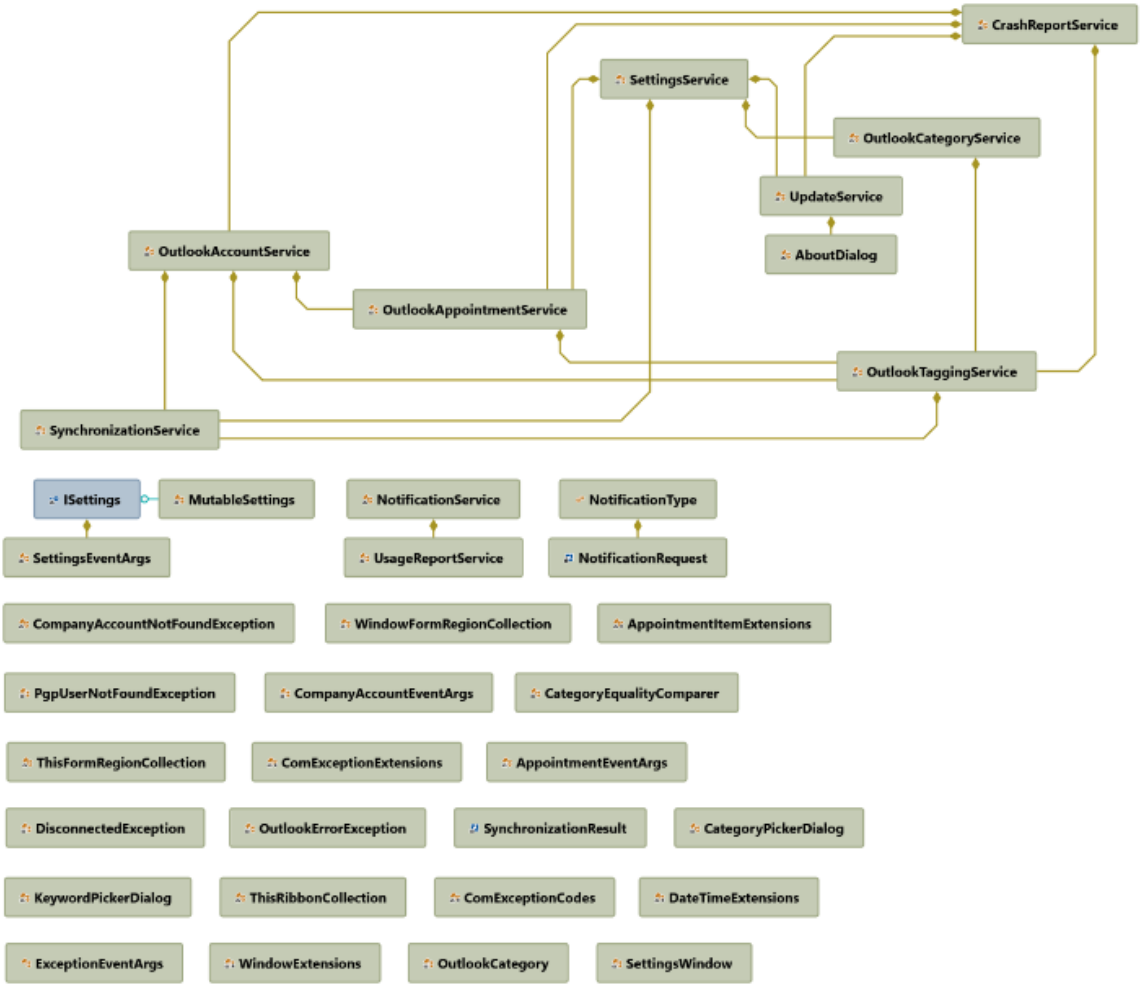


Figure 53 Installer dependencies Graphic

Before deploying, it is necessary to update the version of OutlookPGP. (See Figure 54)

Specifying a version is essential if we want users to be able to go back to a version in the event of an irremediable difficulty.

We should modify the version on OutlookPGP as follows:

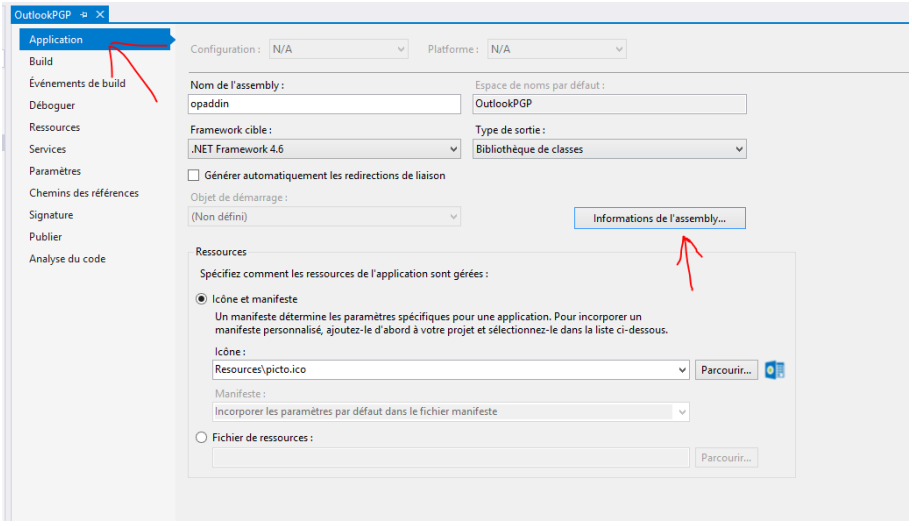


Figure 54 Updating plugin version

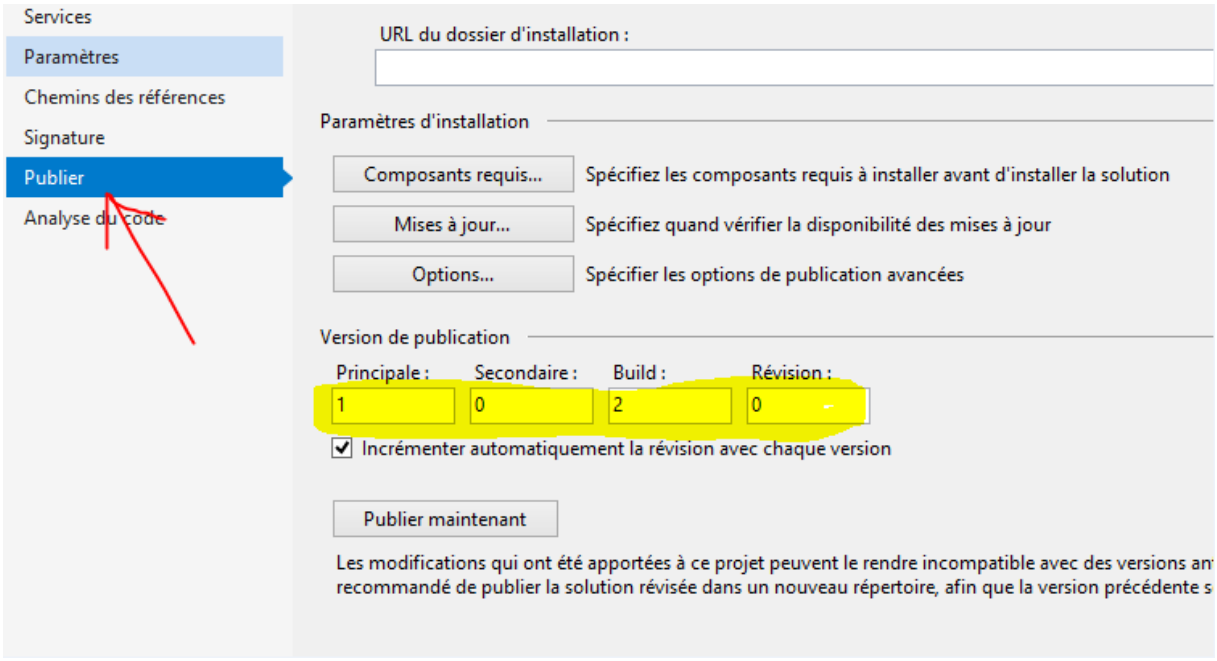


Figure 55 provide the plugin version

On Visual Studio Code: we edit the .csproj file manually, by modifying the ApplicationVersion tags, edit the AssemblyInfo.cs file at the AssemblyVersion and AssemblyFileVersion attributes, then build after this manipulation.

The list of production versions is displayed. Zip the OutlookPGP compilation output \ bin \ <Debug or Release> \ *.

To load a version, we go to OutlookPGP-WebService, "Add-on" tab.

Figure 56 Upload version screen 1

The archive created should be upload using the "Choose .zip archive" button. Then sen it in order to start the deployment process.

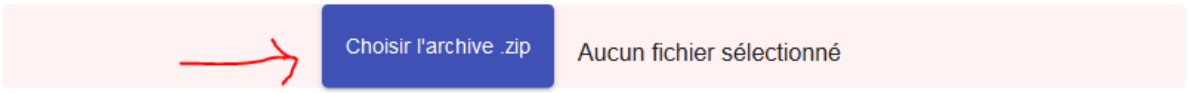


Figure 57 Upload version screen 2

From there, there are two possibilities:

Either the chosen version of OutlookPlugin already exists - you must then choose whether to overwrite the current version (in which case the updated files will be re-deployed to clients as and when) or whether to add a distribution to the given version;

Either the chosen plugin Outlook version does not exist, or a distribution creation modal is displayed.

V.4. OutlookPlugin-WebService

We have developed several web services that allow us to communicate with target applications and send them the content of outlook activities. This communication is done through the identification of the parameters of this application in a constants.cs file, REST API for managing add-on versions and hosting installer files.

The project is currently hosted on local, in the ~ / www / outlookplugin-webservice folder.

Both environments, developement and production are covered.

V.4.1. Web services structure

The web service module is structured as follows:

- `backend/` :
  - `logs/` : set of application logs (can be configured in src / settings.php at the level of the logger key).
  - `public/` : folder containing the "exposed" part of the framework.
    - `angular/` : folder containing the compiled front-end.
    - `distribution/` : folder containing the compiled files distributed.
      - `installer/` : folder containing all the compiled installer files.
      - `plugin/` : folder containing folders for each version / each distribution of OutlookPlugin.
        Folders are named according to a convention: : `<version_outlookplugin> _flavor_ <distribution_id>.` .
  - `src/` :
    - `Commands/` : Symfony commands allowing to use the command line on Slim.
    - `Controllers/` : API controllers.
    - `Misc/` : folder containing several utility classes.
    - `Models/` : package grouping together all DTOs (or "models" under Symfony).
    - `Services/` : package grouping together all the services (repositories under Symfony).
    - `commands.php` : list of commands with their names.
    - `database.sql` : script for creating the database.
      Note : do not forget to update it in the event of a schema change!
    - `dependencies.php` : list of dependencies to provide in dependency injection.
    - `middleware.php` : enabling middleware.
    - `routes.php` : file including all the routes with the roles allowing to restrict access to the routes.
    - `settings.php` : file containing all the application settings.
- `frontend/` : Angular application allowing deployment and management.

Figure 58 web services structure

We present some examples of web services implemented in order to answer the plugin backlog defined beforehand:

GET /plugin/versions/flavor/ {flavorId}/files: this service get the list of files for the given plugin version. (See Figure 59)

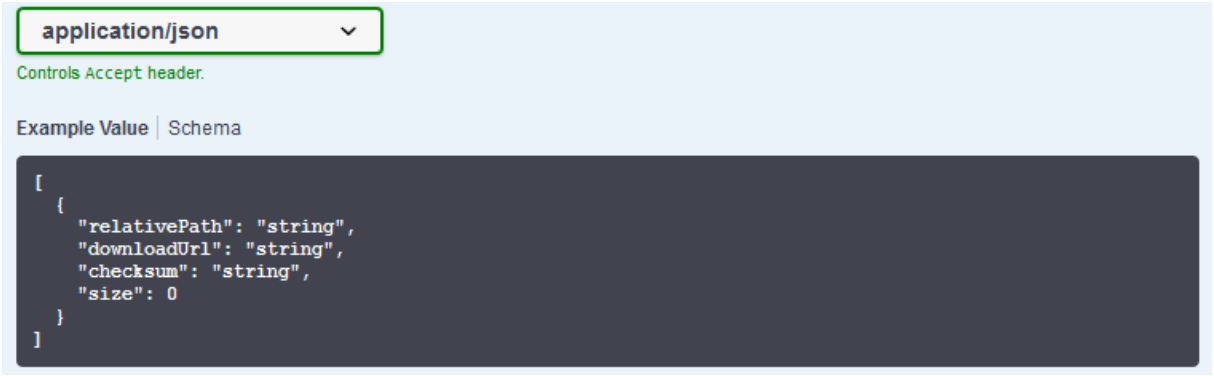


Figure 59 Get plugin versions

GET /plugin/usage-statistics: collects all usage statistics for all versions of OutlookPlugin. (See Figure 60)

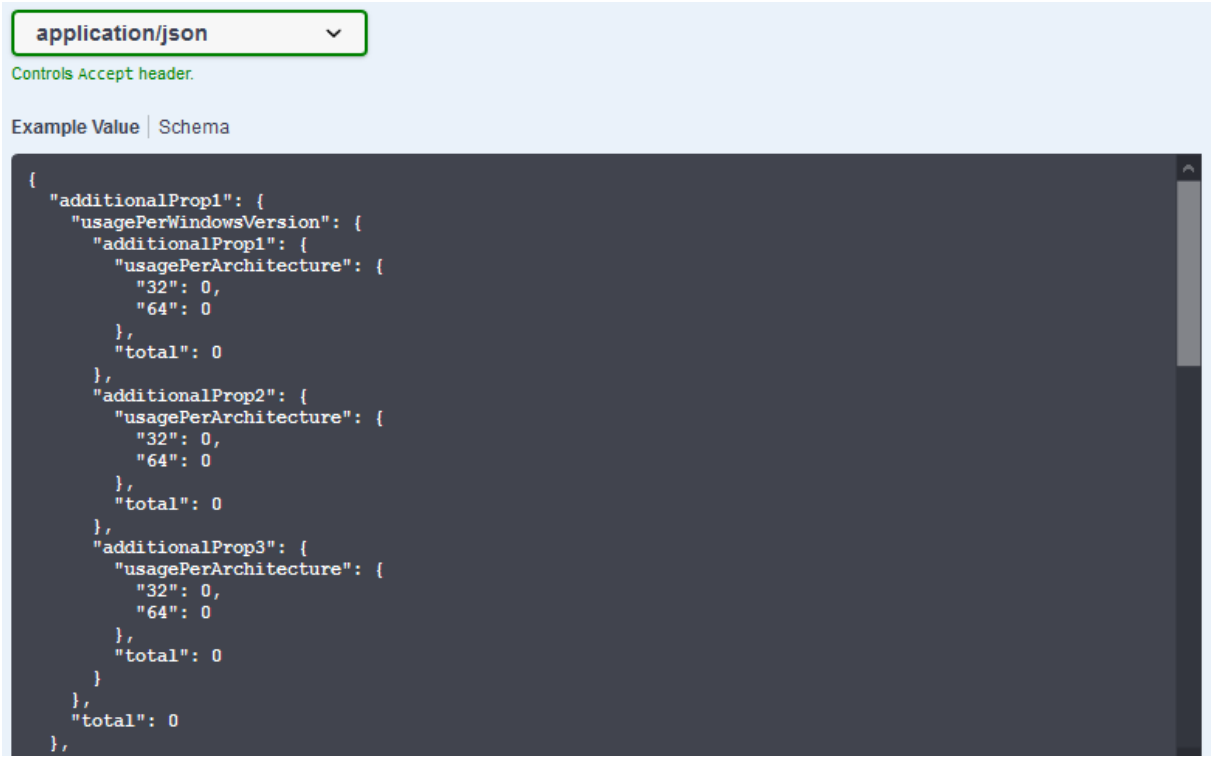


Figure 60 Get plugin usage statistics service

GET/plugin/usage-statistics/version/ {versionNumber}: Retrieves usage statistics for the specified OutlookPlugin version. (See figure 61)

Figure 61 Get plugin usage statistics by a version number service

GET /plugin/versions: retrieves the list of all installed versions. (See Figure 62)

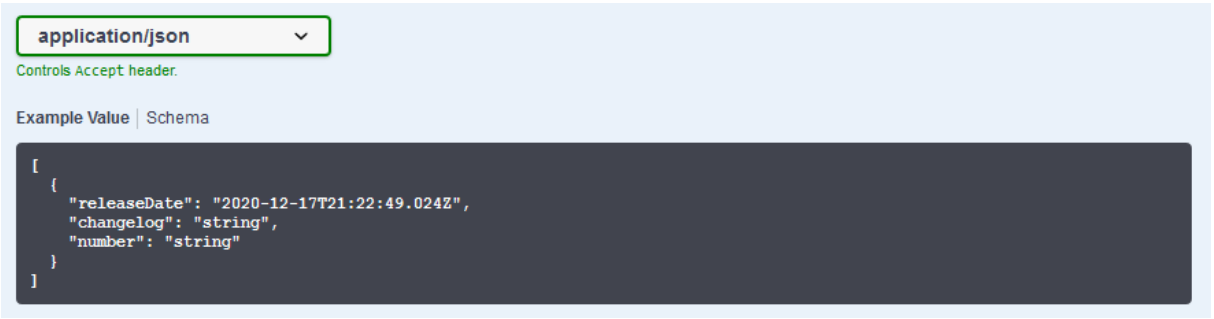


Figure 62 get versions service

V.4.2. Web services features

Graphic interface

The graphical interface can be accessed via the address https://pgp.appheb/outlookpgp-webservice/public/.

Command Line Interface (CLI)

The command line interface is used by going into the backend project folder (or directly outlookpgp-webservice in prod or in dev) and using the command as follows: php public / index.php <command> [arguments ...].

CRON jobs:

The WebService has a single scheduled task that runs every day at midnight. It helps to clean

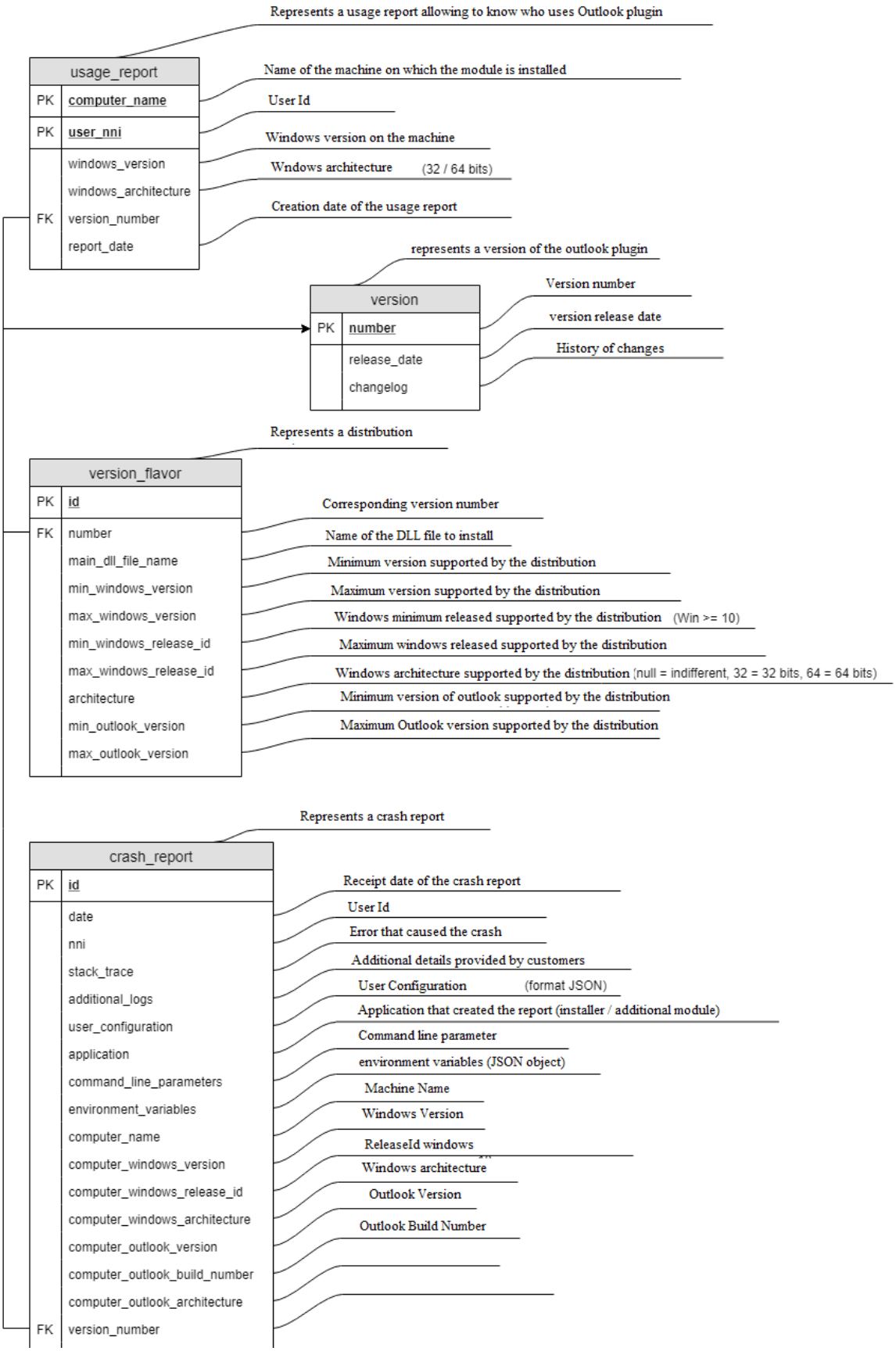
Authentication:

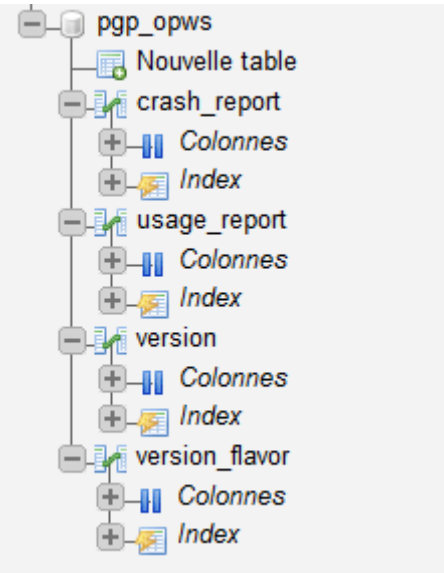
The WebService is protected by several roles and users.

By default, there is only one user per role, but it is possible to add more by modifying src / settings.php.

Passwords and user names are available in technical accounts.

Database:


Represents a usage report allowing to know who uses Outlook plugin
usage_report
PK computer_name
PK user_nni
windows_version
windows_architecture
FK version_number
report_date
Name of the machine on which the module is installed
User Id
Windows version on the machine
Wndows architecture (32 / 64 bits)
Creation date of the usage report
represents a version of the outlook plugin
version
PK number
release_date
changelog
Version number
version release date
History of changes
Represents a distribution
version_flavor
PK id
FK number
main_dll_file_name
min_windows_version
max_windows_version
min_windows_release_id
max_windows_release_id
architecture
min_outlook_version
max_outlook_version
Corresponding version number
Name of the DLL file to install
Minimum version supported by the distribution
Maximum version supported by the distribution
Windows minimum released supported by the distribution (Win >= 10)
Maximum windows released supported by the distribution
Windows architecture supported by the distribution (null = indifferent, 32 = 32 bits, 64 = 64 bits)
Minimum version of outlook supported by the distribution
Maximum Outlook version supported by the distribution
Represents a crash report
crash_report
PK id
date
nni
stack_trace
additional_logs
user_configuration
application
command_line_parameters
environment_variables
computer_name
computer_windows_version
computer_windows_release_id
computer_windows_architecture
computer_outlook_version
computer_outlook_build_number
computer_outlook_architecture
FK version_number
Receipt date of the crash report
User Id
Error that caused the crash
Additional details provided by customers
User Configuration (format JSON)
Application that created the report (installer / additional module)
Command line parameter
environment variables (JSON object)
Machine Name
Windows Version
ReleaseId windows
Windows architecture
Outlook Version
Outlook Build Number



Usage reports: usage reports are grouped by job name and id user. (See Figure 63)

#	Date de création	Nom du poste	NNI	Version d'OutlookPGP	Actions	
242	30/08/2019	ETW86Y7S	f55272	1.0.1.1	Consulter	Supprimer
243	02/09/2019	KFCE8A1Q	g63934	1.0.2.0	Consulter	Supprimer
244	02/09/2019	KFCE8A1Q	G63934	1.0.2.0	Consulter	Supprimer
245	03/09/2019	KQG28AYL	A63522	1.0.1.1	Consulter	Supprimer
246	03/09/2019	KYB28S0A	D26053	1.0.2.0	Consulter	Supprimer
247	04/09/2019	KFCE8A1Q	G63934	1.0.2.0	Consulter	Supprimer
248	04/09/2019	EPW96K4A	g63940	1.0.2.0	Consulter	Supprimer
249	05/09/2019	KYB28S0A	D26053	1.0.2.0	Consulter	Supprimer
250	06/09/2019	KYB28S0A	d26053	1.0.2.0	Consulter	Supprimer

Figure 63 examples of crashes reports

Versions: outlook plugin versions allow to store the history of changes and the release date of each change.

Distributions: outlook versions are not strings. These vrsion numbers must be associated with a distribution containing all the files to be deployed to the user. Each distribution may or may not be compatible with versions of outlook or windows. This allows a version of outlook plugin to have files that will vary depending on the architecture or the version of outlook.

The figure 64 shows the different versions deployed, the number of people who use each version

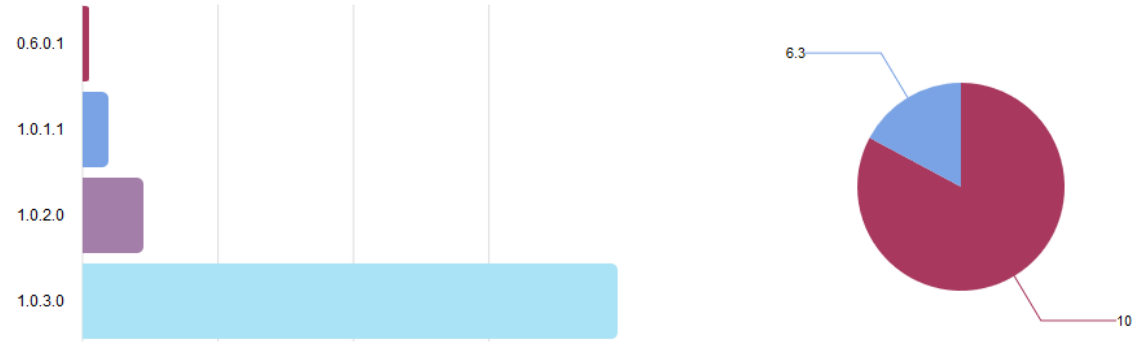


Figure 64 Plugin distribution

Crash report: each application (outlook plugin or installer) can issue crash reports. Each crash report contains as much information as possible that the customer could find.

To allow better monitoring of the plugin, we list the errors and crashes detected, store them in the database and display them in our graphical interface dedicated to the administrator. The figure 65 shows an example of a crash report with all the information that allows us to interpret the error.

Figure 65 Example of a crash report

Deployment:

The deployment is done by FTP. Git was not used because you need to deploy a compiled version of Angular as well as the vendor folder. Compile Angular using the command npm run build-dev for the dev or npm run build-prod for the prod. Replace files and folders modified by FTP. Send public / angular folder to update the front. Do not delete public / distribution /, otherwise the history of distribution files will be deleted. Check the integrity of the distribution files by issuing the command php public / index.php distribution: check.

V.5. Conclusion

In this chapter, we presented our experiments with the implementation of the model composition plugin. These experiments have shown that we have succeeded in conceptually and practically realizing our approach to compose executable models. We used tools that allow us to support this process and the applications produced constitute an interesting result.

Our plugin is generic enough to support many cases of compositions. It is simple and quite effective. Despite the many improvements that remain to be made, this prototype meets our needs.

CHAPTER VI. CONCLUSION

VI.1. General context

Actually system developers have some serious problems to cope with. The systems they develop are becoming increasingly complex as they should manage a huge development processes where models take a central role.

Indeed, in turn to deal with this complexity, it is impractical to describe the whole system by a single model. Instead, the idea of decomposition provides the key solution to this problem, by breaking down a complex system into several simpler and more manageable blocks made by different developer's team and captures different modelling perspectives of the initial problem.

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.

Quite frequently, during the software development process, practitioners need a model composition step to join and compose their solutions in order to solve the original problem and provide desired solutions. Model composition consists of a fundamental process – by which at least two or more models are merged producing one output model as a result called target model based on model transformation principles.

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. This conclusion returns to the various aspects studied, then presents the limitations of our proposals, which open up so many perspectives.

VI.2. Contribution

Our contribution is to propose a conceptual prototype of model composition represented as a plugin that can interact with many applications and in many environments, both to enhance the global understanding of this operation and to broaden the scope of application of model composition approaches in the MDA community.

Previous work about model composition that we discuss in Chapter III, proposes various operators that handle a large range of operations on model. By studying the different approaches, we have been identified some criteria that influence their implementation. Some approaches deal only with the matching problem, while others feel this issue pre-board as a step in a larger process.

To synthesize, we can defined the composition as a model management operation, which generate a single model by the combination of the contents of at least two models and we intuitively identify some similarities as follows:

• Every technique composes a pair of models.

• Every technique proposes a mechanism for detecting similar or equivalent model element.

• Every technique proposes a mechanism that uses matching for combining models.

• Every approach proposes a mechanism of composition based on the components detected in the beginning.

So, this study allows us to realize that there are two categories of composition: the white box composition which is involved in the internal structure of the components, and the black box composition which comprises components as they are without any change.

We can find it even in the model composition in which there are also two types of composition: one allows to compose component as they are, and the other composes them but after that their structure is transformed.

This study was interested in assessing different approaches by different criteria and to know that, each approach has its own design model for implementing services.

According to this assessment, structural composition mechanisms are clearly defined in the approaches oriented components. Behavioral composition mechanisms are very different depending on the approaches and are based on the notion of scheduling. Some composition techniques already proposed various operations on a set of models.

In special cases, reusing or adapting these techniques seems an interesting path to build a new composition model operation.

Indeed several techniques of composition method have been suggested in the literature. However, there is no work that considers the coexistence of these different composition methods in order to answer practical questions such as: when should we prefer one composition method over the other? Is it possible to solve a given problem of a several composition methods?

So in our work we focus on MDA approach as a whole concept through a novel conceptual for automatic model composition based on the two-hemisphere model driven approach, which is an approach involved in the context of model driven architecture and proposes to create models from initial presentation of problem domains.

The idea of the two-hemisphere-model driven approach comes from the necessity to implement the concept of separation—be able to create specifications that capture requirements in a form that is understandable by less technical stakeholders; for example, the project manager; these people were not comfortable with UML class diagrams, but were perfectly able to understand the required information represented in a simple graphical manner.

Indeed, our conceptual prototype start from the point where we have several UML class diagrams made by several team development process and we need to combine them in order to have a global representation of the system through one class diagram. If we take the example of two operations in two models that appear with the same signature (name, type, parameters), so to remedy this problem, it is necessary to include a step of reconciliation between the separate designs or strengthen semantics associated with the input metamodel, so that we can implement finer comparison strategies that address the behaviors described by the methods.

In this context we were working in collaboration with Riga Technical University of Latvia in this taxonomy to create a prototype based on Two Hemisphere Model Approach in order to compose models in order to allow a low coupling between applications.

The approach is a sequence of the two-hemisphere model driven approach and answers in its turn the standards defined by MDA. The first phase will be a general analysis to build the system requirement which. The second phase is decentralized on design phase, during this phase several teams can work separately to achieve design templates for blocks belonging to the same system. The third phase is a conflict resolution phase between design models which aims to identify and treat addresses conflicts of modelling between models in the frame of "Multi-modelling paradigm" [25]. We are primarily concerned with this syntactic conflicts over naming modelling elements problems, and structural inconstancies. The last step is to merge bricks in order to achieve the overall model. The two-hemisphere model driven approach on which our work is based proposes using of business process model and concept model to represent systems in the platform independent manner and describes how to transform these models into UML diagrams. The strategy supports gradual model transformation from problem domain models into program components, where problem domain models reflect two fundamental things: system functioning (processes) and structure (concepts and their relations).

The main contribution of this thesis is thus to propose a novel conceptual definition of model composition. A webservices architecture differs from a classic monolithic approach in that it breaks down an application to isolate key functions. Each of these functions is called a "service" and these services can be developed and deployed independently of each other. Thus, each can operate without affecting the others.

Concretely, we show that to achieve this goal, we must first compose the metamodels, then the models. We explain in this work how these compositions can be performed without modifying the compound elements, using webservices, which is an architectural approach for application development.

A webservices architecture differs from a classic monolithic approach in that it breaks down an application to isolate key functions. Each of these functions is called a "service" and these services can be developed and deployed independently of each other. Thus, each can operate without affecting the others.

Webservices [24] propose an architectural style where applications are decomposed into small independent building blocks (the webservices), each of them focused on a single business capability. Webservices communicate with each other with lightweight mechanisms and they can be deployed and maintained independently, which leads to more agile developments and technological independence between them [25]. The decomposition of a system into webservices forces developer teams to build webservice compositions to provide their customers with valuable services [21]. It seems that the decentralized nature of webservices makes the choreography approach more appropriate to define these compositions [12].

This approach is first validated by the creation of a plugin based on the webservices approach. This plugin or service is able to run in multiple environments and interact with multiple applications.

VI.3. Limitations and Observations

We have presented a prototype for model composition based on the-two hemisphere model driven approach, which offer some power in understanding and implementing the composition activities.

Our work in based on the evaluation of the existing methods in order to identify the approach that better specifies the composition process and then produce a new one that completes and remains unfilled rows in each evaluated method.

The input to our prototype consists of two models, a set of comparison and matching rules between input model elements, and a set of declarative features provided by the analyst or the user in order to constrain and drive model compositions activities.

The target model is automatically generated, reviewed and further modified from the analyst by adding more restrictions and properties. We have defined a repository for the resolution of potential composition conflicts [14]. It allows both to adapt input models and fix the composed model.

The principal limitation of the proposed approach is that to be reusable the framework only relies of the structure of the models to compose.

The type of correspondence between elements are the only features which can be used to take into account some semantics of models to compose. Our current experiments show that it is not an issue when working with structural models such as class diagrams, but it becomes a clear limitation when working with modelling languages such as sequence diagrams.

To produce a meaningful composition operator for sequence diagrams, the order in which events and messages have to be composed is based on the semantics of sequence diagrams. Indeed, there are still different problems we need to address in our prototype.

The major problem we need to investigate is related to the merging rules. As mentioned before, actually we cannot rigorously talk about the composition of elements behavior, in order to then be able to compose a valid behavioral features. To solve the problem we need to work on the decomposition of concerns in terms of responsibilities in order to be able to define composition rules in a fine level of granularity at which a conflicting situation can be handled.

VI.4. Perspectives

We provide a modeling prototype that supports the definition of merging, matching and managing conflicts in order to build a specific model composition operations. The prototype 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.

In the context of this thesis, we focus effort on the characterization of the key concepts of merging and matching in model composition. The systematic literature review results presented in Chapter II reflect the variability of rules that existing model composition approaches put into action to achieve a specific purpose in a specific context. The conceptual model composition prototype described in this paper currently handles only homogeneous models, those that share the same meta-model. It would be interesting to extend this approach to handle heterogeneous input models as well. Also, we are currently dealing only with one-to-one match relationships between input model elements.

We plan to investigate the extension of our approach for handling many-to-many match relationships. We also plan to extend our prototype to provide a better support for the interactive weaving process, and captures positive and negative result of previous interactions.

As a future work to what is presented in this thesis we are currently investigating a sophisticate mechanism that allows automating also the verification part which involves actually an interaction with an analyst in order to review the target model. This automation will allow focusing more on the merging process and benefit from the generic implementation for the whole process.

In the context of our plugin based on webservices architecture, the possible perspetive for improving the meta-model are distributed according to conceptual and implementation aspects.

Conceptual aspects

We introduced the process of inheritance between types of collaboration schema. This process is a key part of our multi-tenant management.

It is the support that facilitates the development of specialized behaviors of the composite service to face the particular requirements of some of its customers. However, this process of inheritance still needs to be formally specified, in particular on the possibility of inheritance between services.

Another process critical to the reification of a service composition into a composite service, is the dynamic generation of service descriptions. This process should be able to build the service description of a composite service from the descriptions of its constituents. Thus, it will allow the publication of composite services which can be reused homogeneously with any other service. It will participate in the incremental constructions of composites compositions and improve the reuse.

VI.5. Publications

This section presents the publications that we produced during this thesis:

- « El Marzouki N., Lakhrissi Y., El Mohajir M., Nikiforova.O. Enhancing Conflict Resolution Mechanism for Automatic Model Composition ». Web of Science, applied computer systems, Riga Technical University Journal (doi: 10.1515/acss-2016-0006).

- « El Marzouki N., Lakhrissi Y., El Mohajir M. A Comparative Study of Structural Model Composition Methods and Techniques ». David Journal (doi: 10.17265/1934-7332/2016.03.005).

- « El Marzouki Nisrine, Younes Lakhrissi, Oksana Nikiforova, Mohammed El Mohajir, Konstantins Gusarovs. Behavioral And Structural Model Composition Techniques: State Of Art And Research Directions », WSEAS journal. 2017

- « El Marzouki N., Lakhrissi Y., El Mohajir M., Nikiforova.O. Toward a Generic Metamodel for Model Composition Using Model Transformation ». Scopus Procedia Computer Science, volume 104, 2017, Pages 564- 571, Elsevier.

- « El Marzouki N., EL AISSI M., LOUKILI Y.; Lakhrissi Y., El Mohajir M., Nikiforova.O., Implementing a Digital Workspace in the Era of Covid-19 Based on Model Compositon), to 2020 6th IEEE Congress on Information Science and Technology (CiSt).

- « Ayoub KORCHI A., Mohamed Karim KHACHOUCH M., BENJELLOUN S., El Marzouki N., Lakhrissi Y. Toward Moroccan Virtual University: Technical Proposal') for The IEEE international conference on electronics, control, optimization and computer science icecocs'20.

- « El Marzouki N., Lakhrissi Y., El Mohajir M., Nikiforova.O. May 2017 The Application Of An Aytomatic Model Composition Prototype On the- Two Hemishpere Model Driven Approach, DOI: 10.1109/WITS.2017.7934673 Conference: 2017 IEEE International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)

- « Ņikiforova, O., El Marzouki, N., Kuņicina, N., Vangheluwe, H., Florin, L., Iacono, M., Al-Ali, R., Orue, P. Several Issues on Composition of Cyber- Physical Systems Based on Principles of the Two- Hemisphere Modelling», Scopus In: Proceedings of the 4th Workshop of the MPM4CPS COST Action, Poland, Gdańsk, 15-16 September, 2016. Malaga: Departamentos Lenguajes y Ciencias de la Computación Universidad de Málag, 2016, pp.44-55

- « El Marzouki N., Lakhrissi Y., El Mohajir M. A Study of Behavioral and Structural Composition Methods and Techniques ». IEEE Explore (doi: 978-1-4673-7689-1/16/2016 IEEE)

REFERENCES

[Abiteboul et al., 91] S. Abiteboul, A. Bonner. "Objects, Views". Proceedings of ACM SIGMOD, pp 238-247, mai 1991.

[ACLF10] M. Acher, P. Collet, P. Lahire, and R. France. Comparing approaches to implement feature model composition. Modelling Foundations and Applications, pages 3–19, 2010.

[AEC+10] A. Anwar, S. Ebersold, B. Coulette, M. Nassar, and A. Kriouile. A rule-driven ap-proach for composing viewpoint-oriented models. Journal of Object Technology, 9(2):89–114, 2010.

[Al Ali et al,14] Al Ali, R., Bureš, T., Gerostathopoulos, I., Keznikl, J., Plášil, F., 2014. Architecture Adaptation Based on Belief Inaccuracy Estimation. In Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA 2014).

[Amyot et al., 03] D. Amyot, A. Eberlein. "An evaluation of scenario notations, construction approaches for telecommunication systems development". Telecommunication Systems, 24(1):61–94, 2003.

[APCS03] Marcus Alanen, Ivan Porres, Turku Centre, and Computer Science. Difference and union of models. In UML 2003. LNCS, pages 2–17. Springer, 2003.

[ATL 2005] "The ATL UML to JAVA transformation". Available at http://www.eclipse.org/gmt/atl/atlTransformations/

[ATL, 07] Eclipse/M2M Project Web Page. http://www.eclipse.org/m2m/, 2007.

[ATL-Java] The ATL UML to JAVA transformation. Available at http://www.eclipse.org/gmt/atl/atlTransformations/

[Babris et al,16] Badica, A., Badica, C., Leon, F., Buligiu, I., 2016. Modeling and Enactment of Business Agents Using Jason, In Proceedings of the 9th Hellenic Conference on Artificial Intelligence, SETN 2016.

[Babris et al,19] Babris, K., Nikiforova, O., Sukovskis, U., 2019. Brief Overview of Modelling Methods, Life-Cycle and Application Domains of Cyber-Physical Systems. Applied Computer Systems, 2019, Vol. 24, no. 1, pp. 5.-12.

[Baniassad et al., 04] E. Baniassad, S. Clarke. "Theme: An approach for aspect-oriented analysis, design". Proc. of the International Conference on Software Engineering, p. 158-167, 2004.

[Barbierato et al,11] Barbierato, E., Gribaudo, M., Iacono, M., 2011. Exploiting multiformalism models for testing and performance evaluation in SIMTHESys, In Proceedings of 5th International ICST Conference on Performance Evaluation Methodologies and Tools - VALUETOOLS 2011

[Barbierato et al,13] Barbierato, E., Dei Rossi, G., Gribaudo, M., Iacono, M., Marin, A., 2013. Exploiting product forms solution techniques in multiformalism modelling. In Electronic Notes in Theoretical Computer Science, Elsevier

[Barbierato et al,16] Barbierato, E., Gribaudo, M., Iacono, M., 2016. Modeling Hybrid Systems in SIMTHESys, In Electronic Notes on Theoretical Computer Science, Elsevier.

[Bardou, 98a] D. Bardou. "Etude de langages à prototypes, du mécanisme de délégation et de son rapport à la notion de point de vues". Thèse de doctorat en Informatique, LIRMM, université de Montpellier 2, 1998.

[Bardou, 98b] D. Bardou. "Roles, Subjects, Aspects: How do they relate?". Position paper at the Aspect Oriented Programming Workshop. 12th European Conference on Object-Oriented Programming (ECOOP '98), LNCS, vol. 1543, Springer, 1998.

[Barra et al., 04] E. Barra, G. Genova, J. Llorens. "An approach to Aspect Modelling with UML 2.0". In Aspect-Oriented Modeling Workshop, AOM 2004, Lisbon, Portugal, October 2004.

[Basch et al., 03] M. Basch, A. Sanchez. "Incorporating Aspects into the UML". Third International Workshop on Aspect-Oriented Modeling (AOM'03), March 2003.

[BBDF+06] J. Bézivin, S. Bouzitouna, M. Del Fabro, M.P. Gervais, F. Jouault, D. Kolovos, I. Kurtev, and R. Paige. A canonical scheme for model

composition. In Model Driven Architecture–Foundations and Applications, pages 346–360. Springer, 2006.

[BDK92] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In Advances in Database Technology?EDBT'92, pages 152–167. Springer, 1992.

[Beek, 94] M. Von Der Beeck. "A comparison of Statecharts variants". In FTRTFT '94, volume 863 of LNCS, pages 128-148. Springer-Verlag, 1994.

[Béz05] J. Bézivin. On the unification power of models. Software and Systems Modeling, 4(2):171–188, 2005.

[Bézivin et al., 05] J. Bézivin, F. Jouault. "Using ATL for Checking Models". In proc. of the International Workshop on Graph, Model Transformation (GraMoT), Tallinn, Estonia, 2005.

[Bézivin et al., 05b] J. Bézivin, F. Jouault, D. Touzet. "An introduction to the ATLAS Model Management Architecture". Rapport de recherche N° 05.01. LINA, université de Nantes, Février 2005.

[Bézivin, 04] J. Bézivin, F. Jouault, P. Rosenthal, P. Valduriez. "The AMMA platform support for modeling in the large, modelling in the small". Research Report LINA, (04.09), 2004.

[Biro et al,18] Biro M., Mashkoor A., Sametinger J., Seker R. Software Safety and Security Risk Mitigation in Cyber-physical Systems. IEEE Software. 2018. vol. 35. no. 1. pp. 24–29.

[BMS+19] Biro M., Mashkoor A., Sametinger J., Seker R. Software Safety and Security Risk Mitigation in Cyber-physical Systems. IEEE Software. 2018. vol. 35. no. 1. pp. 24–29.

[Bontemps et al., 04a] Y. Bontemps, P. Heymans. "As fast as sound (lightweight formal scenario synthesis, verification)". In 3rd International Workshop on Scenarios, State Machines: Models, Algorithms,, Tools (SCESM '04), Edinburgh, UK, 2004.

[Bontemps et al., 04b] Y. Bontemps, P.Y. Schobbens, C. Löding. "Synthesis of Open Reactive Systems from Scenario-Based Specifications,". Fundamenta Informaticae, vol. 62, no. 2, pp. 139-169, July 2004.

[Bontemps, 01] Y. Bontemps. "Automated Verification of State-based Specifications Against Scenarios (A Step towards Relating Inter-Object to Intra-Object Specifications)". Master's thesis, University of Namur, rue Grandgagnage, 21 - 5000 Namur(Belgium), June 2001.

[Bor07] A. Boronat. Moment: a formal framework for model management. PhD in Comput-er Science, Universitat Politenica de Valencia (UPV), Spain, 2007.

[BP08] Cédric Brun and Alfonso Pierantonio. Model differences in the eclipse modelling framework. UPGRADE The European J for the Informatics Professional, IX(2):29–34, 2008.

[Brill et al., 04] M. Brill, W. Damm, J. Klose, B. Westphal, H. Wittke. "Live Sequence Charts, An Introduction to Lines, Arrows,, Strange Boxes in the Context of Formal Verification". LNCS 3147, pp. 374–399, Springer-Verlag Berlin Heidelberg 2004.

[Budinsky et al., 03] F. Budinsky, D. Steinberg, R. Ellersick. "Eclipse Modeling Framework : A Developer's Guide". Addison-Wesley.

[Buh, 98] R. J. A. Buhr. "Use Case Maps as Architectural Entities for Complex Systems". In: IEEE Transactions on Software Engineering, 24(12), 1131-1155, Dec 1998.

[Caron et al., 03] O. Caron, B. Carré, A. Muller, G. Vanwormhoudt. "A Framework for Supporting Views in Component Oriented Information Systems". OOIS, vol. 2817 de Lecture Notes in Computer Science, Springer, p. 164-178, septembre 2003.

[Carré et al., 90-a] B. Carré, L. Dekker, J.M. Geib. "Multiple, Evolutive Représentation in the ROME language". Actes de TOOLS'90, pp. 101-109, 1990.

[Carré et al., 90-b] B. Carré, J.M. Geib. "The Point of View Notion for Multiple Inheritance". Proceedings of ECOOP/OOPSLA'90, pp. 312-321, 1990.

[Carré, 89] B. Carré. "Méthodologie orientée objet pour la représentation des connaissances, concepts de points de vue, de représentation multiple et évolutive d'objets". Thèse du LIFL, 1989.

[Castejon, 05] H-N. Castejon. "Synthesizing state-machine behaviour from UML collaborations and Use Case Maps". In: Proc. of the 12th Int. SDL Forum, Norway, LNCS 3530, Springer, 2005.

[CDNFP10] L. Cavallaro, E. Di Nitto, C.A. Furia, and M. Pradella. A tile-based approach for self-assembling service compositions. In Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on, pages 43–52. IEEE, 2010.

[CDREP08] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE, pages 222–231. IEEE, 2008.

[Charrel et al., 93] P.J. Charrel, D. Galaretta, C. Hanachi, B. Rothenburger. "Multiple Viewpoints for Development of Complex Software". Actes de IEEE International Conference on Systems, Man, Cybernetics, pp. 556-561, 17-20 octobre 1993.

[CJC+11b] C. Clasen, F. Jouault, J. Cabot, et al. Virtual Composition of EMF Models. In 7èmes Journées sur l'Ingénierie Dirigée par les Modèles, 2011.

[CJC11a] C. Clasen, F. Jouault, and J. Cabot. Virtualemf: a model virtualization tool. Ad-vances in Conceptual Modeling. Recent Developments and New Directions, pages 332–335, 2011.

[Clarke, 01] S. Clarke. "Composition of Object-Oriented Software Design Models". PhD thesis, Dublin City University, 2001.

[Clarke, 02] S. Clarke. "Extending Standard UML with Model Composition Semantics". Science of Computer Programming, 44, p. 71-100, 2002.

[Clarke01] S. Clarke. "Composition of Object-Oriented Software Design Models". PhD thesis, Dublin City University, 2001.

[Coady et al., 01] Y. Coady, G. Kiczales, M. Feeley, G. Smolyn. "Using Aspect C to Improve the Modularity of Path-Specific Customization in Operating System Code". 8 th European Software Engineering Conference (ESEC), 9 th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), p. 88-98,Vienna, Austria, 2001.

[Cottenier et al., 07] T. Cottenier, A. van den Berg, T. Elrad. "Motorola WEAVR: Model Weaving in a Large Industrial Context". Aspect-Oriented Software Development (AOSD), Vancouver, Canada, 2007.

[Coulette et al., 06] B. Coulette, Y. Lakhrissi, M. Nassar, A. Kriouile. "Notion de patrons multivue - Application au profil VUML". Dans : Workshop OCM-SI, associé à INFORSID 2006, Hammamet, 12/05/2006-14/05/2006, Hermès, mai 2006.

[Coulette et al., 96] B. Coulette, A. Kriouile, S. Marcaillou. "L'approche par points de vue dans le développement orienté objet des systèmes complexes". Revue l'Objet, vol. 2, n°4, pp. 13-20, février 1996.

[CPS,19] Cyber-Physical Systems. URL: https://ptolemy.berkeley.edu/projects/cps/ (read: 11.03.2019).

[Crégut et al., 05] X. Crégut, S. Marcaillou, M. Nassar, B. Coulette. "Un patron de génération de code pour le profil VUML". LMO-OCM'2005, pp. 5-11, Berne, Suisse, mars 2005.

[CRP07] Antonio Cicchetti, Davide Di Ruscio, and Alfonso Pierantonio. A metamodel inde-pendent approach to difference representation. Journal of Object Technology, 6(9):165–185, Octo-ber 2007. TOOLS EUROPE 2007 — Objects, Models, Components, Patterns.

[Cueignet et al., 92] X. Cueignet, V. Lextrait. "Génération de serveur de vues". Thèse de l'université de Sophia Antipolis, décembre 1992.

[CW98] R. Conradi and B. Westfechtel. Version models for software configuration man-agement. ACM Computing Surveys (CSUR), 30(2):232–282, 1998.

[Dahchour et al., 04] M. Dahchour, A. Pirotte, E. Zimányi, "A role model, its metaclass implementation". Information Systems Journal, volume 29, p. 235-270, Elsevier, 2004.

[Dahchour et al., 06] M. Dahchour, H. Rayd, Y. Lakhrissi, A. Kriouile, "Extension d'UML par les rôles". Proc. of the 9th Maghrebian Conference on Information Technologies (MCSEAI 2006), Agadir, Morocco, December 2006.

[Dahchour et al., 07] M. Dahchour, H. Rayd, Y. Lakhrissi, A. Kriouile. "Extension d'UML par les rôles" (version étendue de MCSEAI 2006). Dans : la revue électronique des technologies de l'information ETI, Ecole Mohammadia d'Ingénieurs, Rabat - Maroc, Vol. 4, juin 2007.

[Damm et al., 01] W. Damm, D. Harel. "LSCs: Breathing life into message sequence charts". Formal Methods in System Design, 19(1):45–80. Preliminary version appeared in Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99), 2001.

[DDFV09] M. Didonet Del Fabro and P. Valduriez. Towards the efficient development of model transformations using model weaving and matching transformations. Software and Systems Modeling, 8(3):305–324, 2009.

[DDZ08] J. Dingel, Z. Diskin, and A. Zito. Understanding and improving uml package merge. Software and Systems Modeling, 7(4):443–467, 2008.

[Debrauwer 98] L. Debrauwer. "Des vues aux contextes pour la structuration fonctionnelle de bases de données à objets en CROME". Thèse de doctorat en Informatique, LIFL, Université des Sciences et Technologies de Lille, décembre 1998.

[Dekker, 94] L. Dekker. "FROME : Représentation multiple et classification d'objets avec points de vue". Thèse de l'Université de Lille, juin 1994.

[Del Fabro et al., 06] M. Didonet Del Fabro, J. Bézivin, P. Valduriez. "Weaving Models with the Eclipse AMW plugin". In: Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany, 2006.

[Des00] René Descartes. Discours de la méthode. Flammarion, April 2000.

[DFBJ+05] M.D. Del Fabro, J. Bezivin, F. Jouault, E. Breton, and G. Gueltas. Amw: a generic model weaver. Proceedings of the 1ere Journée sur l'Ingénierie Dirigée par les Modeles (IDM05), 3(4.7):7–11, 2005.

[DFF+09] Z. Drey, C. Faucher, F. Fleurey, V. Mahé, and D. Vojtisek. Kermeta language. Ref-erence Manual, 2009.

[DFV07] M.D. Del Fabro and P. Valduriez. Semi-automatic model integration using match-ing transformations and weaving models. In Proceedings of the 2007 ACM symposium on Applied computing, pages 963–970. ACM, 2007.

[DMR03] H.H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. Web, Web-Services, and Database Systems, pages 221–237, 2003.

[Do06] H.H. Do. Schema matching and mapping-based data integration. Verlag Dr. Müller (VDM), pages 3–86550, 2006.

[DRIP11] D. Di Ruscio, L. Iovino, and A. Pierantonio. What is needed for managing co-evolution in mde? In Proceedings of the 2nd International Workshop on Model Comparison in Practice, pages 30–38. ACM, 2011.

[Ecl06] EMF Eclipse. Eclipse modeling framework. http://www.sysml.org/docs/specs/OMGSysML-v1.1-08-11-01.pdf, 2006.

[Eclipse, 07] QVT Operational - M2M component. http://www.eclipse.org/m2m/qvto/doc/M2M-QVTO.pdf

[El Asri et al., 04] B. El Asri, M. Nassar, A. Kriouile, B. Coulette. "Views, subjects, roles, aspects : A comparison along software lifecycle". Proceedings of 6th International Conference on Enterprise Information Systems ICEIS'04, Porto-Portugal, April 2004.

[El Asri et al., 05] B. El Asri, M. Nassar, B. Coulette, A. Kriouile. "MultiViews component for information development". Proceedings of the 7th International Conference on Enterprise Information Systems (ICEIS'2005), pp. 217-225, Miami, USA, May 24-28, 2005.

[El Asri, 05] B. El Asri. "Vers des composants multivues distribués". Thèse nationale, l'ENSIAS de Rabat, octobre 2005.

[EL Marzouki et al,16] El Marzouki, N., Nikiforova, O., Lakhrissi, Y., El Mohajir, M., 2016. Enhancing Conflict Resolution Mechanism for Automatic Model Composition. In Grundspenkis J. et al. (Eds) Scientific Journal of Riga Technical University Applied Computer Systems.

[EM00] A. Egyed and N. Medvidovic. A formal approach to heterogeneous software model-ing. Fundamental Approaches to Software Engineering, pages 178–192, 2000.

[Epsilon, 06] Epsilon SubProject 2006. http://www.eclipse.org/gmt/epsilon/

[Eva04] E. Evans. Domain-driven design: tackling complexity in the heart of software. Ad-dison-Wesley Professional, 2004.

[Fab08] Marcos Didonet Del Fabro. Amw use case - tool interoperability of bug tracking tools. http://www.eclipse.org/gmt/amw/usecases/interoperability/, June 2008.

[Fav03] J.M. Favre. Meta-model and model co-evolution within the 3d software space. In ELISA: Workshop on Evolution of Large-scale Industrial Software Applications, pages 98–109, 2003.

[FBFG08] F. Fleurey, B. Baudry, R. France, and S. Ghosh. A generic approach for automatic model composition. Models in Software Engineering, pages 7–15, 2008.

[FGC+06] Patrick Farail, Pierre Gaufillet, Agusti Canals, Christophe Le Camus, David Sci-amma, Pierre Michel, Xavier Crégut, and Marc Pantel. The TOPCASED project: a toolkit in open source for critical aeronautic systems design. In Embedded Real Time Software (ERTS), Toulouse, France, 25-27 January 2006.

[Finkelsetin et al., 92] A. Finkelsetin, J. Kramer, B. Nuseibeh, L. Finkelstein, M. Goedicke. "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development". International Journal of Software Engineering, Knowledge Engineering 2(1):31-58, World Scientific Publishing Co, March 1992.

[Finkelstein et al., 90] A. Finkelstein, J. Kramer, M. Goedicke. "Viewpoint Oriented Software Development". Proceedings of Software Engineering, Applications Conference, p. 337-351, Toulouse, December 1990.

[FISCHER95] Fischer, B., "Decomposition of Time Series - Comparing Different Methods in Theory and Practice", Eurostat Working Paper, 1995.

[Fow04] Martin Fowler. Domain specific language. 2004.

[France et al., 04a] R. France, D. Kim, S. Ghosh, E. Song. "A UML-based pattern specification technique", IEEE Trans. Sofw. Eng., 2004.

[France et al., 07] R. France, F. Fleurey, R. Reddy, B. Baudry, S. Ghosh. "Providing Support for Model Composition in Metamodels". In procc of the 11th IEEE EDOC conference, pp 253-264. 2007.

[GEF, 07] The Graphical Editing Framework (GEF). http://www.eclipse.org/

[GJCB09] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. A Domain Specific Language for Expressing Model Matching. In Proceedings of the 5ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM09), pages 33–48, Nancy, France, France, 2009.

[GKJ10] Kelly Garcés, Wolfgang Kling, and Frédéric Jouault. Automatizing the Evaluation of Model Matching Systems. In Workshop on matching and meaning 2010 address = Leicester, United Kingdom, page To appear, Leicester, Royaume-Uni, 2010.

[GKP07] B. Gruschko, D. Kolovos, and R. Paige. Towards synchronizing models with evolv-ing metamodels. In Proceedings of the International Workshop on Model-Driven Software Evolu-tion, 2007.

[GMF, 07] The Graphical Modeling Framework (GMF). http://www.eclipse.org/

[Gottlob et al., 96] G. Gottlob, M. Schrefl, B. Röck B. "Extending object-oriented systems with roles". ACM Trans. Office Information Systems, 14 (3), p. 268-296, 1996.

[Gribaudo et al,14] Gribaudo, M., Iacono, M., 2014. An introduction to multiformalism modelling. In Theory and applications of multi-formalism modelling, IGI-Global.

[Griffin, 04] C. Griffin. "Transformations in Eclipse". Workshop on Model-Driven Development. WMDD / IBM / © ATHENA Consortium 2004.June 2004.

[Gro09] Richard C. Gronback. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional, 1 edition, 2009.

[Harel et al., 00] D. Harel, H. Kugler. "Synthesizing state-based object systems from LSC specifications". Int. J. of Foundations of Computer Science (IJFCS)., 13(1):5–51, Febuary 2002. (Also,Proc. Fifth Int. Conf. on Implementation, Application of Automata (CIAA 2000), July 2000, Lecture Notes in Computer Science, Springer-Verlag, 2000.).

[Harel et al., 03] D. Harel, H. Kugler, R. Merlly, A. Pnueli. "Smart Play-Out". OOPSLA'03, Aaheim, California, USA. ACM 1-58113-751-6/03/0010, October 2003.

[Harel et al., 04b] D. Harel, H. Kugler. "The RHAPSODY semantics of statecharts (on, on the executable core of the UML)" (preliminary version). In SoftSpez Final Report, LNCS 3147, pages 325–354. Springer, 2004.

[Harel et al., 05a] D. Harel, H. Kugler, A. Pnueli. "Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements". Springer-Verlag Berlin Heidelberg 2005.

[Harel et al., 05b] D. Harel, H. Kugler, G. Weiss. "Some Methodological Observations Resulting from Experience Using LSCs, the Play-In/Play-Out Approach". Proc. Scenarios: Models, Algorithms, Tools, Lecture Notes in Computer Science, Springer-Verlag, 2005 .

[Harel et al., 96] D. Harel, A. Naamad. "The STATEMATE semantics of statecharts". ACM Transactions on Software Engineering, Methodology, 5(4):293–333, 1996.

[Harel et al., 97] D. Harel, E. Gery. "Executable object modeling with statecharts". IEEE Computer , pp. 31-42, July 1997.

[Harel et Marelly, 03] D. Harel, R. Marelly. "Specifying, Executing Behavioral Requirements: The Play In/Play-Out Approach". Software, System Modeling (SoSyM), pp 82-107, 2003.

[Harel, 84] D. Harel. "Statecharts: A Visual Formalism for Complex Systems". Science of Computer Programming 8 (1987), 231–274. (Preliminary version: Technical Report CS84-05, The Weizmann Institute of Science, Rehovot, Israel, February 1984.)

[Harel, 87] D. Harel. "Statecharts: A visual formalism for complex systems". Science of Computer Programming, 8(3):231–274, June 1987.

[Harrison et al., 93] W.Harrison, H. Ossher. "Subject-oriented programming : a critique of pure objects". Proceedings of OOPSLA'93, Washington D.C., pp. 411-428, Se, 1993.

[HBJ+08] M. Herrmannsdoerfer, S. Benz, E. Juergens, et al. Cope: A language for the coupled evolution of metamodels and models. In 1st International Workshop on Model Co-Evolution and Consistency Management, 2008.

[Her11] M. Herrmannsdoerfer. Solving the ttc 2011 reengineering case with edapt. Elec-tronic Proceedings in Theoretical Computer Science, 74, 2011.

[Hil09] Guillaume Hillairet. Emftriple: (meta)models on the web of data. http://code.google.com/a/eclipselabs.org/p/emftriple/, June 2009.

[HK03] Jan Hendrik Hausmann and Stuart Kent. Visualizing model mappings in uml. In Proceedings of the 2003 ACM symposium on Software visualization, SoftVis '03, pages 169–178, New York, NY, USA, 2003. ACM.

[HKR+07] C. Herrmann, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel. An algebraic view on the semantics of model composition. In Model Driven Architecture-Foundations and Applica-tions, pages 99–113. Springer, 2007.

[Hu F et al,18] Hu F. Cyber-Physical Systems: Integrated Computing and Engineering Design. New York: CRC Press. 2018. 398 p.

[Huizing, 91] C. Huizing. "Semantics of Reactive Systems: Comparison". PHD thesis, Eindhoven University of Technology, 1991.

[Humberto et al., 05] N. Humberto, M. Castejon. "Synthesizing State-Machine Behaviour from UML Collaborations, Use Case Maps". SDL 2005, LNCS 3530, pp. 339–359, Springer-Verlag Berlin Heidelberg 2005.

[Hyades, 02] Eclipse Hyades Project. http://www.eclipse.org/hyades/

[IFx-site] http://www-if.imag.fr/IFx/

[ITU-MSC, 00] ITU-TS, Recommendation Z.120 (11/99): MSC 2000. ITU-TS, Geneva, 1999.

[ITU-MSC-B, 95] ITU-TS, Recommendation Z .120: Message Sequence Chart (MSC)-Annex B : Algebraic Semantics of Message Sequence Charts . ITU-TS, Geneva, 1995.

[ITU-SDL, 00] ITU-T, Recommendation Z.100: Specification, Description Language (SDL), 2000.

[ITU-UCM, 02] ITU-T, URN Focus Group (2002), Draft Rec. Z.152 - UCM: Use Case Map Notation (UCM). Geneva, 2002.

[ITU-URN] ITU-T, Recommendation Z.150, User Requirements Notation (URN)-Language Requirements, Framework, Geneva, Switzerland. http://www.UseCaseMaps.org/urn/

[Jacobson et al., 04] I. Jacobson, P-W Ng. "Aspect-Oriented Software Development with Use Cases". Addison-Wesley, 2004.

[Java-meta, 05] http://www.eclipse.org/m2m/atl/atlTransformations/UML2Java/ExampleUML2Java%5Bv00.01%5D.pdf

[JFB08] C. Jeanneret, R. France, and B. Baudry. A reference process for model composition. In Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling. ACM, 2008.

[Jouault et al., 05] F. Jouault, I. Kurtev. "Transforming Models with ATL. In Proceedings of the Model Transformations in Practice". Workshop at Models 2005, Montego Bay, Jamaica 2005.

[Jouault et al., 06a] F. Jouault. "Contribution à l'étude des langages de transformation de modèles". Thèse de doctorat, Université de Nantes, septembre

[Jouault et al., 06c] F. Jouault, I. Kurtev. "On the Architectural Alignment of ATL, QVT". In: Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 06). ACM Press, Dijon, France, pages 1188—1195, 2006.

[KDRPP09] Dimitrios S. Kolovos, Davide Di Ruscio, Alfonso Pierantonio, and Richard F. Paige. Different models for model matching: An analysis of approaches to support model differenc-ing. In Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Mod-els, CVSM '09, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[KHHL09] M. Koegel, M. Herrmannsdoerfer, J. Helming, and Y. Li. State-based vs. operation-based change tracking. In proceedings of MODELS'09 MoDSE-MCCM Workshop, 2009.

[Kiczales et al., 01] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W.G. Griswold. "An Overview of AspectJ". Proceeding of ECOOP'01, Springer Verlag LNCS2072, 2001.

[Kiczales, 97] G. KICZALES. "Aspect-Oriented Programming". European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241, Finland, June 1997.

[KICZALES19] G. KICZALES. "Aspect-Oriented Programming". European Conference on Object-Oriented 2019

[KICZALES97] G.KICZALES."Aspect-Oriented Programming". European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241, Finland, June 1997.

[Klein, 06] J. Klein. "Aspects Comportementaux et Tissage". Thèse de l'Université de Rennes 1, Rennes, Décembre 2006.

[KPP06a] D. Kolovos, R. Paige, and F. Polack. Merging models with the epsilon merging language (eml). Model Driven Engineering Languages and Systems, pages 215–229, 2006.

[KPP06b] D.S. Kolovos, R.F. Paige, and F.A.C. Polack. Model comparison: a foundation for model composition and model transformation testing. In Proceedings of the 2006 international workshop on Global integrated model management, pages 13–20. ACM, 2006.

[Kriouile, 95] A. Kriouile, "VBOOM, une méthode orientée objet d'analyse et de conception par points de vue". thèse d'Etat de l'université Mohammed V de Rabat, 1995.

[Kristensen, 96] B.B. Kristensen. "Object-oriented modeling with roles". Proc. of the Int. Conf. on Object Oriented Information Systems, OOIS'95, Springer, Berlin, p.57-71, Dublin, Ireland 1996.

[KRPP10] D. Kolovos, L. Rose, R. Paige, and FAC Polack. The epsilon book. Structure, 178, 2010.

[Krüger, 00] I. H. Krüger. "Distributed System Design with Message Sequence Charts". PhD thesis, Technischen Universität München, July 2000.

[Lakhrissi et al., 07] Y. Lakhrissi, I. Ober, B. Coulette, M. Nassar, A. Kriouile. "Vers la notion de machine à états multivue dans le profil VUML". Dans : Workshop WOTIC 05/07/2007-06/07/2007Rabat, juillet 2007.

[Lakhrissi et al., 08a] Y. Lakhrissi, B. Coulette, I. Ober, M. Nassar, A. Kriouile. "Démarche VUML statique et dynamique - Application à une étude de cas". Rapport de recherche, IRIT/RR-2008-1-FR, IRIT, février 2008.

[Lakhrissi et al., 08b] Y. Lakhrissi, I. Ober, B. Coulette, M. Nassar, A. Kriouile. "Prise en compte des aspects comportementaux dans la démarche de modélisation de VUML". Dans : ERTSI, associé à la conférence INFORSID, Fontainebleau, 27/05/2008-27/05/2008, Hermès, mai 2008.

[Lakhrissi et al., 08c] Y. Lakhrissi, A. Anwar, M. Nassar, A. Kriouile. "Composition des machines à états par point de vue dans VUML". Workshop JIMD'2008. 03/07/2008-05/07/2008. ENSIAS Rabat, juillet 2008.

[Le Guennec, 01] A. Le Guennec. "Génie Logiciel et Méthodes Formelles avec UML Spécification, Validation et Génération de tests". Thèse de l'Université de Rennes 1, Rennes 2001.

[Le Moigne, 90] J.L. Le Moigne. "La modélisation des systèmes complexes". Dunod, 1990.

[Leon et al,16] Leon, F., Badica C., 2016. A Comparison Between Jason and F# Programming Languages for the Enactment of Business Agents. In Proceedings of the International Symposium on Innovations in Intelligent Systems and Applications.

[LGH05] N. Liu, J. Grundy, and J. Hosking. A visual language and environment for compos-ing web services. In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, pages 321–324. ACM, 2005.

[LGJ07] Yuehua Lin, Jeff Gray, and Frédéric Jouault. Dsmdiff: a differentiation tool for domain-specific models. European Journal of Information Systems, 16(4):349–361, 2007.

[Liang et al., 06] H. Liang, J. Dingel et Z. Diskin. "A Comparative Survey of Scenario-based to State-based Model Synthesis Approaches". SCESM'06, Shanghai, China, May 2006.

[Marcaillou et al., 94] S. Marcaillou, A. Kriouile, B. Coulette. "VBOOL : une extension d'Eiffel intégrant le concept de points de vue". actes de MCSEAI'94, pp. 115-125, Rabat, Avril 1994.

[Marcaillou, 95] S. Marcaillou. "Intégration de la notion de points de vue dans la modélisation par objets – Le langage VBOOL". thèse de l'université Paul Sabatier de Toulouse, 1995.

[Marino, 93] O. Marino. "Raisonnement classificatoire dans une représentation à objets multi-points de vue". thèse de l'Université Joseph Fourier-Grenoble 1, novembre 1993.

[Marzak, 97] A. Marzak. "Conception de VBTOOL, outil support de la méthode VBOOM, réalisation des fonctionnalités : Analyse et conception". Thèse pour l'obtention du diplôme de spécialité de 3ème cycle de l'université Mohamed V, 1997.

[MBBF10] S. Mosser, A. Bergel, and M. Blay-Fornarino. Visualizing and assessing a composi-tional approach of business process design. In Software Composition, pages 90–105. Springer, 2010.

[MDA09] "MDA guide version 1.0.1." [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf. [Accessed:

[MILANI016] Fredrik Milani et al, "High-order statistics in global sensitivity analysis: Decomposition and model reduction-Computer Methods in Applied Mechanics and Engineering", Volume 301, 1 April 2016, Pages 80-115

[Mili et al., 01] H. Mili, H. Mcheick, J. Dargham, S. Dalloul. "Distribution d'objets avec vues". Revue L'Objet-7/2001, LMO'2001, pp. 27-44, 2001.

[MKBJ08] B. Morin, J. Klein, O. Barais, and J.M. Jézéquel. A generic weaver for supporting product lines. In Proceedings of the 13th international workshop on Early Aspects, pages 11–18. ACM, 2008.

[Muller et al., 03] A. Muller, O. Caron, B. Carré, G. Vanwormhoudt. "Réutilisation d'aspects fonctionnels des vues aux composants". Revue RSTI-L'objet, vol. 9, n°1-2, LMO'2003, pp. 241-255, 2003.

[Muller et al., 05] P-A. Muller, F. Fleury, J-M. Jezequel. "Weaving executability into object-oriented meta-languages". In Proceedings of MODELS/UML 2005, pages 264–278, Montego Bay, Jamaica, October 2005.

[Muller, 06] A. Muller. "Construction de systèmes par application de modèles paramétrés". Thèse de l'Universit´e de Lille 1, 2006.

[Nas03] M. Nassar. Vuml: a viewpoint oriented uml extension. In Automated Software En-gineering, 2003. Proceedings. 18th IEEE International Conference on, pages 373–376. IEEE, 2003.

[Nassar et al., 03] M. Nassar, B. Coulette, X. Crégut, S. Marcaillou, A. Kriouile. "Towards a View based Unified Modeling Language". Proceedings of 5th International Conference on Enterprise Information Systems ICEIS'03, pp. 257-265, Angers, April 2003.

[Nassar et al., 04] M. Nassar, B. Coulette, A. Kriouile. "Génération de code dans VUML". Journal Marocain d'Automatique, d'Informatique et de Traitement du Signal, article sélectionné de la conférence COPSTIC'03, 2004.

[Nassar et al., 09] M. Nassar, A. Anwar, S. Ebersold, B. El Asri, B. Coulette, A. Kriouile. "A Code Generation in VUML profile: a Model Driven Approach". 7th IEEE/ACS AICCSA 2009. IEEE Computer Society Press, Rabat, May 10-13, 2009.

[Nassar, 03] M. Nassar. "VUML : a Viewpoint oriented UML Extension". Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE'2003 - Doctoral symposium). pp. 373-376, Montreal, Canada, October 6-10, 2003.

[Nassar, 05] M. Nassar. "Analyse/conception par points de vue : le profil VUML". Thèse INPT, Toulouse, 28 septembre 2005.

[Nicolas et al., 05] H. Nicolas, C. Martinez. "Synthesizing State-Machine Behaviour from UML Collaborations, Use Case Maps". In SDL Forum, pages 339–359, 2005.

[Nikiforova et al,17] Nikiforova O., El Marzouki N., Gusarovs K., Vangheluwe H., Bures T., Al-Ali R., Iacono M., Orue Esquivel P., and Leon F. "The Two-Hemisphere Modelling Approach to the Composition of Cyber-Physical

Systems" - Proceedings of International Conference on Software Technologies (ICSOFT 2017), 24-26 July, 2017, Madrid, Spain. SCITEPRESS Digital Library, pp. 286-293, DOI: 10.5220/0006424902860293

[NKA+15] Nikiforova O., Kozacenko L., Ahilcenoka D., Gusarovs K., Ungurs D., Jukss M., Comparison of the Two-Hemisphere Model-Driven Approach to Other Methods for Model-Driven Software Development, Scientific Journal of Riga Technical University: Applied Computer Systems, Grundspenkis J. et al. (Eds), Vol.18, 2015, pp. 33-42

[Ober et al., 06] I. Ober, S. Graf and I. Ober. "Validating timed UML models by simulation and verification". International Journal of Software Tools for Technology Transfer (STTT), Volume 8, Number 2, pages 128-145, Springer Verlag, April, 2006.

[Ober et al., 08a] I. Ober, Y. Lakhrissi. "Observation-based interaction, concurrent aspect-oriented programming". Dans : International Conference on Software Engineering Research, Management, Applications (SERA 2008), Prague, Rép. Tcheque, 20/08/2008-22/08/2008, Walter Dosch, Roger Lee (Eds.), Springer, SCI, août 2008.

[Ober et al., 08b] I. Ober, B. Coulette, Y. Lakhrissi. "Behavioral modelling, composition of object slices using event observation". Dans MODELS 2008 (ACM/IEEE 11th International Conference on Model Driven Engineering Languages, Systems), Toulouse 28/09/2008-03/10/2008, Springer, 2008.

[ObjectGeode-site] ObjectGeode, available at http://www.telelogic.com/products/objectgeode/.

[OdO07] K.S.F. Oliveira and T.C. de Oliveira. A guidance for model composition. In Soft-ware Engineering Advances, 2007. ICSEA 2007. International Conference on, pages 27–27. IEEE, 2007.

[Omega-site] http://www-omega.imag.fr/tools/IFx/IFx.php

[OMG09] "Unified modeling language: superstructure v.2.2," Object Management

[OMG-CORBA] OMG, CORBA Components , version 3.0 full specification. OMG document formal/02-06-65. June 2002. http://www.omg.org/cgi-bin/doc?formal/02-06-65

[OMG-EMOF, 06] [OMG, 06] Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Core

[OMG-MOF 08] Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.0, avril 2008.

[OMG-MOF] OMG 2002. OMG/MOF Meta Object Facility (MOF) 1.4. Final Adopted Specification Document. formal/02-04-03.

[OMG-OCL] OMG 2003, UML2 OCL Final Adopted Specification, http://www.omg.org/docs/ptc/03-10-14.pdf.

[OMG-UML] Object Management Group, Inc. Unified Modeling Language (UML) 2.1.2 Superstructure, novembre 2007. http://www.omg.org/uml.

[OMG-XMI, 03d] XML Metadata Interchange (XMI), v2.0. www.omg.org/cgi-bin/doc?formal/2003-05-02

[OMG-XML] XML Metadata Interchange (XMI), v2.0. www.omg.org/cgi-bin/doc?formal/2003-05-02

[OMN+,20] Oksana Nikiforova, Mauro Iacono, Nisrine El Marzouki, Andrejs Romanovs and Hans Vangheluwe Enabling Composition of Cyber-Physical Systems with the Two-Hemisphere Model-Driven Approach, In: MULTI-PARADIGM MODELLING APPROACHES FOR CYBER-PHYSICAL SYSTEMS. B.Tekinerdogan, D.Blouin, H.Vangheluwe, M.Goulão, P.Carreira, V.Amaral ed. 125 London Wall, London EC2Y 5AS, United Kingdom: Academic Press is an imprint of Elsevier, 2020. pp. 149-168, ISBN 978-0-12-819105-7.

[Omondo, 01] Omondo Eclipse UML. http://www.eclipsedownload.com/index.html

[Ossher et al., 01] H. Ossher, P. Tarr. "Using multidimensional separation of concerns to (re)shape evolving software". Communications of the ACM, Vol. 44, No. 10, pp. 43-50, October 2001.

[Ossher et al., 95] H. Ossher, M. Kaplan, W. Harrison, A. Katz, V. Kruskal. "Subject-oriented composition rules". Proceedings of the ACM Conference on Object-Oriented Systems, Languages,, Applications, Austin, TX, OOPSLA'1995, pp. 235-250, Oct 1995.

[OWK03] Dirk Ohst, Michael Welle, and Udo Kelter. Differences between versions of uml diagrams. SIGSOFT Softw. Eng. Notes, 28(5):227–236, September 2003.

[Peltier, 02] M. Peltier. "Transformation entre un profil UML et un métamodèle MOF". Revue l'Objet, vol. 8, n°1-2/2002, LMO'2002, p. 25-40, 2002.

[Pernici, 90] B. Pernici. "Objects with roles". Proceedings of the ACM--IEEE Conference on Office Information Systems, Cambridge, MA, 1990.

[Pil08] Integration Pillar. Integrating continuity of operations (coop) into the enterprise architecture. http://www.juniper.net/us/en/local/pdf/resource-guides/9050014-en.pdf, January 2008.

[Pilone et al., 07] D. Pilone, N. Pitman. "UML2 In A Nutshell". Shroff Publishers & Distributors O'REILLY, 2007.

[Pnueli et al., 91] A. Pnueli, M. Shalev. "What is in a step: On the semantics of Statecharts". In TACS '91, volume 526 of LNCS, pages 244--264. Springer-Verlag, 1991.

[PragmaDev-site] PragmaDev: RTDS V3.1, http://www.pragmadev.com/

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. The VLDB Journal, 10(4):334–350, December 2001.

[Reddy et al., 06] Y. Reddy, S. Ghosh, R. France, G. Straw, J-M. Bieman, N. McEachen, E. Song, G. Georg. "Directives for Composing Aspect-Oriented Design Class Models". Transactions of Aspect-Oriented Software Development, Vol.1, No. 1, LNCS 3880, p75-105, Springer, 2006.

[RFG+05] R. Reddy, R. France, S. Ghosh, F. Fleurey, and B. Baudry. Model composition-a signature-based approach. In Aspect Oriented Modeling (AOM) Workshop, 2005.

[Rhapsody-Guide] I-Logix. Rhapsody 6.0 User Guide.

[Rhapsody-Tutorial] I-Logix. Tutorial for Rhapsody in J (Release 4.1 MR2), 2003.

[Riehle et al., 98] D. Riehle. "Framework Design: A Role Modeling Approach". PhD thesis, No. 13509. Zrich, Switzerland, ETH Zrich, 2000.

[Rieu et al., 92] D. Rieu, G.T. Nguyen. "Object Views for Engineering Databases". Actes de "third international conference on data, knowledge systems for manufacturing, engineering, AFCET'92", pp. 335-349, Lyon, mars 1992.

[RRH03] A. Rausch, B. Rumpe, and L. Hoogendoorn. Aspect-oriented framework modeling. In Proceedings of the 4th AOSD Modeling with UML Workshop (UML Conference 2003). Citeseer, 2003.

[Rumbaugh et al., 96] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy et W. Lorensen. "OMT : modélisation et conception orientées objet". Prentice-Hall, 1996.

[RV08] J.E. Rivera and A. Vallecillo. Representing and operating with model differences. Objects, Components, Models and Patterns, pages 141–160, 2008.

[SE05] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. Jour-nal on Data Semantics, IV:146–171, December 2005.

[Softeam, 99] Softeam. "Profiles UML et langage J : Contrôlez totalement le développement d'applications avec UML". White Paper, 1999.

[Soley et al., 00] Soley et al., MDA Model Driven Architecture, by Richard Soley and the OMG Staff Strategy Group, Object Management Group White Paper, Draft 3.2 - November 27, 2000.

[Spinczyk et al., 02] O. Spinczyk , G.,reas, S.P. Wolfgang. "AspectC++: an aspect-oriented extension to the C++ programming language". Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile, embedded applications, Sydney, Australia, February 01, 2002.

[SRS07] S. R. Schach, Object-Oriented & Classical Software Engineering, 7th ed. McGraw-Hill Education, 2007.

[Steimann, 01] F. Steimann. "Role = Interface: a merger of concepts". Journal of Object-Oriented Programming, vol. 14(4), pp. 23–32, 2001.

[Straw et al., 04] G. Straw, G. Georg, E. Song, S. Ghosh, R. France, J-M. Bieman. "Model composition directives". In Proceedings of 7th International Conference on The Unified Modeling Language. Model Languages, Applications (UML 2004), volume 3273 of LNCS, pages 84–97. Springer, 2004.

[Sztipanovits et al., 97] J. Sztipanovits, G. Karsai. "Model-Integrated Computing". Computer, Apr. 1997, pp. 110-112.

[TAR+99] P.L. Tarr, H. Ossher, W. Harrison, M. Stanley, Jr. Sutton. "N Degrees of Separation: Multi-Dimensional Separation of Concerns". International Conference on Software Engineering, pp. 107-119, 1999.

[Tarr et al., 99] P.L. Tarr, H. Ossher, W. Harrison, M. Stanley, Jr. Sutton. "N Degrees of Separation : Multi-Dimensional Separation of Concerns". International Conference on Software Engineering, pp. 107-119, 1999.

[Tau-site] Telelogic Tau, available at http://www.telelogic.se/products/tau/.

[TBWK07] Christoph Treude, Stefan Berlik, Sven Wenzel, and Udo Kelter. Difference compu-tation of large models. In Proceedings of the the 6th joint meeting of the European software engi-neering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ESEC-FSE '07, pages 295–304, New York, NY, USA, 2007. ACM.

[UAW06] Steffen Zschaler Uwe Aßmann and Gerd Wagner. Ontologies for Software Engi-neering and Software Technology, chapter Ontologies,

Metamodels, and the Model-Driven Para-digm, pages 249–273. null, 2006.

[Uchitel, 03] S. Uchitel. Elaboration of Behaviour Models, Scenario based Specifications using Implied Scenarios. PhD thesis, Imperial College London, January 2003.

[UCM] Use Case Maps Web Page, UCM Users Group, 1999. http://www.UseCaseMaps.org

[UML07] OMG UML. Uml 2.0: Superstructure specification. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/, November 2007.

[UML11] OMG UML. Uml 2.4.1. http://www.omg.org/spec/UML/2.4.1/, August 2011.

[vdBPV10] Mark van den Brand, Zvezdan Protic, and Tom Verhoeff. Fine-grained metamodel-assisted model comparison. Ín Proceedings of the 1st International Workshop on Model Compari-son in Practice, IWMCP'10, pages 11–20, New York, NY, USA, 2010. ACM.

[VH10] Konrad Voigt and Thomas Heinze. Metamodel matching based on planar graph edit distance. In Proceedings of the Third international conference on Theory and practice of model transformations, ICMT'10, pages 245–259, Berlin, Heidelberg, 2010. Springer-Verlag.

[VIR10] Konrad Voigt, Petko Ivanov, and Andreas Rummler. Matchbox: combined meta-model matching for semi-automatic mapping generation. In Proceedings of the 2010 ACM Sympo-sium on Applied Computing, SAC '10, pages 2281–2288, New York, NY, USA, 2010. ACM.

[Whittle et al., 00] J. Whittle, J. Schumann. "Generating statechart designs from scenarios". In 22nd International Conference on Software Engineering (ICSE '00), pages 314–323, ACM Press, New York, NY, USA, 2000.

[WIDRP12] D. Wagelaar, L. Iovino, D. Di Ruscio, and A. Pierantonio. Translational semantics of a co-evolution specific language with the emf transformation virtual machine. Theory and Prac-tice of Model Transformations, pages 192–207, 2012.

[wik11] Eclipse wiki. Development resources/howto/proposal phase. http://wiki.eclipse.org/Development_Resources/HOWTO/Proposal_Phase, August 2011.

[XS05] Zhenchang Xing and Eleni Stroulia. Umldiff: an algorithm for object-oriented de-sign differencing. In Proceedings of the 20th IEEE/ACM

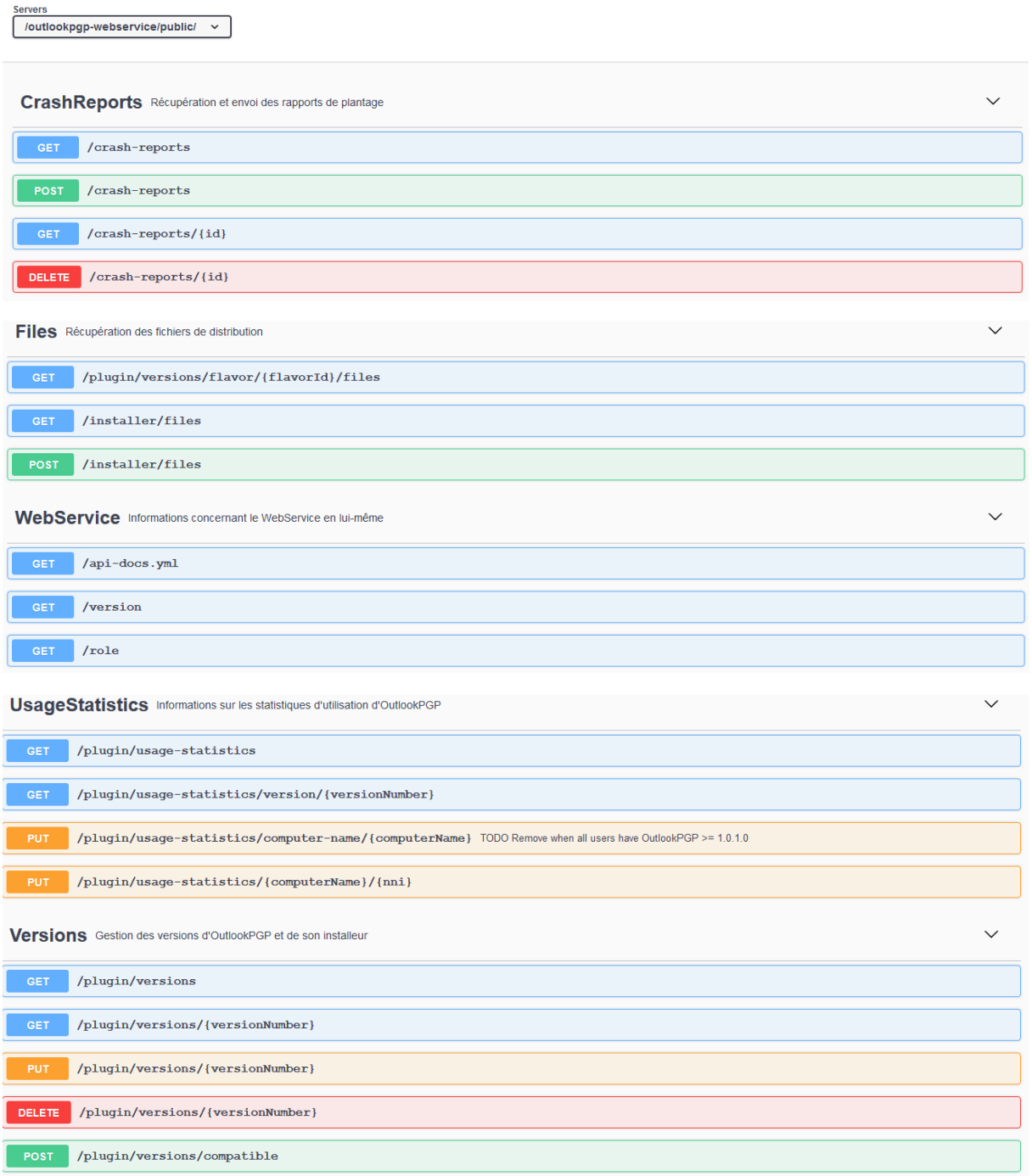
international Conference on Automated software engineering, ASE '05, pages 54–65, New York, NY, USA, 2005. ACM.

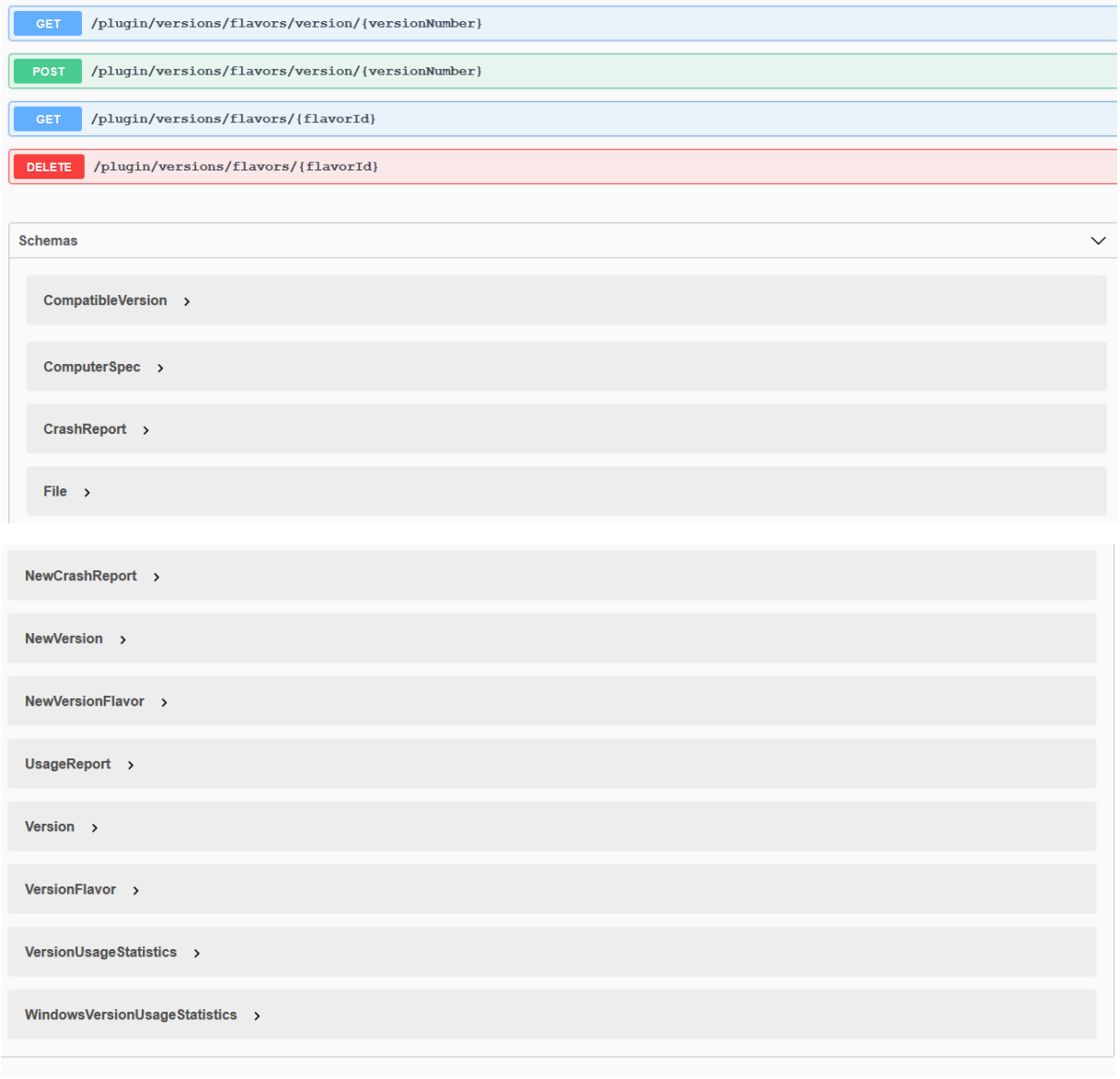
[ZDD06] A. Zito, Z. Diskin, and J. Dingel. Package merge in uml 2: Practice vs. theory? Mo-del Driven Engineering Languages and Systems, pages 185–199, 2006.

[Ziadi et al., 04] T. Ziadi, L. Helouet, J. Jézéquel. "Revisiting statechart synthesis with an algebraic approach". In 26th International Conference on Software Engineering (ICSE '04), pages 242–251, IEEE Computer Society, Washington, DC, USA, 2004.

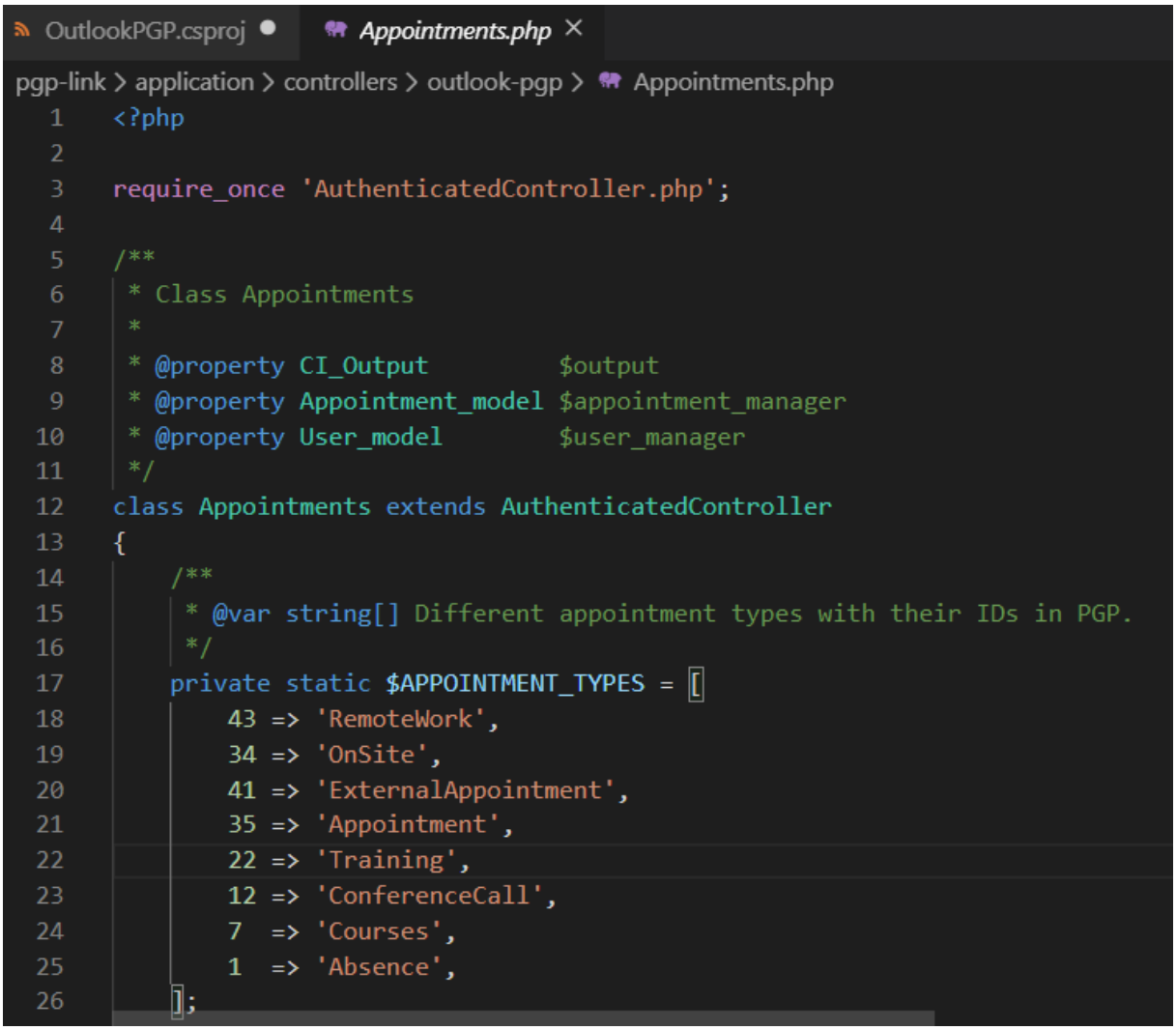
[Zito et al., 06] A. Zito, Z. Diskin, J. Dingel. "Package Merge in UML 2: Practice vs. Theory?". Proc. of the 9th International Conference on Model Driven Engineering Languages, Systems (MoDELS 2006), Genoa, Italy, October 2006.

ANNEXE A: WEBSERVICES PLUGIN SWAGGER DOCUMENTATION




GET /plugin/versions/flavors/version/{versionNumber}
POST /plugin/versions/flavors/version/{versionNumber}
GET /plugin/versions/flavors/{flavorId}
DELETE /plugin/versions/flavors/{flavorId}
Schemas
CompatibleVersion
ComputerSpec
CrashReport
File
NewCrashReport
NewVersion
NewVersionFlavor
UsageReport
Version
VersionFlavor
VersionUsageStatistics
WindowsVersionUsageStatistics

ANNEXE B: CODE SNIPPET FROM DEVELOPED WEB SERVICES



OutlookPGP.csproj ● AbstractApi.php ×

pgp-link > application > controllers > AbstractApi.php

```php
require(APPPATH . '/libraries/REST_Controller.php');

use Restserver\Libraries\REST_Controller;

require_once('utiles/PGPException.php');

/**
 * Class AbstractApi
 *
 * @property CI_Config $config
 */
abstract class AbstractApi extends REST_Controller
{
    function execut_post()
    {
        try {
            $actual_link = (isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] === 'on'
            log_message('INFO', "URL : " . $actual_link);
            log_message('INFO', "REQUEST : " . json_encode($this->post()));
            $this->db->trans_start();

            $this->integrity_verification();
            $this->busneiss_verification();
            $result = $this->execut();
            $this->post_execute();
```

```php
    protected function integrity_verification()
    {
        if ($this->post('aut_token') == null) {
            throw new PGPException(1, 'ABS_SERV_MISSING_TOKEN', 'The service requir
        }
    }

    protected function busneiss_verification()
    {
        if ($this->post('aut_token') != $this->config->item('authentication_token')
            throw new PGPException(1, 'ABS_SERV_MISSING_TOKEN',
                'Your session is exepered please make sure to reconnect');
        }
    }

    abstract protected function execut();

    protected function post_execute()
    {
    }
}
```