Année : 2012

Thèse N° : 6/ST2I

جامعة محمد الخامس السويسي
Université Mohammed V - Souissi

Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes

Centre d'Etudes Doctorales en Sciences des Technologies de l'Information et de l'Ingénieur

THESE DE DOCTORAT

CONTRIBUTION A LA COMPOSITION DE MODELES UML PAR UNE APPROCHE HYBRIDE : Cas du diagramme de classes

Présentée par

BENABDELLAH CHAOUNI Samia

Le 20 Décembre 2012

Formation doctorale: Informatique
Structure de recherche: Equipe Qualité des Architectures logicielles, développement et Intégration (AlQualsadi)

JURY

Professeur Rachid OULAD HAJ THAMI, Université Mohammed V-Souissi, ENSIAS,	**Président**
Professeur Ismail KASSOU Université Mohammed V-Souissi, ENSIAS,	**Directeur de thèse**
Professeur Mounia FREDJ Université Mohammed V-Souissi, ENSIAS,	**Co-Directeur de thèse**
Professeur Salma MOULINE, Université Mohammed V-Agdal, FSR,	**Co-Directeur de thèse**
Professeur Jean-Pierre GIRAUDIN, Université Pierre-Mendès-France, Grenoble	**Rapporteur**
Professeur Laïla KJIRI, Université Mohammed V-Souissi, ENSIAS,	**Rapporteur**
Professeur Ounsa ROUDIES, Université Mohammed V-Agdal, EMI,	**Rapporteur**

Dédicaces

A mes parents
A mes beaux parents
A mon mari
A mes frères et sœurs
A ma famille
A ma belle famille

Je leur dédie ce travail qui n'aurait pu aboutir sans leur soutien

Remerciements

Je tiens à exprimer ma profonde reconnaissance à mon encadrante de thèse, Mme Mounia FREDJ, Professeur à l'Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes, pour ses qualités tant scientifiques qu'humaines. Son soutien, sa disponibilité, sa patience ainsi que la collaboration étroite avec laquelle nous avons travaillé, ont été déterminants pour les résultats obtenus. J'exprime ici ma profonde gratitude à son égard et l'estime respectueuse que je lui porte.

J'exprime également ma profonde gratitude à ma co-encadrante de thèse, Mme Salma MOULINE, Professeur à la Faculté des Sciences de l'université Mohammed V Souissi, pour son aide précieuse, sa disponibilité et son écoute. Ce travail a grandement profité de son encadrement, de ses conseils et de ses qualités humaines.

Je tiens à remercier très vivement mon directeur de thèse M. Ismail KASSOU, Professeur à l'ENSIAS, d'avoir accepté ma candidature de thèse. Merci pour sa disponibilité et son écoute.

Je tiens aussi à remercier M. Rachid OULAD HAJ THAMI, Professeur à l'ENSIAS, pour m'avoir fait l'honneur d'accepter d'examiner mes travaux et de présider le jury de ma soutenance de thèse.

Mes remerciements vont également à mes rapporteurs, M. Jean-Pierre GIRAUDIN, Professeur à l'université Pierre Mendès France de Grenoble, Mme Laila KJIRI, Professeur à l'ENSIAS et Mme Ounsa ROUDIES, Professeur à l'Ecole Mohammadia d'Ingénieurs, qui m'ont fait le grand honneur de bien vouloir évaluer ma thèse à travers ce document. Je les remercie également pour les remarques et conseils prodigués.

Ma reconnaissance va aux professeurs et doctorants de l'Equipe Qualité des Architectures logicielles, développement et Intégration (AlQualsadi), pour leur disponibilité et le partage des connaissances. Merci pour les réunions d'équipe qui ont largement contribué à l'amélioration de mes travaux.

Mes plus vifs remerciements s'adressent, plus particulièrement, à Soumaya Slimani, Badr Elmir, Boutaina Chakir, Hatim Hamid et Fatima-Zahra Aazi pour les discussions enrichissantes que nous avons eues et les informations que nous avons échangées. Je tiens à remercier également les étudiants de l'ENSIAS qui, au cours de leurs projets, se sont intéressés à ma problématique de thèse et m'ont apporté de l'aide. Je cite notamment Youness Mrani et Ilyass Rahmoune.

Dans le but de valider les propositions de ma thèse, j'ai réalisé des expérimentations. Cette activité n'a pu être réalisée sans l'aide de plusieurs personnes. Je remercie tous les sujets participants aux expérimentations.

Je ne pourrais oublier mes amis, Nawal, Widad, Omar et Mohamed, et mes cousines Hasnaa, Doha et Ghita pour leurs encouragements. Je tiens à remercier tous ceux qui m'ont soutenue. Un immense Merci à mes parents pour leur soutien sans faille, leurs prières et leur confiance en moi, à ma belle-famille pour ses encouragements, à mes frères et sœur Marouan, Andrea, Imane, Karim, et Mohamed, à toute ma famille, à tous mes amis proches ou éloignés, pour leurs encouragements et leurs marques d'affection.

Un grand merci à ma sœur Safaa, dont la présence à mes côtés lors de la période de rédaction a été capitale. Enfin, un merci particulier à Mehdi, mon mari, pour sa patience et son soutien dans les moments difficiles, pour toutes les concessions qu'il a dû faire et toute l'aide qu'il a su m'apporter.

Résumé

La composition de modèles est l'un des défis de la modélisation des systèmes logiciels. Les modèles sont généralement construits indépendamment les uns des autres. Par conséquent, les hétérogénéités entre les modèles, de différents types (syntaxique, sémantique, structurel local et structurel global), peuvent engendrer des conflits (problèmes et incohérences) pouvant apparaître dans le modèle résultat. En effet, des éléments jugés différents mais qui sont réellement équivalents, créent des éléments redondants dans le modèle résultat ; ou inversement, des éléments jugés équivalents mais qui sont réellement différents, créent une perte d'information dans le modèle résultat.

A cet effet, cette thèse propose une approche de composition de modèles, spécifiquement les diagrammes de classes UML, permettant d'avoir un modèle résultat sans conflits. Cette approche, étant hybride, intègre ainsi tous les aspects - sémantique, syntaxique, structurel local et global- afin d'augmenter la probabilité d'avoir de réelles correspondances et de réelles différences. Nous avons utilisé les ontologies afin de traiter l'aspect sémantique de la comparaison de modèles. Nous proposons un système de composition Co-Models à base de règles syntaxique, sémantique et structurelle. Les règles ont été proposées de manière informelle puis formelle. Notre approche a été implémentée sous Java et validée par deux cas d'application : un cas du domaine bancaire et un autre du e-gov. Enfin, des expérimentations ont été effectuées dans l'objectif d'évaluer le résultat d'une composition de modèles par notre système.

MOTS-CLEFS

Composition de modèles, Fusion de modèles, Diagrammes UML, Comparaison sémantique, Ontologie, OWL

Abstract

Models composition is one of the challenges of software systems modeling. Models were usually created independently, and accordingly present many types of heterogeneity (syntactic, semantic, local structural and global structural). These heterogeneities cause conflicts when composing models, and may introduce problems and inconsistencies in the result model. Indeed, when composing models, considering two elements as different when they are equivalent will introduce redundancy. Worst, considering two elements as equivalent when they are different will result on information lost.

The purpose of this work is to present an approach of models composition, specifically UML class diagrams, against conflicts. This hybrid approach is thus integrates all aspects - semantic, syntactic, structural local and structural global - in order to increase the probability of detecting the right connections and right differences. We used ontologies to treat the semantic aspect of the model comparison. We propose a syntactic, semantic and structural rule-based system called Co-Models. The rules have been proposed informally and formally. Our approach has been implemented in Java. It has been validated against two real life use cases from different domains: banking and e-government. Finally, experiments were carried out to evaluate the result of models composition using our approach.

KEYWORDS

Models composition, Models merging, UML diagrams, Semantic comparison, Ontology, OWL

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

AMW	Atlas Model Weaver
AOM	Aspect-Oriented Modelling
ATL	Atlas Transformation Language
CIM	Computation Independent Model
DAML	Darpa Agent Markup Language
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DOE	Differential Ontologies Editor
DTD	Document Type Definition
ECL	Epsilon Comparaison Language
EMF	Eclipse Modeling Framework
EML	Epsilon Merging Language
ETL	Epsilon Transformation Language
GEF	Graphical Editing Framework
GIB	Gestion intégrée de budget
GID	Gestion Intégrée de la Dépense
GIR	Gestion Intégrée de la Recette
GMF	Graphical Modeling Framework
IDM	Ingénierie Dirigée par les Modèles
KR	*Knowledge Representation*
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MEF	Ministère de l'Economie et des Finances
MOF	Meta- Object Facility
OCL	Object Constraint Language
ODM	Ontology Definition Metamodel
OIL	Ontology Inference Layer
OMG	Object Management Group
OWL	Web Ontology Language
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query View Transform
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SHOE	Simple HTML Ontology Extension
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VUML	View based Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XOL	XML-based Ontology exchange Language

Chapitre **1**

Introduction générale

1. Introduction générale

1.1. Contexte général et problématique

Le génie logiciel est un domaine de recherche qui a pour objectif de répondre à un problème qui s'énonçait en deux constatations : d'une part le logiciel n'était pas fiable, et d'autre part, il était difficile de réaliser, dans les délais prévus, des logiciels satisfaisant leur cahier des charges. L'importance d'une approche méthodologique s'est imposée à la suite de la crise de l'industrie du logiciel à la fin des années 1970. Cette crise était principalement due à l'augmentation des coûts, les difficultés de maintenance et d'évolution, la non fiabilité, le non-respect des spécifications et le non-respect des délais.

Pour apporter une réponse à tous ces problèmes, le génie logiciel s'intéresse particulièrement à la manière dont le code source d'un logiciel est spécifié puis produit. Ainsi, le génie logiciel touche au cycle de vie des logiciels : l'analyse du besoin, l'élaboration des spécifications, la conceptualisation, le développement, la phase de test et la maintenance.

La phase de conceptualisation consiste à créer une représentation simplifiée d'un problème, à travers les modèles. C'est une activité importante dans le cycle de vie du logiciel. Les avantages de la conceptualisation comprennent l'évaluation précoce de la faisabilité technique, l'exactitude et l'exhaustivité des exigences, la gestion de la complexité ainsi que la réutilisation [Booch, 1994], [Coleman et al., 1994], [Cook et al., 1994], [Rumbaugh et al. 1991], [Shlaer et al., 1988].

De nos jours, face à la complexité des systèmes logiciels, la construction d'un modèle reste une tâche difficile. Pour faire face à cette complexité, les concepteurs ont très souvent recours à la décomposition puis la composition. En effet, la gestion, voire la diminution de la complexité, est une problématique récurrente à tout raisonnement scientifique. La décomposition d'un problème en sous-problèmes est la clé pour comprendre une situation donnée et proposer des solutions. Lorsque qu'une solution est disponible pour chaque sous-problème, on produit une représentation globale de la solution sur laquelle on peut raisonner.

Les travaux de cette thèse s'orientent dans ce sens. Nous nous intéressons, particulièrement, à la problématique de la composition de modèles UML et plus précisément la composition de diagrammes de classes UML. Nous avons choisi UML en tant que standard défini par l'Object Management Group (OMG). De plus, c'est le langage le plus utilisé dans les laboratoires et les industries du génie logiciel.

La composition de modèles est définie comme une opération qui combine deux ou plusieurs modèles (modèles sources) en un seul (modèle cible) [Didonet et al., 2006]. Elle suscite de nombreux travaux dans différents domaines d'application. Notre problématique concerne l'ensemble de ces domaines d'application et peut s'appliquer à :

- **La conceptualisation collective : approche de conception par composition**

Dans un contexte de complexité croissante des systèmes d'information, et plus particulièrement des systèmes logiciels, la conceptualisation collective, effectuée entre différentes équipes réparties à la fois géographiquement et temporellement, devient une activité majeure du processus de développement,

notamment en la phase de conception. Plusieurs modèles métier sont produits et doivent être composés pour obtenir un modèle métier global cohérent [Haddar et al., 2002], [Hacene, 2003] et [Brunet et al., 2006], [Mhiri, 2007].

- **La modélisation orientée points de vue**

Afin de gérer la complexité des systèmes logiciels, la plupart des approches reposent sur le principe de décomposition du système en un ensemble d'unités logicielles de complexité moindre. Parmi ces approches, certaines permettent d'analyser et de concevoir un système selon les points de vue des acteurs, interagissant avec ce système, c'est-à-dire concevoir, de façon décentralisée, un modèle UML pour chaque point de vue appelé modèle partiel, et, ensuite, produire un modèle par composition des modèles partiels [Anwar et al., 2007a], [Sabetzadeh et al., 2007].

- **Le processus de développement logiciel dirigé par les cas d'utilisation**

Des méthodologies de développement logiciel, basées sur UML, proposent une approche où le processus de développement est dirigé par les cas d'utilisation (Use Case Driven software development process). Cela veut dire que tous les artefacts (incluant les modèles d'analyse et de conception, leur implémentation et les spécifications des tests associés) ont des liens de traçabilité avec les cas d'utilisation. Lorsque le modèle des cas d'utilisation a de nombreux cas d'utilisation, la gestion de traçabilité, entre chaque cas d'utilisation et les éléments correspondants dans le diagramme de classes qui en résulte, peut être une tâche difficile. Dans ce scénario, l'idée est de travailler avec chaque cas d'utilisation, séparément, pour avoir son diagramme de classes partiel (qui est un morceau du diagramme de classes résultant). Les modèles partiels doivent, donc, être composés pour obtenir le modèle résultat [Boronat et al., 2007].

- **La modélisation orientée aspect**

Les fonctionnalités d'une application, appelées préoccupations transversales, deviennent au fur et à mesure que l'application évolue, difficiles à identifier, à comprendre et à faire évoluer au niveau des modèles de conception. La modélisation orientée aspects (AOM : Aspect-Oriented Modelling) vise à résoudre ce problème en proposant de séparer le modèle en deux parties : une partie fonctionnelle qui encapsule le modèle métier, noyau de l'application, et une partie secondaire qui regroupe les fonctionnalités transversales disséminées.

L'opération de composition centrale de l'AOM est le tissage. Le tissage consiste à composer les modèles d'aspects, représentant les préoccupations non fonctionnelles (préoccupations transverses), avec le modèle modélisant le cœur fonctionnel de l'application [Reddy et al., 2006][France et al., 2007].

- **La réutilisabilité des patrons**

En génie logiciel, un pattern est une solution réutilisable pour résoudre des problèmes récurrents de conception. Traditionnellement, les patrons appropriés sont identifiés par les concepteurs de logiciels, pour être ensuite appliqués à un problème de conception logiciel récurrent. Toutefois, les patterns existants doivent être correctement composés avec les modèles du software (diagrammes de classes UML) existants [Rungworawut, 2010].

- **La gestion des versions des modèles dans le développement basé sur les modèles**

La gestion des versions est une activité importante dans tout grand projet de développement de logiciels. Les produits (logiciels) peuvent changer continuellement. Selon le principe MDA, les modifications doivent se faire au niveau des modèles. Chaque concepteur modifie le modèle initial pour l'améliorer selon son besoin et son espace de travail. Plusieurs versions du modèle initial sont créées et doivent alors être composées dans un seul modèle résultat contenant tous les changements proposés par tous les concepteurs [Alanen et al., 2003] et [Bartelt, 2008].

La composition de modèles peut éventuellement être utilisée aussi dans les cas suivants :

- **La construction d'un modèle de domaine**

La construction d'un modèle de domaine UML peut se faire par la composition de modèles existants appartenant (ou non) à des méta-modèles hétérogènes (composition de diagrammes de classes UML, schémas de base de données relationnelles, ontologies de tâches, etc.) [Mhiri, 2007].

- **L'évolution interne d'une organisation**

Le domaine des systèmes d'information s'est profondément transformé, ces dernières années, sous l'influence de l'évolution des organisations. Cette évolution peut être d'origine interne, engendrée par la restructuration des organisations, la création de nouvelles filiales ou de nouvelles implantations géographiques ou encore l'évolution de l'activité de l'entreprise. Conséquence de ces facteurs, de nouveaux systèmes d'information avec leurs modèles métiers se créent, qu'il convient de composer avec les modèles existants [Izza, 2006].

- **La fusion de deux organismes**

L'évolution des organisations peut être aussi d'origine externe, expliquée par l'évolution de deux organisations ayant le même domaine d'activité et qui veulent fusionner. Dans ce cas, il faut composer leurs systèmes d'informations et plus précisément leurs modèles [Izza, 2006].

- **La composition des applications dans le contexte MDA**

La composition des applications est une tâche difficile du fait de la complexité des logiciels et/ou l'hétérogénéité des langages du code source (java, C, C++, etc.). Selon l'approche MDA, les modèles des exigences et de conception qui représentent l'application (e.g. diagramme de classes) sont séparés du modèle du code final de l'application. De ce fait, il serait intéressant et plus facile de réaliser la composition à un niveau d'abstraction plus élevé, c'est-à-dire au niveau des modèles indépendants de la plate-forme, et de générer automatiquement à partir du modèle résultat, le modèle spécifique à la plate-forme et donc l'application résultante des applications sources [Bouzitouna et al., 2004].

En s'appuyant sur les définitions proposées par [Haddar et al., 2002], [Jing et al., 2005], [Bézivin et al., 2006], [Boronat et al., 2007] et [Rungworawut, 2010], nous définissons la composition de modèles comme la combinaison de deux modèles sources pour créer un modèle cible, contenant les informations appartenant aux deux modèles initiaux, et sans redondance. Elle se compose de trois principales opérations :

- La première est l'opération de **comparaison** (appelée aussi matching). Elle prend en entrée les modèles sources, les compare afin de trouver les équivalences entre leurs éléments. Elle

produit en sortie le modèle de correspondances (appelé aussi mappings) qui contient les liens entre les éléments des deux modèles et les éléments non équivalents (appelés éléments isolés).
- La deuxième opération est celle de la **fusion**. Elle prend en entrée le modèle de correspondances et combine les éléments équivalents.
- Enfin, la troisième opération est celle de la **translation**. Elle prend en entrée les éléments isolés, et les translate (copie).
  Les deux dernières opérations (fusion et translation) produisent en sortie le modèle résultat.

Les modèles à composer ont été, a priori, construits indépendamment les uns des autres. De ce fait, des conflits peuvent apparaître lors de la composition, à cause d'hétérogénéités qui peuvent exister entre les modèles. Ces hétérogénéités peuvent être de différents types :

- Syntaxiques : c'est le cas lorsque les mêmes éléments des deux modèles sources sont représentés par des noms différents (par exemple, deux classes « Compte » et « CompteBancaire »).
- Sémantiques : nous retrouvons ce type d'hétérogénéité lorsque des éléments ayant le même sens, sont représentés différemment (par exemple, deux attributs « poste » et « fonction »).
- Structurelles locales : il s'agit d'éléments qui ont le même nom ou le même sens, mais pas le même contenu (par exemple des classes qui ont le même nom mais pas les mêmes attributs et opérations).
- Structurelles globales : c'est le cas lorsque des éléments ont le même nom ou le même sens mais pas le même voisinage (par exemple, deux classes qui ont le même nom mais pas les mêmes associations, relations de généralisation et classes qui les entourent).

Conformément aux hétérogénéités définies ci-dessus, la comparaison inclut les aspects syntaxique, sémantique, structurel local et structurel global.

Notre analyse de l'existant a montré qu'aucune approche ne couvre complètement tous les cas possibles des types de comparaison. Ces approches peuvent ne pas détecter, par exemple, l'équivalence entre « CompteBancaire » et « Compte », « Dirigé » et « EstDirigéPar », ou encore « marié » et « célibataire ». De plus, tous les travaux de composition de modèles ne combinent pas tous les types de comparaison, ce qui diminue les chances de trouver de « vraies » correspondances ou de « vraies » différences, c'est-à-dire deux éléments des deux modèles réellement équivalents, ou deux éléments réellement différents.

Ces limites peuvent engendrer des conflits (problèmes d'incohérences) pouvant apparaître dans le modèle résultat. En effet, des éléments jugés différents (lors de la comparaison) mais qui sont réellement équivalents, créent des éléments redondants dans le modèle résultat ; ou inversement, des éléments jugés équivalents mais qui sont réellement différents, créent une perte d'information dans le modèle résultat.

La composition de modèles est donc une opération relativement complexe qui est encore loin d'être complètement résolue. L'objectif principal de notre thèse consiste à proposer une approche de composition de modèles (en particulier de diagrammes de classes UML), qui permet d'avoir une représentation du système résultat sans existence de conflits. La problématique peut être résumée sous la forme des deux questions suivantes :

- Comment réaliser une comparaison afin de détecter toutes les correspondances et trouver de vraies correspondances et de vraies différences ?
- Comment réaliser la fusion et la translation des éléments présentés différemment dans les modèles sources ?

Afin de répondre à cette problématique, nous proposons une approche qui introduit la sémantique dans la comparaison de modèles, afin d'approcher le raisonnement humain. Cette approche couvre tous les éléments des types de comparaison afin de détecter toutes les correspondances, et combine tous les types de comparaison (syntaxique, sémantique, structurel global et structurel local) afin d'augmenter la probabilité d'avoir de réelles correspondances et de réelles différences.

En ce qui concerne la phase de fusion/translation, nous avons élaboré un ensemble de directives permettant d'indiquer les actions à appliquer pour décider comment les éléments vont apparaître dans le modèle résultat. Par exemple, comment des éléments synonymes, ou des éléments en relation de polysémie, ou encore deux associations avec une composition différente vont apparaître dans le modèle résultat.

1.2. Contributions

Le travail, présenté dans ce mémoire, propose une approche de composition hybride intégrant tous les aspects : sémantique, syntaxique, structurel local et global. De plus, cette approche offre des directives de composition pour déterminer comment les éléments vont apparaître dans le modèle résultat.

Nous proposons un système de composition, que nous baptisons « Co-Models » pour **Co**mposition of **Models,** qui prend en entrée deux modèles et produit en sortie le modèle résultat. Il comporte deux sous-systèmes :

- Un système de comparaison de modèles qui prend en entrée deux modèles et produit en sortie (1) le mapping entre les éléments des modèles et (2) les éléments isolés. C'est un système à base de trois types de règles : celles qui comparent syntaxiquement les éléments de modèles, celles qui les comparent sémantiquement et celles qui les comparent structurellement. Afin de prendre en compte l'aspect sémantique, nous avons eu besoin de stratégies s'appuyant sur des propriétés de nature sémantique. Pour cela, notre système se réfère à un dictionnaire de synonymes et une ontologie de domaine. Cette dernière permet de fournir des informations sémantiques pertinentes et de prise de décision lors de la comparaison, telles que l'équivalence, la disjonction ou l'inverse de deux concepts. Pour ce qui est de l'aspect syntaxique des éléments, nous utilisons des dictionnaires d'acronymes et d'abréviations.
- Un système de fusion/translation qui prend en entrée le mapping entre les éléments de modèles (résultat du système de comparaison), la liste des éléments isolés et les deux modèles sources, et produit en sortie le modèle résultat. Ce système à base de règles de fusion et de translation permet de fusionner les éléments en correspondance et de translater (rajouter) les éléments isolés. Ces règles contiennent aussi des directives qui permettent de déterminer comment vont apparaître dans le modèle résultat, les éléments présentés de manières différentes dans les modèles sources sans que le modèle cible ne contienne de conflits.

Nous avons également élaboré un processus de composition visant à décrire les étapes de notre approche de composition que nous modélisons dans des diagrammes d'activités.

Les règles de comparaison, fusion et de translation sont définies en logique des prédicats du 1er ordre. Pour cela, nous avons eu besoin d'exprimer le modèle UML (représentant les éléments en entrée et en sortie du système Co-Models), l'ontologie de domaine et d'autres ressources (représentant les références du système) en logique des prédicats.

Ceci nous a amené à définir des transformations de modèles, à savoir la transformation d'un diagramme de classes UML et d'une ontologie, en un modèle logique.

Une implémentation de notre prototype a été réalisée en langage Java sous l'environnement de développement Eclipse, du fait qu'il dispose de bibliothèques permettant de gérer les ontologies.

Nous avons validé notre approche par deux études de cas. La première est la fusion de deux organismes. Il s'agit de la fusion de modèles des systèmes d'information de deux banques. La deuxième étude de cas s'intéresse à la composition d'applications par composition de modèles. Il s'agit d'une étude de cas du domaine du e-gov au Maroc, dans laquelle nous proposons la création de l'application de la Gestion Intégrée des Recettes (GIR) au sein du Ministère de l'Economie et des Finances, par la composition des modèles d'applications existantes. L'objectif est d'appliquer notre approche de composition de modèles, dans un cadre réel.

Enfin, des expérimentations ont été effectuées visant deux principaux objectifs. Le premier est d'évaluer dans quelle mesure notre approche se rapproche du raisonnement humain. Le second objectif est de connaître les avis des utilisateurs sur Co-Models.

1.3. Plan de la thèse

Ce rapport est organisé en 6 chapitres (cf. Figure 1-1). Le second chapitre est dédié d'une part, aux notions de modèle, méta-modèle, transformation de modèles qui déterminent les outils et éléments nécessaires à l'application du paradigme de l'Ingénierie Dirigée par les Modèles, et d'autre part, au concept de l'Ontologie.

Le chapitre 3 présente un état de l'art de la composition de modèles. Afin d'illustrer les hétérogénéités pouvant exister entre les modèles, un exemple y est présenté. Cet état de l'art est suivi d'une analyse synthétique des approches existantes, qui a permis de les évaluer et d'en dégager les limites.

Le chapitre 4 fait l'objet de nos contributions. Il est scindé en deux sous-parties. La première partie est consacrée à la description de notre système de composition Co-Models et du processus de composition de modèles. La seconde partie propose notre approche de composition présentée d'abord de manière informelle ensuite formellement. Elle détaille la présentation des différents méta-modèles utilisés (UML, OWL et logique des prédicats), les transformations des modèles (UML et OWL) vers le modèle en logique des prédicats, et, enfin, les règles de composition en langage formel.

Le chapitre 5 concerne la mise en œuvre et la validation de notre approche. Il est constitué de trois parties. Tout d'abord, deux études de cas des domaines bancaire et du e-gov sont présentées. La première étude de cas reprend l'exemple bancaire du chapitre 3, et complète le processus de composition des modèles sources. Ensuite, le prototype Co-Models est explicité afin de décrire son

fonctionnement. Enfin, les résultats et l'analyse des expérimentations, servant de support de validation, sont décrits dans la dernière partie.

Le chapitre 6 clôture ce document en présentant une revue des travaux réalisés ainsi qu'une discussion de nos perspectives.

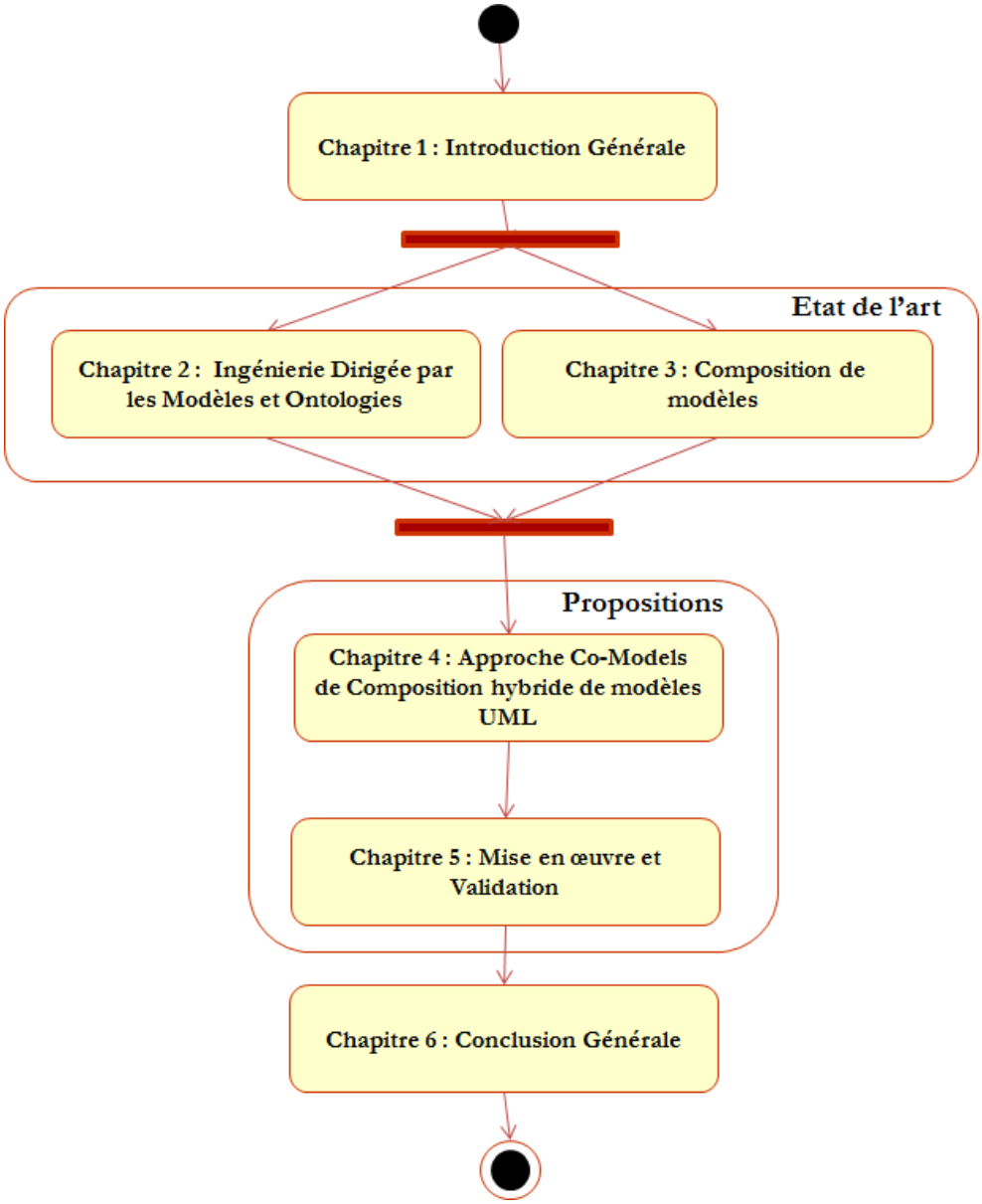


Figure 1-1 Plan de la thèse

Chapitre **2**

Ingénierie Dirigée par les Modèles et Ontologies

2. Ingénierie Dirigée par les Modèles et Ontologies

Introduction

Nos travaux s'intéressent à la composition de modèles UML et plus précisément des diagrammes de classes. Notre approche s'appuie sur les ontologies afin de résoudre la problématique de la composition sémantique de modèles. Nous avons eu besoin de faire référence à un modèle riche sémantiquement qui décrit le vocabulaire d'un domaine particulier et fourni des informations pertinentes lors de la composition.

Pour répondre au besoin de formaliser notre solution, nous avons procédé par transformer les modèles UML et OWL, vers des modèles en logique des prédicats. Notre réflexion et les développements associés s'appuient ainsi, d'une part, sur les notions de modèle, méta-modèle et transformation de modèles et, d'autre part, sur le concept d'Ontologie.

Ce chapitre a pour objectif de définir le cadre général de cette thèse. Une première partie est dédiée à l'Ingénierie Dirigée par les Modèles. Elle a pour objectif de présenter ses principes de base. Une seconde partie présente les constituants des ontologies, leurs classifications, leurs langages d'expression ainsi que leurs éditeurs. Ceci permettra de définir le choix du type de l'ontologie, du langage et de l'éditeur de l'ontologie utilisés dans nos travaux.

2.1. Ingénierie dirigée par les modèles

L'Ingénierie Dirigée par les Modèles (IDM), ou Model Driven Engineering (MDE) en anglais, a permis plusieurs améliorations significatives dans le développement de systèmes complexes en permettant de se concentrer sur une préoccupation plus abstraite que la programmation classique, à savoir le modèle. Il s'agit d'une forme d'ingénierie générative dans laquelle tout ou partie d'une application est engendrée à partir de modèles. L'idée centrale est d'utiliser autant de modèles, à différents niveaux d'abstraction, que la description d'un système le nécessite. L'IDM place les modèles au cœur du processus de développement des applications. Ceci augmente la pérennité du savoir métier de l'application [Combemale, 2008].

Nous présentons dans cette section les éléments nécessaires à l'application de l'IDM : modèle, méta-modèle, langage de modélisation et transformation de modèles.

2.1.1. Concepts de l'IDM

Il n'existe pas, à ce jour, de définition universelle de ce qu'est un modèle (tout comme il n'existe pas de définition universelle de ce qu'est un objet, un aspect, etc.). La définition la plus complète, à notre avis, est celle proposée par Bézivin et Gérbé dans [Bézivin et al., 2001] :

*"A **model** is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual **system**."*

Comme le suggère cette définition, le ***modèle*** et le ***système étudié*** ont deux rôles complémentaires, basés sur une unique relation liant un modèle au système qu'il modélise.

L'objectif premier de l'IDM, est d'avoir un modèle productif, pouvant être exploitable par une machine et donc que le langage dans lequel ce modèle est exprimé soit clairement défini. La définition du langage de modélisation se fait par un méta-modèle. La notion de méta-modèle est au cœur même de l'IDM. Bien que certains aspects relatifs à cette notion restent sujets à controverses, nous pouvons noter, néanmoins, une convergence dans les définitions que l'on trouve dans la littérature :

*"A **meta-model** is a model that defines the language for expressing a **model**"* [MOF, 2002].

*"A **meta-model** is a specification model for a class of SUS (System Under Study) where each SUS in the class is itself a valid model expressed in a certain modelling language"* [Kleppe et al., 2003].

Nous avons explicité le principe et concepts de l'IDM. Nous présentons, dans ce qui suit, l'architecture dirigée par les modèles – ou Model Driven Architecture (MDA) qui est la spécification de référence de l'OMG mettant en œuvre les principes de l'IDM.

2.1.2. Model Driven Architecture

En novembre 2000, l'OMG[1] (Object Management Group) a rendu publique sa nouvelle orientation dans un document intitulé MDA. L'initiative MDA repose sur un changement de paradigme important dans le domaine du génie logiciel. La rapidité de cette évolution s'explique, d'une part, par la nécessité de découpler les éléments que l'on peut considérer comme stables dans les logiques métiers, de l'évolution des supports technologiques informatiques, et, d'autre part, par l'urgence de stopper l'empilement des technologies.

Le principe clé de MDA consiste à utiliser les modèles aux différentes phases du cycle de développement d'une application. Plus précisément, MDA préconise l'élaboration de modèles d'exigence (CIM : Computation Independent Model), d'analyse et de conception (PIM : Platform Independent Model) et de code (PSM : Platform Specific Model).

L'objectif majeur de MDA est l'élaboration de modèles pérennes, indépendants des détails techniques des plates-formes d'exécution (J2EE, .Net, PHP ou autre), afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité.

D'une façon macroscopique, le MDA peut se résumer à l'élaboration de modèles indépendants de plates-formes (PIM) et à la transformation de ceux-ci en modèles dépendants de plates-formes (PSM). La transformation utilisée pour revenir à un modèle indépendant de plate-forme (PIM) à partir d'un modèle spécifique de plate-forme (PSM) est une opération de rétro-ingénierie (reverse engineering). Ces transformations sont nécessaires pour permettre l'intégration d'applications existantes dans le processus MDA [Bézivin et al., 2002].

Les deux principaux artefacts de l'Ingénierie Dirigée par les Modèles étant les modèles et les transformations de modèles, nous présentons, dans la section suivante, les principes de la transformation de modèles.

2.1.3. Transformation de modèles

Un des éléments clé de l'IDM consiste à pouvoir rendre opérationnels les modèles à l'aide de transformations. Cette notion est au centre de l'approche MDA. D'un point de vue général, on

[1] http://www.omg.org/

appelle transformation de modèles tout programme dont les entrées et les sorties sont des modèles. La transformation de modèles est basée sur la méta-modélisation, (cf. Figure 2-1) et est définie par un modèle de transformation Mt (conforme à son méta-modèle MMt) qui permet de transformer un modèle Ma (conforme à son méta-modèle MMa) en un modèle Mb (conforme à son méta-modèle MMb). Le MMt défini le langage de transformation utilisé tel que ATL ou QVT [Bézivin, 2004].

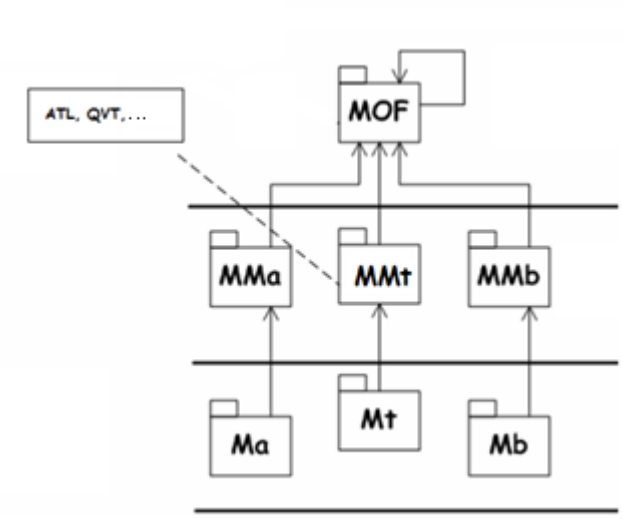


Figure 2-1 Transformation de modèles basée sur la méta-modélisation [Bézivin, 2004]

Le processus de transformation de modèles, basé sur la méta-modélisation, est composé de trois étapes : (1) la définition des règles de transformation, (2) l'expression des règles de transformation et (3) l'exécution des règles de transformation. Nous détaillons, dans la suite, ces trois étapes.

- **Définition des règles de transformation**

Etant donné un modèle source écrit dans un langage *ls* (tel que UML), et un modèle cible écrit dans un langage *lc* (tel que java), il s'agit, dans cette étape, d'élaborer une mise en correspondance des concepts de ls avec ceux de lc (par exemple une classe UML correspond à une ou plusieurs classes java). La méta-modélisation est utilisée pour mettre en place une base de règles exhaustive et générique.

Les règles de transformation sont établies entre le méta-modèle source et le méta-modèle cible. Le processus de transformation prend alors en entrée un modèle conforme au méta-modèle source et produit en sortie un ou plusieurs autre(s) modèle(s) conforme(s) au méta-modèle cible, en utilisant les règles ainsi établies.

- **Expression des règles de transformation**

Les règles de transformation, exprimées dans un langage de transformation, forment le modèle de transformation.

Après la définition des règles, ces dernières sont exprimées dans un langage de transformation. On peut citer ATL (Atlas Transformation Language) [ATL, 2006], ou QVT (Query View Transform) [QVT, 2011] un standard défini par l'OMG. Une comparaison de ces langages est disponible dans [Jouault, 2007].

- **Exécution des règles de transformation**

Une fois spécifiées et exprimées, les règles requièrent un moteur d'exécution pour être exécutées. Ce moteur prend, en entrée, le modèle et le méta-modèle sources, le méta-modèle cible, ainsi que le modèle de transformation et son méta-modèle (représentant la grammaire du langage de transformation). Il produit en sortie le modèle cible.

Nous avons mis en avant les principes généraux de l'Ingénierie Dirigée par les Modèles, l'Architecture Dirigée par les Modèles ainsi que la transformation de modèles. Nous introduisons, dans ce qui suit, les ontologies, un concept clé utilisé dans nos travaux.

2.2. Ontologies

2.2.1. Définitions

L'ontologie a été définie par Aristote comme étant la science de l'Être. Le terme lui-même date du XVIIe. Bien que des débats préexistent, la naissance de l'ingénierie des connaissances, puis l'essor du web sémantique, placent la définition, la construction et l'utilisation d'ontologie au centre de nombreux travaux. On parle plus souvent d'ontologies (au pluriel) afin de refléter les multiples facettes que recouvre cette appellation. Depuis le début des années 90, l'ingénierie des connaissances a grandement contribué à diffuser le terme « ontologie ». [Gruber, 1993] reprend ce terme, précédemment utilisé par [Sowa, 1984], et l'introduit en informatique. Il ne fait alors pas référence à Aristote ou au domaine de la philosophie, mais bien au domaine de la sémantique, au sens où l'entendent les théoriciens de la représentation des connaissances.

Plusieurs auteurs, comme [Guarino, 1996] ou encore [Dameron, 2003], passent en revue les différentes définitions de la littérature afin d'examiner le type de représentation des connaissances, dénoté par le terme ontologie. Gruber [Gruber, 1993] propose une première définition et introduit la notion de conceptualisation : « une ontologie est une spécification partagée d'une conceptualisation ». Cette définition permet de nombreuses interprétations et demande à être clarifiée et précisée. Les travaux menés par [Guarino, 2000] permettent de définir cette notion de conceptualisation. Il s'agit, d'une part, de préciser la notion de domaine d'application ou de discours, en distinguant parmi les entités peuplant un domaine, les individus et les propriétés, et d'autre part, de préciser quelles sont les propriétés essentielles.

La définition proposée par [Bachimont, 2000] s'appuie directement sur la représentation des connaissances en tant que domaine d'étude, et sur la formalisation en tant que moyen de mise en œuvre. Dans ce contexte, l'ontologie réconcilie la dimension sémantique et la dimension syntaxique nécessaires aux langages formels informatisés : « Définir une ontologie pour la représentation des connaissances, c'est définir, pour un domaine et un problème donnés, la signature fonctionnelle et relationnelle d'un langage formel de représentation et la sémantique associée ».

[Charlet, 2002] propose une définition rigoureuse et affinée de ce qu'est une ontologie. Une ontologie implique ou comprend une certaine vue du monde par rapport à un domaine donné. Cette vue est souvent conçue comme un ensemble de concepts, leurs définitions et leurs interrelations, appelés une conceptualisation. Une ontologie peut prendre différentes formes mais elle inclura nécessairement un vocabulaire de termes et une spécification de leur signification. Une ontologie est une spécification rendant partiellement compte d'une conceptualisation.

Enfin, [Roche, 2005] a donné une définition générique et simple. Une ontologie est une conceptualisation d'un domaine, à laquelle sont associés un ou plusieurs vocabulaires de termes. Les concepts se structurent en un système et participent à la signification des termes. Une ontologie est définie pour un objectif donné et exprime un point de vue partagé par une communauté. Une ontologie s'exprime dans un langage (représentation) qui repose sur une théorie (sémantique) qui garantit des propriétés de l'ontologie en termes de consensus, cohérence, réutilisation et partage.

Nous retenons cette dernière définition car elle englobe et résume les définitions précédentes. Dans ce qui suit, nous présentons, de manière non exhaustive, les éléments qui constituent l'ontologie.

2.2.2. Constituants d'une ontologie

Les ontologies rassemblent les connaissances propres à un domaine donné. En représentation des connaissances, ces ontologies existent sous la forme de concepts et de relations, et permettent d'en fixer la sémantique. L'hypothèse de départ est donc qu'il existe des objets individuels qui peuvent être énumérés. Les concepts et les relations de l'ontologie sont organisés sous une forme hiérarchique qui admet une relation de subsomption. Nous présentons, ci-dessous, les différents composants de l'ontologie, considérée en tant qu'objet informatique.

2.2.2.1. Concepts

Les connaissances portent sur des objets auxquels on fait référence à travers des concepts, également appelés classes dans certains travaux [Uschold et al., 1995]. Par exemple, « CompteBancaire » est un concept d'une ontologie.

2.2.2.2. Propriétés

Les propriétés sont des caractéristiques attachées aux concepts (par exemple, la propriété « numéro » du concept « CompteBancaire »). Une des grandes difficultés de l'ingénieur des connaissances qui développe une ontologie, est de décider si une connaissance doit être modélisée sous la forme d'une propriété ou bien à l'aide d'une relation liant un autre concept. Une solution courante est de choisir la propriété, lorsque la valeur de la connaissance est un type tel qu'un entier ou une chaîne de caractères. Les valeurs que peut admettre une relation sont d'un genre plus complexe puisqu'il s'agit d'un autre concept présent dans l'ontologie.

2.2.2.3. Relations

Les relations représentent un type d'interaction entre les concepts d'un domaine. Elles lient les concepts primitifs (ou simples) entre eux pour construire des représentations conceptuelles complexes appelées concepts définis. Elles sont caractérisées par un terme (voire plusieurs), une signature qui précise le nombre d'instances de concepts que la relation lie, et l'ordre de ces concepts, c'est-à-dire la façon dont la relation doit être lue. Par exemple, la relation « Posséder » lie des instances du concept « Client » et des instances du concept «CompteBancaire », dans cet ordre. Des exemples de relations binaires sont : « observé-par », « associé-à», « qualifié-de », ou encore « connecté-à ». [Kassel, 2002] et [Guarino et al., 1995] formalisent les principales relations jugées utiles à la modélisation d'une ontologie : « instance-de », « sorte-de », « appartenance-à » et « dépendance ».

La relation de subsomption « est-un » (is-a), a un statut particulier, car elle structure la hiérarchie ontologique. Un concept C1 (concept père) subsume un concept C2 (concept fils) si toute propriété

sémantique de C1 est également une propriété sémantique de C2 et si C2 est plus spécifique que C1. (Exemple, il y a une relation de subsomption entre les concepts « Personne » et « Client »).

Nous avons défini les ontologies ainsi que leurs constituants. Dans ce qui suit, nous présentons les types d'ontologies les plus courants dans la littérature.

2.2.3. Typologies des ontologies

Nous reprenons, à cet effet, les travaux de [Mizoguchi et al., 1995], [Heijst et al., 1997] et [Guarino, 1998]. Cet état de l'art présente plusieurs typologies connues qui classifient les ontologies en fonction du niveau de formalisme du modèle, des objets qu'elles modélisent, et du degré de granularité des connaissances représentées.

2.2.3.1. Classification selon le degré de formalisme de la représentation

La classification faite par [Uschold et al., 1996] distingue les ontologies selon le type de langage utilisé, et donc, en fonction du degré de formalisme de la représentation :

- Les ontologies, hautement informelles, sont des ontologies écrites en langage naturel.
- Les ontologies semi-informelles utilisent un langage naturel structuré et limité.
- Les ontologies semi-formelles définissent les concepts dans un langage artificiel et défini formellement.
- Les ontologies, rigoureusement formelles, sont définies dans un langage contenant une sémantique formelle, des théorèmes et des preuves de propriétés.

2.2.3.2. Classification selon les objets que modélisent les ontologies

Comme le proposent [Gómez-Pérez et al., 2004], la classification peut également se faire en fonction des objets que modélisent les ontologies, pour répondre à un objectif précis. Nous trouvons les ontologies pour la représentation des connaissances, les ontologies génériques, les ontologies de domaine, les ontologies de haut niveau, les ontologies de tâches, ainsi que les ontologies d'application.

La Figure 2-2 montre les relations qui existent entre les quatre derniers types d'ontologie. Il s'agit de la relation « peut dépendre de ».

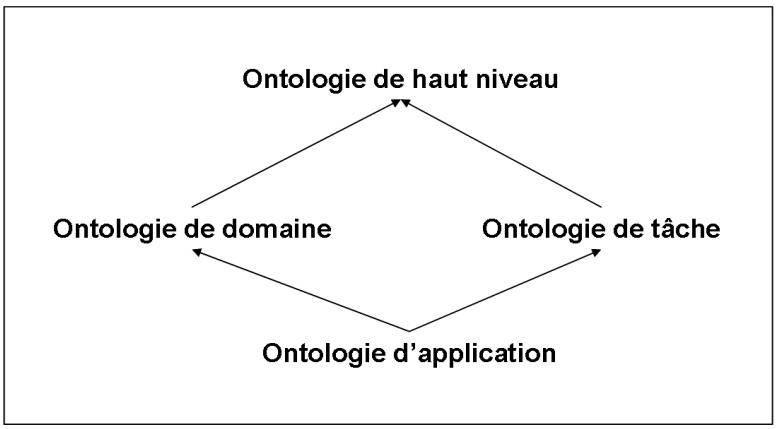


Figure 2-2 Classification des ontologies [Guarino, 1998]

- **Ontologies pour la représentation des connaissances [Heijst et al., 1997]**

Ce type d'ontologies regroupe les concepts utilisés pour formaliser les connaissances. Parmi les ontologies de représentation, on trouve des ontologies qui vont décrire les notions utilisées dans toutes les ontologies pour spécifier les connaissances, telles que les concepts, les relations, etc. La « Frame-Ontology » est un bon exemple d'ontologie de représentation. Elle définit, de manière formelle, les concepts utilisés principalement dans des langages à base de frames : classes, sous-classes, attributs, valeurs, relations et axiomes [Gruber, 1993]. Un autre exemple est l'ontologie « *Knowledge Representation* (KR) ». C'est une ontologie à visée universelle. Les concepts dans KR sont représentés par des prédicats unaires. De plus, un nouveau concept dans KR est la conjonction des prédicats unaires d'un niveau plus haut. Par exemple, le concept *Proposition* dans KR est la conjonction des deux concepts de plus haut niveau *Abstract* et *Relative*.

Les ontologies de représentation sont en fait indépendantes des différents domaines de connaissances, puisqu'elles décrivent des primitives cognitives communes aux différents domaines [Guarino et al., 1994].

- **Ontologies de domaine [Mizoguchi et al., 1995] [Heijst et al., 1997]**

Les ontologies de domaine sont construites sur un domaine particulier de la connaissance comme la physique, la mécanique, la chimie, la médecine et la modélisation d'entreprise. Elles rendent compte du vocabulaire d'un domaine spécifique au travers de concepts et de relations qui modélisent les principales activités, les théories et les principes de base du domaine en question. De nombreuses ontologies de domaine existent déjà, telles que MENELAS[2] dans le domaine médical, et TOVE (TOronto Virtual Entreprise)[3] dans le domaine de la modélisation et du manufacturing des entreprises. Ce type d'ontologies est bien souvent raccroché à des ontologies de haut niveau de conceptualisation.

- **Ontologies de haut niveau [Mizoguchi et al., 1995], [Guarino et al., 1995], [Guarino, 1997]**

Une distinction est établie entre les ontologies de domaine portant sur des concepts renvoyant à des objets matériels ou à des concepts d'assez bas niveau (c'est-à-dire n'offrant que des possibilités limitées de raffinement), et les ontologies portant sur des concepts de haut niveau. Ces dernières décrivent des notions générales comme les notions d'objet, de propriété, d'état, de valeur, de moment, d'événement, d'action et de cause à effet [Sowa, 2000]. Parmi ces ontologies, on cite souvent « Upper Cyc ». Elle contient presque 3000 termes relatifs aux concepts les plus généraux sur les connaissances humaines « Every concept one can imagine can be correctly linked into the Upper Cyc ontology ». Elle a été construite en une douzaine d'années. En théorie, les ontologies de haut niveau doivent pouvoir être reliées au sommet des ontologies de domaine.

- **Ontologies génériques [Mizoguchi et al., 1995] [Heijst et al., 1997]**

L'ontologie générique, appelée aussi méta-ontologie ou noyau d'ontologie, véhicule des connaissances génériques qui, bien que moins abstraites que celles modélisées dans l'ontologie de haut niveau, doivent être assez générales pour être réutilisées dans différents domaines. Elle organise des

[2] http://estime.spim.jussieu.fr/Menelas/
[3] http://www.eil.utoronto.ca/enterprise-modelling/tove/

connaissances factuelles ou des connaissances visant à résoudre des problèmes génériques d'un ou de plusieurs domaines. Ce type d'ontologies modélise des connaissances se rapportant aux choses, événements, temps, espace, causalité, etc. L'ontologie méréologique (Mereology Ontology) est un exemple classique d'ontologie générique [Borst, 1997].

- **Ontologies de tâches [Mizoguchi et al., 1995], [Guarino, 1998]**

L'ontologie de tâches décrit les connaissances portant sur des tâches et/ou des activités particulières. Ces ontologies fournissent un ensemble de termes au moyen desquels on peut décrire, au niveau générique, comment résoudre un type de problème. Selon [Mizoguchi et al., 1995], cette ontologie caractérise l'architecture computationnelle d'un système à base de connaissances qui réalise une tâche. On peut les assimiler à l'inventaire des rôles, tâches et méthodes utiles à la description de la résolution de problèmes. Les ontologies de tâches peuvent dépendre des ontologies de haut niveau.

- **Ontologies d'application [Heijst et al., 1997]**

Ce sont les ontologies les plus spécifiques, elles contiennent les connaissances requises pour une application particulière. Les ontologies d'application étendent et spécialisent les connaissances contenues dans l'ontologie de domaine et dans l'ontologie de tâche pour une application donnée. Selon [Maedche et al., 2001], les concepts dans l'ontologie d'application correspondent souvent aux rôles joués par les objets du domaine tout en exécutant une certaine activité. On peut citer, par exemple, Physsys qui a été construite pour assister des ingénieurs dans le développement d'applications concernant l'ingénierie de systèmes physiques dynamiques [Pim et al. 1997]. PhysSys exploite l'ontologie EngMath couvrant tous les aspects liés à la modélisation mathématique en ingénierie [Gruber et al., 1994].

2.2.3.3. Classification selon le niveau de granularité

La dernière classification, que nous présentons, est en fonction du niveau de granularité, c'est-à-dire du niveau de détails des objets de la conceptualisation. En fonction de l'objectif opérationnel, une connaissance plus ou moins fine du domaine est nécessaire, et des propriétés, considérées comme accessoires dans certains contextes, peuvent se révéler indispensables pour d'autres applications.

- Granularité fine : les ontologies de granularité fine, correspondent à des ontologies très détaillées, possédant ainsi un vocabulaire plus riche, capable d'assurer une description détaillée des concepts pertinents d'un domaine ou d'une tâche.
- Granularité large : les ontologies de granularité large, correspondent à des ontologies avec un vocabulaire moins détaillé. Les ontologies de haut niveau ont une granularité large, du fait que les notions, sur lesquelles elles portent, peuvent être raffinées par des notions plus spécifiques [Fürst, 2004].

Pour nos travaux, nous avons besoin d'utiliser une ontologie de domaine, semi-formelle, et d'une granularité fine.

La section suivante, aborde l'étude des différents langages existants, nous permettant, ainsi, de déterminer le langage d'expression de l'ontologie de domaine, utilisée dans nos travaux.

2.2.4. Langages

Il existe différents langages dans lesquels les ontologies peuvent être exprimées (cf. Figure 2-3). Nous décrivons, ci-dessous, les principaux langages, à savoir SHOE, XML, RDF, RDFS, OIL, DAML, DAML+OIL et OWL.

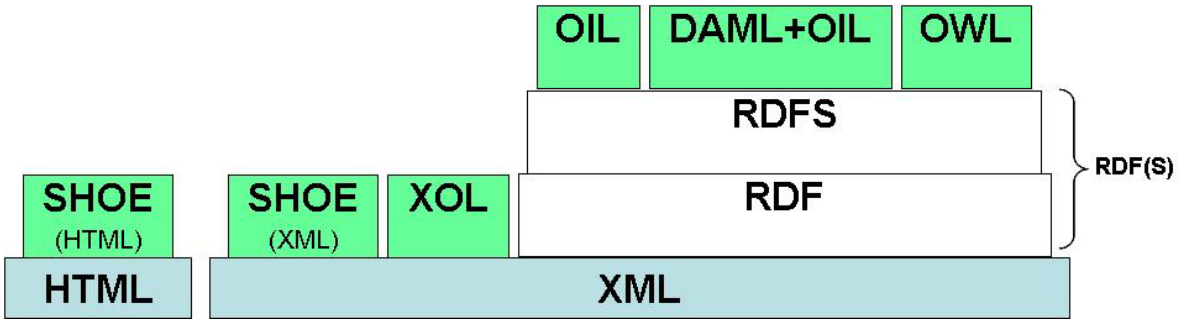


Figure 2-3 Langages des ontologies [Corcho et al., 2000]

2.2.4.1. SHOE

SHOE[4] (Simple HTML Ontology Extension), basé sur le web, combine les "frames" et les règles. Il a été construit comme une extension de HTML en 1996. Il utilise des étiquettes différentes de celles des spécifications HTML, de ce fait permettant l'insertion des ontologies dans des documents HTML. Plus tard, sa syntaxe a été adaptée à XML.

2.2.4.2. XML

XML (eXtensible Markup Language)[5] est un méta-langage, dérivé de SGML (Standart General Markup Language), utilisé pour définir des langages de marquage comme XHTML. Ces langages de marquage permettent la structuration des documents du web, non plus sur la base d'une structure figée, comme le permettait HTML, mais en laissant la possibilité au concepteur de distinguer les données selon leur sens et leur contenu. Un document XML se présente sous la forme de données taggées par un ensemble de balises, chacune pouvant comporter des attributs et des valeurs. Il n'y a pas de définition figée des balises. Ainsi, leur grammaire peut être consignée dans une DTD séparée (Document Type Definition). Cette DTD n'a aucune vocation à traiter la sémantique et ne sert qu'à définir des conventions de syntaxe.

Les schémas XML, normalisés par le W3C[6] (World Wide Web Consortium) en 2001, permettent de la même façon que les DTD, de modéliser et de valider les documents et leurs données. Ces schémas XML sont, cependant, beaucoup plus précis que les DTD car ils permettent de donner des contraintes plus strictes aux valeurs. Une des contradictions fondamentales entre XML et les ontologies réside dans le fait que XML ne traduit finalement que des grammaires tandis que les ontologies s'attachent à représenter la sémantique des objets d'un domaine, en déterminant les concepts et les relations qui le peuplent.

[4] http://www.cs.umd.edu/projects/plus/SHOE/
[5] http://www.w3.org/XML/
[6] www.w3.org

Un langage a pourtant été défini sur la base de XML, avec pour principal objectif l'échange d'ontologies. XOL (XML-based Ontology exchange Language)[7] tente de répondre à deux impératifs : d'une part, la nécessité d'avoir un langage de représentation des connaissances orienté objet, et d'autre part, la nécessité d'avoir une syntaxe XML permettant l'interopérabilité et intégrant les capacités des parsers XML. Il faut noter que l'objet même de XOL n'est pas la représentation d'ontologies mais bien leur échange. XOL peut toutefois être utilisé comme langage intermédiaire lors du transfert d'ontologies dans différents systèmes de bases de données [Corcho et al., 2000].

2.2.4.3. RDF

RDF (Ressource Description Framework)[8] est un modèle de représentation sémantique des informations du web qui utilise la syntaxe de XML. Ces représentations comportent des méta-données sur les ressources du web, comme les auteurs de pages web, leur date de création, etc. Les ressources du web sont l'élément de base de RDF. Chaque ressource est pourvue d'un identifiant uniforme de ressource (Uniform Resource Identifier, URI). Initialement recommandé par le W3C dans le but de standardiser les définitions et les usages des méta-données, RDF est également utile à la représentation de données. Les éléments principaux de RDF sont les objets, leurs attributs et les valeurs de ces attributs. L'intérêt principal de RDF est de définir un mécanisme permettant de décrire des données indépendamment de tout domaine et de toute spécificité. De même qu'avec XML, RDF ne permet pas la déclaration de propriétés particulières, leur définition est totalement libre.

Les schémas RDF (RDFS)[9] permettent de définir le vocabulaire utilisé dans les descriptions RDF. Ils confèrent un formalisme de représentation riche, incluant des classes, sous-classes, propriétés, sous-propriétés, des règles d'héritage de propriétés, etc., mais ne normalisent pas les inférences que l'on pourrait faire avec. La structure objet-classe des RDFS permet de représenter un modèle du domaine en définissant des objets du domaine et leurs relations pour rendre compte d'une ontologie.

2.2.4.4. OIL

OIL (Ontology Inference Layer)[10] est un langage de représentation d'ontologies qui dérive de RDF. Les principaux fondements du langage OIL sont les langages de frames (tels que XOL ou RDF) et les logiques de descriptions. OIL a été défini dans l'objectif de permettre la spécification et l'échange d'ontologies. Le premier objectif prévalant à la création du langage OIL, projet financé par la Communauté Européenne, était d'en faire un langage largement intuitif pour l'homme.

OIL permet la description d'ontologies sur la base des éléments de la logique de frames : les classes, les propriétés de classes, les relations entre les classes, etc. Les classes s'organisent en hiérarchie incluant des classes et des sous-classes. Elles peuvent se combiner par le biais d'expressions logiques : intersection, union, etc. Les propriétés se déclarent de la même façon que les axiomes logiques. Elles peuvent être fonctionnelles, c'est-à-dire n'avoir qu'une seule et même valeur, transitives ou symétriques. Des restrictions peuvent être apportées sur le type des propriétés ainsi que sur le nombre de valeurs qu'elles peuvent, le cas échéant, prendre. Un des aspects fondamentaux de OIL est sa sémantique formelle. Ainsi, chaque classe est organisée en ensemble d'objets et chaque propriété en

[7] ftp://smi.stanford.edu/pub/bio-ontology/xol.doc
[8] http://www.w3.org/RDF/
[9] http://www.w3.org/TR/rdf-schema/
[10] http://www.ontoknowledge.org/oil/

ensemble de paires d'objets, cette organisation devant, bien évidemment, se conformer aux contraintes spécifiées lors de la définition des classes et des propriétés.

Une ontologie exprimée en OIL est elle-même annotée avec des données propres, telles que le titre de l'ontologie, son ou ses créateurs, sa date de création, sa date de réactualisation, etc.

OIL suit les spécifications du W3C Dublin Core Standard pour exprimer ces méta-données sur l'ontologie elle-même.

2.2.4.5. DAML et DAML+OIL

DAML (DARPA Agent Markup Language)[11] est également un langage permettant la représentation d'ontologies. Il a été développé par DARPA (Defense Advanced Research Projects Agency) aux États-Unis dans le but de développer des langages et des outils permettant de rendre les contenus de documents accessibles et exploitables par des machines. DAML est une combinaison de XML et de RDF permettant de spécifier des objets mais également les relations entre ces objets. La dernière version de DAML se combine avec OIL (DAML+OIL)[12]. Ce nouveau langage supporte désormais la définition d'un certain nombre d'axiomes comme l'équivalence de classes ou de propriétés.

2.2.4.6. OWL

OWL (OntologyWeb Language)[13], créé en 2001 par le W3C [Smith et al., 2004] [Dean et al., 2004], hérite du langage DAML+OIL et doit permettre de représenter des ontologies sur le web. OWL est destiné à être utilisé lorsque les informations contenues dans les documents doivent être traitées par des applications logicielles, c'est-à-dire lorsqu'elles ne sont pas simplement « montrées » à l'utilisateur.

Une ontologie OWL est composée d'un en-tête (méta-données), d'axiomes et de faits. Les axiomes concernent la définition complète ou partielle de concepts et de relations (ou propriétés), la spécification de propriétés sur les relations (propriétés algébriques) et la définition d'axiomes sur les classes et les relations (équivalence, expression booléenne). Parmi les relations, on distingue celles dont le domaine de valeur sera de type primitif (attribut) de celles dont le domaine de valeur sera un autre concept (relation). Les faits concernent des individus, pour lesquels on donne des valeurs aux propriétés des classes dont ils sont les instances.

L'éventail des choix possibles concernant les langages de représentation et de spécification d'ontologies est très large. Cependant, le langage OWL tend à s'imposer, principalement, sur la base de critères techniques, notamment, en fonction des degrés d'expressivité et de formalisme voulus. Pour toutes ces raisons, notre choix s'est porté sur OWL.

2.2.5. Éditeurs d'ontologies

Nous allons passer en revue, dans cette section, les différents outils permettant de gérer (construire, modéliser, ouvrir, etc.) des ontologies, afin de sélectionner l'outil le plus adapté à nos travaux.

[11] http://www.daml.org/
[12] http://www.w3.org/TR/daml+oil-reference
[13] http://www.w3.org/TR/owl-ref/

2.2.5.1. PROTÉGÉ

PROTÉGÉ [14] a été développé par le Stanford Medical Informatics de l'université de médecine de Stanford depuis 1995. Il est construit autour d'un modèle de connaissances, inspiré par le paradigme des frames : classes, attributs et contraintes sur les attributs sont les primitives de modélisation proposées. Ce modèle autorise une liberté de conception importante, puisque le contenu des formulaires de spécification des classes peut être modifié suivant les besoins, via un système de méta-classes, qui constituent des sortes de « patrons » pour les classes du modèle du domaine. Il est adapté à la construction d'ontologies depuis la version PROTÉGÉ 2000. L'interface très complète, ainsi que l'architecture logicielle permettant l'insertion de pluggins (notamment des pluggins pour gérer les représentations sous forme graphique, par exemple OWLViz[15]), ont grandement contribué au succès de PROTÉGÉ. En quelques années, cet éditeur s'est imposé comme la référence, avec une communauté d'utilisateurs extrêmement importante et active. Ses nombreuses extensions lui permettent en particulier de gérer des langages standards comme RDF et surtout OWL [Knublauch et al., 2004], de créer des axiomes formels de manière intuitive et d'accéder aux ontologies par des interfaces graphiques évoluées. Cet outil permet d'importer et d'exporter des ontologies XML, RDF, RDFS, DAML+OIL et OWL.

2.2.5.2. OILEd

L'éditeur OILEd [16] a été développé par l'université de Manchester pour éditer des ontologies dans les langages de représentation OIL, puis DAML+OIL , les précurseurs de OWL [Bechhofer et al., 2001]. Il est donc explicitement orienté vers la représentation en logique de description expressive et, à ce titre, fournit tous les éléments d'interface permettant de spécifier des hiérarchies de concepts et de rôles, ainsi que la construction des expressions complexes définissant ces entités.

La simplicité, la robustesse de cet outil et la présence d'un raisonneur de logique de description, capable de tester la satisfiabilité des ontologies construites ou d'expliciter de nouvelles relations de subsomption entre concepts complexes, en font un outil de référence relativement populaire. Comme le soulignent les concepteurs, il s'agit plutôt d'un « bloc-notes » offrant assez de fonctionnalités pour permettre à des utilisateurs de construire des ontologies et de démontrer comment il est possible d'employer le raisonneur pour examiner les ontologies et en assurer l'uniformité.

OILEd permet d'importer et d'exporter les ontologies construites dans des langages standards tels que DAML+OIL, RDF ou OWL et d'exporter les ontologies RDFS.

2.2.5.3. ONTOEDIT

ONTOEDIT [Sure et al., 2002] est un outil mis au point par l'institut AIFB de l'université de Karlsruhe et qui est maintenant commercialisé par la société Ontoprise GmbH. Il s'inspire de l'approche par frames mais gère de nombreux formats libres de la communauté web sémantique (FLogic, DAML+OIL, RDFS). Il s'est, en effet, le premier intéressé à la modélisation « intuitive » des axiomes, indépendamment d'un formalisme ou d'un autre, pour faciliter la traduction d'un langage de représentation à un autre.

[14] http://protege.stanford.edu/
[15] http://www.co-ode.org/downloads/owlviz/
[16] http://oiled.man.ac.uk/

ONTOEDIT gère la synonymie en admettant plusieurs noms pour un même concept. Il permet d'importer et d'exporter les ontologies construites dans différents langages : RDF(S), DAML+OIL et FLogic.

2.2.5.4. WebODE

WebODE[17] est une plateforme en ligne développée par le groupe Ontological Engineering du département d'Intelligence artificielle de la faculté d'Informatique de l'université polytechnique de Madrid. WebODE est composée de plusieurs modules : un éditeur d'ontologie qui intègre la plupart des services nécessaires à la construction d'ontologies (édition, navigation, comparaison, fusion, raisonnement, etc.), un système de gestion des connaissances à base ontologique, un générateur automatique de portail du web sémantique, un outil pour annoter les ressources du web et un éditeur de services pour le web sémantique. La plateforme WebODE met l'accent sur la possibilité d'un travail collaboratif, et sur la possibilité, comme dans PROTÉGÉ, d'étendre la plateforme à l'aide de modules complémentaires, comme un moteur d'inférences. WebODE, similaire en cela aux autres éditeurs, accepte l'import et l'export d'ontologies en RDFS, DAML+OIL et OWL.

2.2.5.5. DOE

DOE (Differential Ontologies Editor) a été développé à l'Institut National de l'Audiovisuel en France par [Troncy et al., 2002]. L'éditeur DOE offre des interfaces de création, modification et suppression de concepts et de relations, une représentation graphique de l'arbre ontologique, et des fonctionnalités de recherche et de navigation dans la structure créée. Il propose une interface ergonomique pour associer, à chaque terme, sa définition encyclopédique, ses éventuels synonymes et ses principes différentiels et cela en plusieurs langues.

Le passage à une ontologie computationnelle s'effectue par un export de l'ontologie formelle dans un certain nombre de langages opérationnels tels que RDFS, DAML+OIL et OWL. Cette traduction s'effectue grâce à des feuilles de style XSLT, appliquées au format de sauvegarde XML de l'éditeur. De la même façon, DOE peut importer des ontologies modélisées dans d'autres outils grâce à des feuilles XSLT dédiées. Pour être tout à fait complet, il faut noter que l'éditeur permet aussi de sauvegarder un certain nombre de méta-données concernant l'ontologie elle-même.

L'interopérabilité de DOE avec d'autres outils, tel que PROTÉGÉ par exemple, est nécessaire dans la mesure où il ne met pas en œuvre les fonctionnalités d'expressivité formelle, prises en charge par les autres environnements [Troncy et al., 2003]. Cet outil ne constitue pas, pour l'instant, un environnement de développement d'ontologies complet mais se situe en amont d'autres éditeurs implémentant, par exemple, les logiques de description.

Le Tableau 2-1 ci-dessous, résume pour chaque éditeur, les langages utilisés pour exprimer les ontologies (colonne 2). Les ontologies importées et exportées par l'éditeur sont présentées dans la colonne 3. La dernière colonne précise si l'éditeur offre une visualisation graphique des ontologies.

[17] http://webode.dia.fi.upm.es/WebODEWeb/index.html

Editeur	Langage	Import/export d'ontologie	Visualisation
PROTÉGÉ	OWL	RDF; RDFS; DAML+OIL; XML; OWL	Avec plugin OWLViz
Oiled	DAML+OIL	Import/export DAML+OIL; OWL RDF Export : RDFS	Visualisation de la subsomption de classes
OntoEdit	RDFS; F-Logic; DAML+OIL	RDFS; F-Logic; DAML+OIL	Non
WebODE	Langage interne	RDFS; DAML+OIL; OWL	Oui
DOE	XML	RDFS; OWL DAML+OIL	Visualisation de l'arbre ontologique

Tableau 2-1 Comparaison des éditeurs d'ontologies

Dans nos travaux, nous avons opté pour l'utilisation de PROTÉGÉ, car, il est un des seuls à supporter OWL. De plus, il fournit une interface intuitive pour la création des ontologies, ainsi que, la possibilité de les visualiser à l'aide de plugin.

Conclusion

Les ontologies peuvent être classifiées en fonction du niveau de formalisme du modèle, des objets qu'elles modélisent, et du degré de granularité des connaissances représentées. De plus, différents langages et éditeurs sont disponibles pour la création d'ontologies. Pour notre approche, notre choix s'est porté sur une ontologie de domaine, semi-formelle, exprimée en OWL dans l'éditeur PROTÉGÉ.

L'objectif de ce chapitre a été de présenter deux parties qui correspondent à une vision globale des éléments nécessaires pour notre thématique. Ceci se concrétise, d'une part, par les principes généraux de l'IDM, c'est-à-dire les modèles, la méta-modélisation et la transformation de modèles, et, d'autre part, par les ontologies, à savoir leurs constituants, leurs classifications, leurs langages d'expression ainsi que leurs éditeurs.

Dans le troisième chapitre, nous présentons un état de l'art, incluant des définitions de la composition de modèles, les hétérogénéités pouvant exister entre deux modèles à composer, ainsi qu'une classification des approches existantes de composition de modèles.

Chapitre **3**

Composition de modèles

3. Composition de modèles

Introduction

La composition de modèles a été traitée par de nombreuses équipes de recherche, pour plusieurs modèles et dans différents domaines d'application. Certaines approches se sont intéressées à la composition de schémas de base de données [Algergawy et al., 2009] [Fagin et al., 2011] [Bellahsene et al., 2011], ou à la composition ou alignement d'ontologies [Ehrig et al., 2005] [Euzenat et al., 2010]. Une vue d'ensemble sur la composition de schémas de base de données et d'ontologies est présentée dans [Bernstein et al., 2011]. D'autres approches traitent la composition de modèles exprimés dans différents langages (modèle UML, schéma de base de données, etc.) [Haddar et al., 2002], [Pottinger et al., 2003], [Boronat et al., 2007]. Enfin, des approches se focalisent sur la composition de modèles UML à savoir les diagrammes de classes [Anwar et al., 2007a], [Boronat et al., 2007], [Fleurey et al., 2007a] [Oliveira et al., 2009], [Rungworawut, 2010], [UML, 2011], et les modèles comportementaux, tels que les diagrammes d'état [Brunet et al., 2006] et les diagrammes de séquences [Jakimi et al., 2007].

Ce chapitre a pour objectif de présenter les approches existantes. Nous nous intéressons, dans cet état de l'art, à la composition de modèles UML et plus précisément aux diagrammes de classes, ainsi qu'à la composition de modèles indépendants d'un langage de modélisation.

Nous commençons, d'abord, par donner un ensemble de définitions relatives à la composition de modèles. Ensuite, un exemple est présenté afin d'illustrer les hétérogénéités pouvant exister entre les modèles. Enfin, à partir de l'état de l'art, une synthèse et une analyse des limites des approches existantes sont exposées afin de définir le périmètre et les objectifs de nos travaux.

3.1. Composition de modèles

3.1.1. Définitions

Dans la littérature, la composition apparaît sous divers termes. Dans [Didonet et al., 2006], la composition de modèles est définie comme une opération qui combine deux ou plusieurs modèles en un seul. Selon [Herrmann et al., 2007], la composition de modèles, dans sa forme la plus simple, se réfère au mécanisme de la combinaison de deux modèles en un nouveau. Une autre définition proposée par [Bézivin et al., 2006] est la suivante : l'opération de composition prend en entrée deux modèles MA et MB et un modèle de correspondances CAB entre eux, et combine leurs éléments en un nouveau modèle en sortie. Le modèle de correspondances représente les liens entre les éléments des différents modèles ; il est produit par une opération Match qui prend un ensemble de modèles en entrée et cherche les équivalences entre leurs éléments.

Certains travaux [Haddar et al. , 2002], [Jing et al., 2005], [Boronat et al., 2007], [Rungworawut, 2010] utilisent l'appellation « intégration de modèles » pour désigner la composition de modèles. Selon [Haddar et al., 2002], l'intégration de modèles consiste à prendre deux à deux les modèles et à créer un nouveau modèle bien formé et cohérent contenant toutes les informations des modèles d'origine.

Selon le dictionnaire Larousse[18], l'intégration désigne l'action d'intégrer. De son étymologie latine « integrare », intégrer signifie « rendre entier », faire entrer dans un ensemble, dans un groupe plus vaste. Selon [Weston, 1993], l'intégration consiste à combiner des composants de telle façon à former un nouvel ensemble constituant un tout pour créer de la synergie.

Dans le contexte de l'entreprise, l'intégration est perçue autrement. Le problème d'intégration est communément appelé intégration d'entreprise (EI - Enterprise Integration). Dans [Vernadat, 1996], l'auteur définit, ainsi, l'intégration d'entreprise comme étant le processus qui « consiste à faire interopérer fortement les personnes, les machines et les applications afin d'accroître la synergie au sein de l'entreprise ». Dans le même ordre d'idées, les auteurs de [Williams et al., 2001] définissent l'intégration d'entreprise comme « la coordination de tous les éléments incluant les processus métier, les ressources humaines, et la technologie d'une entreprise fonctionnant ensemble dans le but d'atteindre la réalisation optimale de la mission de cette entreprise telle que définie dans la stratégie de l'entreprise ».

Un autre terme proche de l'intégration, qu'il convient ici de préciser, est celui de l'interopérabilité. Dans [Vernadat, 1996], l'auteur l'a définie comme étant la capacité à communiquer avec des systèmes distants pour accéder et faire appel à leurs fonctionnalités. L'interopérabilité est également définie par [Wegner, 1996] comme l'aptitude de deux systèmes (ou plus), à communiquer, coopérer et échanger des données et services, et ce, malgré les différences dans les langages, les implémentations et les environnements d'exécution ou les modèles d'abstraction.

Dans d'autres contextes d'application, la composition apparaît sous d'autres termes. En effet, dans le développement de logiciels par aspects [Kiczales et al., 1997] [France et al., 2004], on parle souvent d'opération de tissage. Dans le domaine des bases de données, la composition se manifeste par l'opération d'intégration d'un ensemble de vues d'une base de données, ou de plusieurs schémas de bases de données hétérogènes et réparties [Batini et al., 1986] [Navathe et al., 1986]. Et en ingénierie des connaissances, on parle souvent d'alignement d'ontologies [Ehrig et al., 2005][Euzenat, 2004].

Afin d'éviter toute confusion avec la définition de l'intégration proposée par Vernadat [Vernadat, 1996], nous utilisons le terme « composition » de modèles. Dans nos travaux, et conformément aux travaux cités plus haut, nous proposons la définition suivante :

La composition de modèles consiste à combiner deux modèles sources pour créer un modèle cible, cohérent, contenant toutes les informations appartenant aux deux modèles initiaux.

Les opérations, sur lesquelles se base la composition de modèles, sont les suivantes :

- La première est l'opération de comparaison, appelée aussi matching. Elle prend en entrée les modèles sources, les compare afin de trouver les équivalences entre leurs éléments. Elle produit en sortie (a) le modèle de correspondances appelé aussi mapping, qui contient les liens d'équivalence entre les éléments des deux modèles et (b) les éléments non équivalents que nous appelons éléments isolés.
- La deuxième opération est celle de la fusion. Elle prend en entrée les modèles sources et le modèle de correspondances et combine les éléments équivalents.

[18] http://www.larousse.fr/dictionnaires/francais-monolingue

- La troisième opération est celle de la translation. Elle prend en entrée les modèles sources, et translate (rajoute) les éléments isolés.

Les deux dernières opérations produisent en sortie le modèle cible, appelé aussi modèle résultat.

3.1.2. Exemple de modèles à composer

Dans le but d'illustrer nos travaux, nous présentons un exemple issu du domaine bancaire. Les deux diagrammes de classes M1 (Figure 3-1) et M2 (Figure 3-2) ci-dessous représentent des extraits de modèles de deux banques devant être fusionnées.

- **Cas de la Banque 1**

En ce qui concerne le modèle M1 de la banque 1, un client est considéré comme une personne physique ou morale (entreprise) et il est caractérisé par un numéro de client, son adresse, son numéro de téléphone et par une adresse mail. Une personne morale est définie par sa raison sociale et une personne physique est décrite par son nom, son prénom et sa CIN (Carte d'Identité Nationale).

Un client peut posséder un ou plusieurs comptes au sein d'une banque. Un compte bancaire est caractérisé par un numéro et un solde. Il peut être créé, consulté, crédité et débité. Un compte peut être de deux types : courant ou d'épargne, et peut disposer d'une ou de plusieurs cartes de crédit, identifiées par leur code, leur type (Visa, MasterCard, CMI, etc.), leur date d'expiration, et le montant de retrait maximal. Un compte courant peut autoriser un découvert. Un compte d'épargne définit son taux d'intérêt.

Un membre du personnel possède un code, un mail, un poste, une situation familiale (célibataire ou non), une CIN, et un numéro de téléphone. Ces informations peuvent être ajoutées, modifiées, supprimées ou consultées.

Un membre du personnel peut gérer un ou plusieurs clients, se charge de les ajouter, les modifier, les supprimer ou les consulter. Plusieurs opérations peuvent être faites sur un compte bancaire. Chacune d'elles est définie par sa référence, la date de l'opération, sa nature (virement, payement par carte, etc.), le type (versement, retrait) et le montant de l'opération. Une opération peut concerner deux comptes, dans le cas par exemple d'un virement.

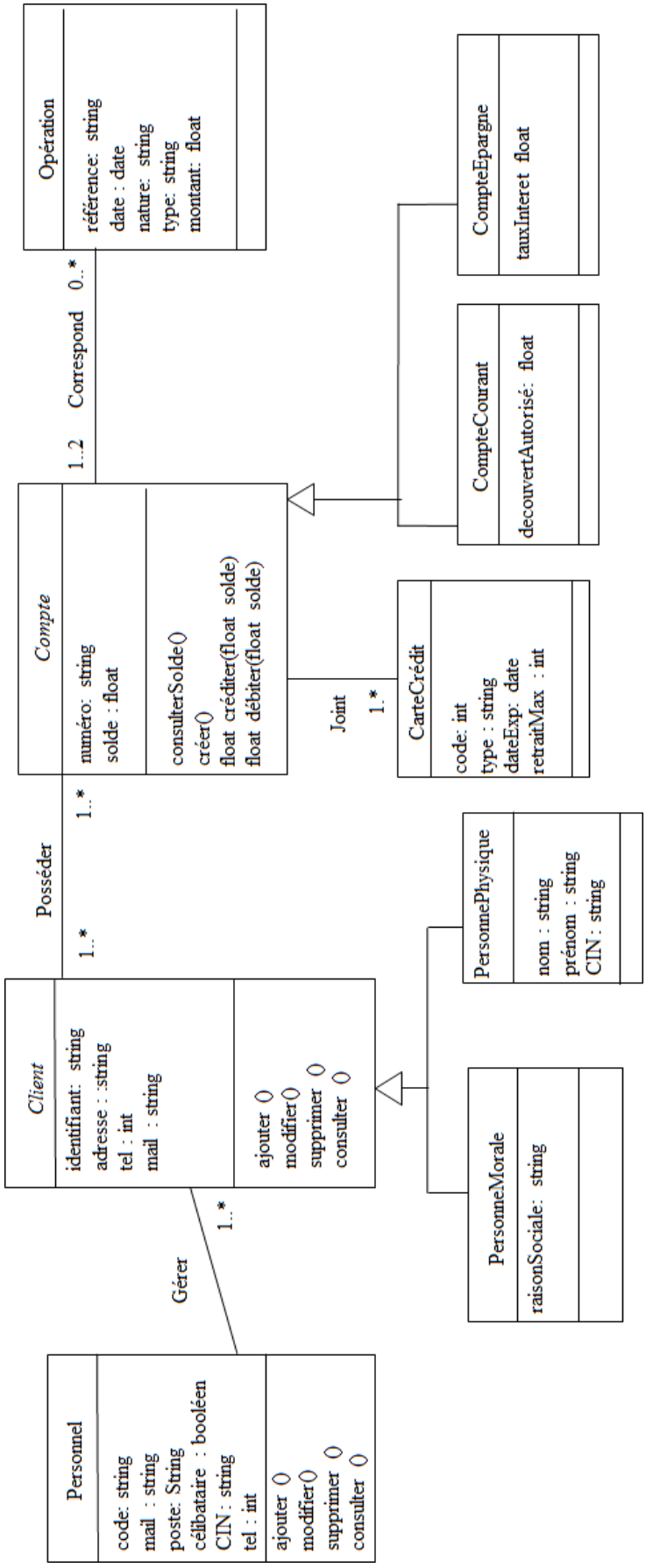


Figure 3-1 Modèle M1 de la Banque n°1

- **Cas de la Banque 2**

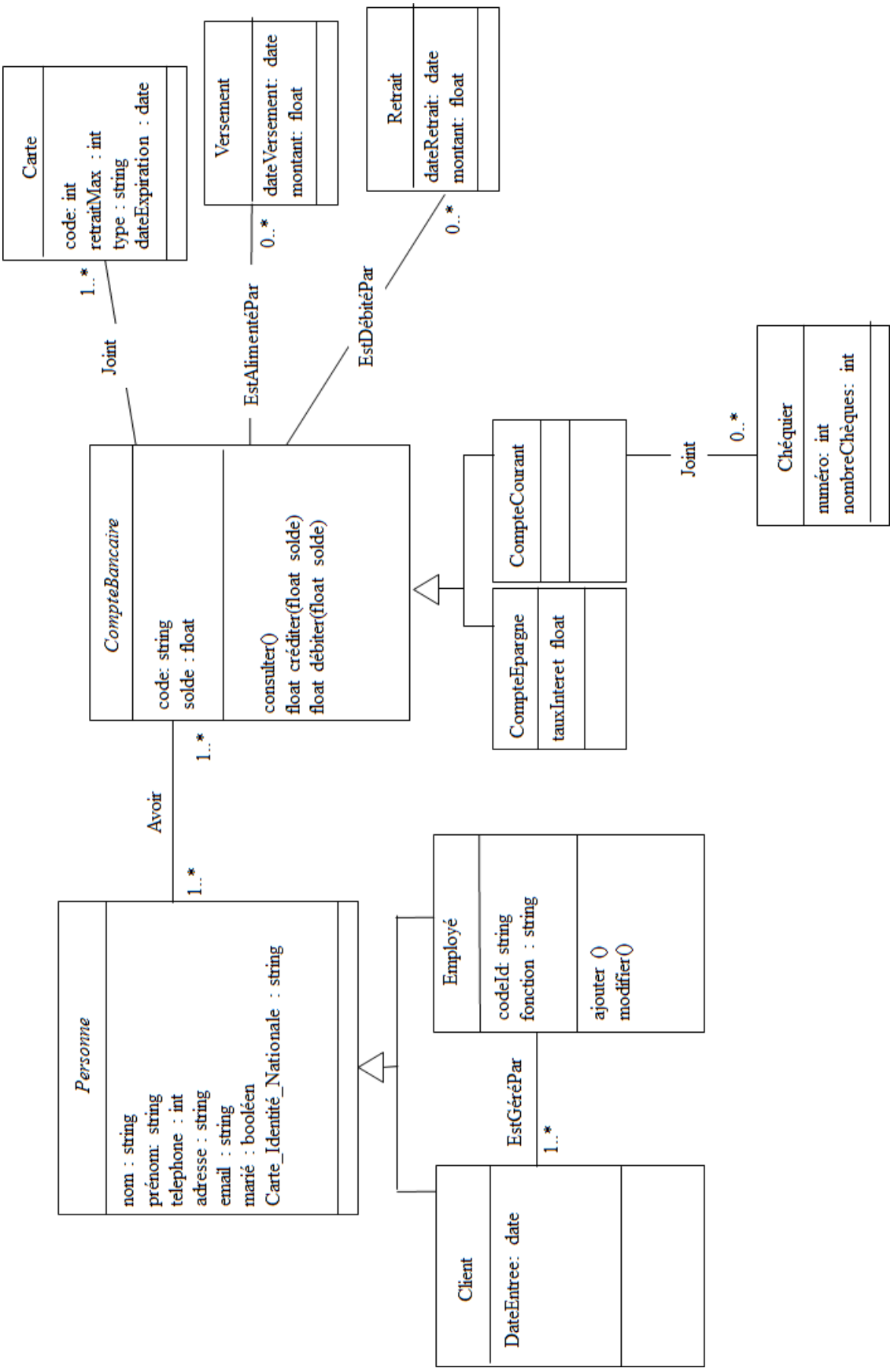


Figure 3-2 Modèle M2 de la Banque n°2

Dans la banque 2, les clients et les employés sont généralisés par une classe Personne. Ils sont caractérisés par un nom, un prénom, un numéro de téléphone, un email, une adresse, une situation familiale (marié ou non) et un numéro de Carte d'Identité Nationale.

Un client possède, aussi, une date d'entrée en banque, et un employé possède, quant à lui, un code et sa fonction. Ces informations peuvent être ajoutées ou modifiées. Les employés gèrent les comptes des clients. Un compte est défini par un code et un solde. Il peut être consulté, crédité et débité.

Chaque personne a un compte courant et/ou un compte d'épargne dont le taux d'intérêt est défini. Un compte bancaire peut disposer d'une ou de plusieurs cartes caractérisées par un code, un retrait maximal, un type et une date d'expiration.

Seul un compte courant peut disposer de chéquiers. Ce dernier est caractérisé par un numéro et un nombre de chèques. Un versement, portant une date de versement et un montant, alimente le compte bancaire ; et un retrait, caractérisé aussi par un montant et une date de retrait, débite le compte bancaire.

3.1.3. Hétérogénéités des modèles

Généralement, les modèles à composer font partie d'un même domaine métier, mais sont construits indépendamment les uns des autres. Ainsi, nous pouvons alors rencontrer différents types d'hétérogénéités entre les éléments de modèles :

- Deux éléments qui ont le même sens, mais représentés par des noms différents
- Deux éléments qui ont le même nom, mais n'ont pas le même sens
- Deux éléments qui ont le même nom, mais n'ont pas la même nature (exemple : un élément est représenté sous forme d'une classe dans un modèle et sous forme d'un attribut dans un autre)
- Deux classes qui ont le même nom mais pas les mêmes attributs et/ou opérations
- Deux attributs qui ont le même nom mais pas le même type
- Deux opérations qui ont le même nom mais pas les mêmes paramètres
- Deux paramètres qui ont le même nom mais pas le même type
- Deux associations qui ont le même nom mais pas les mêmes multiplicités/extrémités d'association/navigabilité/agrégation
- Deux classes qui ont le même nom mais pas le même voisinage ou proximité (les associations, les relations de généralisations et les classes qui les entourent)
- Deux attributs/opérations qui ont le même nom mais n'appartiennent pas à des classes qui ont le même nom (ou noms équivalents)
- Deux paramètres qui ont le même nom mais n'appartiennent pas à des opérations qui ont le même nom (ou noms équivalents)
- Deux associations qui ont le même nom mais ne relient pas des classes qui ont le même nom (ou noms équivalents) deux à deux.

Nous avons classifié les hétérogénéités possibles dans une taxonomie, présentée dans la Figure 3-3. Avant de la décrire, nous définissons la notion d'élément :

Un élément peut être une classe, un attribut, une association, une opération, un paramètre ou une relation de généralisation. Il a une syntaxe, une sémantique, une nature et une structure locale et globale.

- *Syntaxe d'un élément* : la chaîne de caractères du nom de l'élément.
- *Sémantique d'un élément* : le sens lié à l'élément.
- *Nature d'un élément* : détermine s'il s'agit d'une classe, un attribut, une association, une opération, un paramètre ou une relation de généralisation.
- *Structure locale d'un élément* : il s'agit des composants de l'élément
  - *Structure locale d'une classe* : ses attributs et ses opérations
  - *Structure locale d'une association* : ses multiplicités maximale et minimale, sa navigabilité, sa composition (agrégation) et ses extrémités d'association.
  - *Structure locale d'une opération* : ses paramètres et son type de retour
  - *Structure locale d'un attribut* : son type, ses multiplicités maximale et minimale
  - *Structure locale d'un paramètre* : son type.

- *Structure globale d'un élément :*
  - *Structure globale d'une classe* : les associations, les relations de généralisations et les classes qui sont liées à cette classe
  - *Structure globale d'une association* : les classes qu'elle relie
  - *Structure globale d'une opération* : la classe à laquelle elle appartient
  - *Structure globale d'un attribut* : la classe à laquelle il appartient
  - *Structure globale d'un paramètre* : l'opération à laquelle il appartient
  - *Structure globale d'une relation de généralisation* : les classes qu'elle relie.

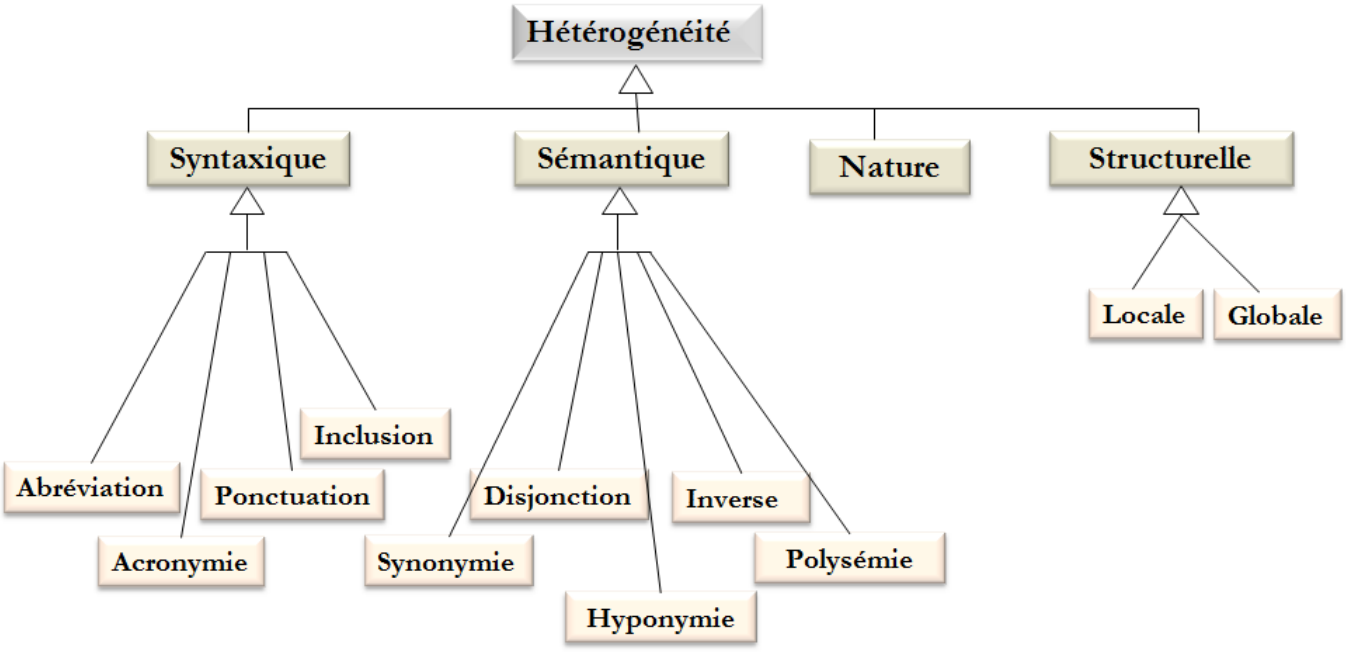


Figure 3-3 Taxonomie des hétérogénéités

Nous classifions les hétérogénéités des éléments de modèles en quatre catégories (cf. Figure 3-3) : syntaxique, sémantique, de nature et structurelle.

L'hétérogénéité syntaxique : correspond aux différences présentes dans la syntaxe des éléments de modèles. Nous pouvons trouver :

- *Inclusion*: une chaîne de caractères est incluse dans l'autre. Nous définissons deux types d'inclusion :
    - *Inclusion totale :* deux éléments sont en relation d'inclusion totale si l'un est inclus dans l'autre. Exemple : les deux attributs « mail » de la classe « Client » du modèle M1 et « email » de la classe « Client » du modèle M2 sont en relation d'inclusion totale.
    - *Inclusion partielle :* deux éléments sont en inclusion partielle, si une partie de la chaîne de caractères d'un élément, est incluse dans la chaîne de caractères de l'autre. Par exemple : « **Cod**Banq » et « **Cod**eB ».
- *Abréviation* : un élément est représenté par un mot dans un modèle et son abréviation dans l'autre. Exemple : l'attribut « tel » de la classe « Personnel » de M1 et l'attribut « telephone » de la classe « Employé » de M2 (hérité de « Personne »).
- *Acronyme* : un élément est représenté par un mot dans un modèle et son acronyme dans l'autre. Exemple : l'attribut « CIN » de la classe « Personnel » de M1 et l'attribut « Carte_Identité_Nationale » de la classe « Employé » de M2 (hérité de « Personne »).
- *Ponctuation* : un élément est représenté par un mot dans un modèle et avec le même mot contenant des ponctuations (« . », « - », « _ », etc.) dans l'autre.

L'hétérogénéité sémantique : correspond aux différences liées à l'interprétation et au sens associé aux éléments des modèles. Ces hétérogénéités sémantiques se produisent principalement lorsque :

- Plusieurs noms symboliques recouvrent le même concept. On parle dans ce cas de *synonymie*. Exemple : les attributs « poste » de la classe « Personnel » de M1 et « fonction » de la classe « Employé » de « M2 ».
- Une information dans un modèle est représentée par son contraire dans l'autre modèle. On parle de *Disjonction*. Exemple : les deux attributs de type Booléen « célibataire » de la classe « Personnel » de M1 et « marié » de la classe « Employé » de M2 (hérité de « Personne »).
- Une information dans un modèle est représentée par son *inverse* dans l'autre. Par exemple, l'association « Gérer » de M1 est l'inverse de l'association « EstGéréPar » de M2.
- Un même nom recouvre des éléments différents. Nous parlons dans ce cas de *polysémie*. Par exemple : une classe « Famille » dans un modèle représente une famille de produits, et une classe « Famille » dans un autre modèle, représente une famille de personnes.
- Un élément est représenté par une classe dans un modèle et par sa sous-classe dans l'autre. Exemple : « Opération » de M1 et « Versement » de M2. Nous parlons dans ce cas d'*hyponymie*.

L'hétérogénéité de nature : il s'agit de l'attribution de différents types d'éléments, par différents concepteurs, à un même concept. Par exemple, l'élément « Versement » est défini comme étant une classe dans le modèle M2, et comme étant un attribut dans le modèle M1. Il s'agit de l'attribut « type » de la classe « Opération ».

L'hétérogénéité structurelle : correspond à une différence dans la structure locale et/ou globale des éléments.

- *L'hétérogénéité de structure locale* : différence dans la structure locale des éléments de modèles. Exemple : les classes « Client » de M1 et « Client » de M2 n'ont pas exactement les mêmes attributs et opérations.
- *L'hétérogénéité de structure globale* : différence dans la structure globale des éléments de modèles. Exemple : les classes « Personnel » de M1 et « Employé » de M2 ne sont pas liées exactement aux mêmes éléments (associations, relations de généralisation et classes).

Conformément aux hétérogénéités définies, nous présentons les types de comparaison suivants :

- *Comparaison de nature* : elle concerne la comparaison du type des éléments de modèles.
- *Comparaison syntaxique* : elle concerne la comparaison de la syntaxe des éléments de modèles. Elle peut donc détecter l'identité, l'inclusion, l'acronymie et l'abréviation entre les éléments.
- *Comparaison sémantique* : elle concerne la comparaison de la sémantique des éléments de modèles. Elle peut détecter la synonymie, la disjonction, l'inverse, la polysémie et l'hyponymie.
- *Comparaison structurelle locale* : elle concerne la comparaison de la structure locale des éléments de modèles. Elle peut détecter les éléments équivalents en structure locale.
- *Comparaison structurelle globale* : elle concerne la comparaison de la structure globale des éléments de modèles. Elle peut détecter les éléments équivalents en structure globale.

3.2. Etude des approches existantes de composition de modèles

Afin d'évaluer les approches existantes de composition de modèles et d'en dégager les limites, nous présentons, dans cette section, un ensemble de travaux en nous appuyant sur des critères de comparaison.

3.2.1. Critères de comparaison des approches

Les caractéristiques de ces travaux sont définies à l'aide de sept critères, que nous avons identifiés, comme les plus pertinents.

- **Type de modèles à composer** : précision du type des modèles à composer, à savoir des diagrammes de classes UML ou des modèles indépendants d'un langage de modélisation (indépendants du méta-modèle).
- **Type de comparaison :** spécification du type de comparaison qui peut être syntaxique, sémantique, structurelle locale et/ou structurelle globale.
- **Type d'approche :** nous avons identifié, dans la littérature, trois types d'approches :
  - *Approche syntaxique* : compare syntaxiquement les modèles.
  - *Approche semi-hybride* : combine deux ou trois types de comparaison (syntaxique, sémantique, structurel local, structurel global).
  - *Approche hybride :* combine les quatre types de comparaison (syntaxique, sémantique, structurel local et structurel global).
- **Fusion :** précision si les travaux assurent la fusion, ou se limitent à la comparaison de modèles.

- **Type de composition :** c'est la manière dont la composition peut être exprimée. Une composition de modèles peut être exprimée sous forme de règles (telle que des règles de comparaison, de fusion, etc.), de règles de transformation, d'algorithmes, de mesures (telle qu'une mesure de similarité), ou de tissage (liens entre éléments de modèles).
- **Langage de composition :** la composition a besoin d'un formalisme pour l'exprimer. Ceci varie d'une approche à une autre. Ce critère précise le langage utilisé par chaque approche. Nous pouvons trouver un langage informel (naturel), un langage formel (logique des prédicats, mathématique, etc.), l'algorithmique, un langage de programmation, etc.
- **Outil :** ce critère précise si les travaux proposent ou non une implémentation de leur approche.

Nous présentons, dans ce qui suit, les travaux de composition de modèles, classés selon le type d'approche. Certains travaux se sont focalisés uniquement sur l'étape de comparaison de modèles. Pour cela, nous préciserons si l'approche assure la composition de modèles dans son ensemble ou uniquement l'étape de comparaison.

3.2.2. Approches syntaxiques

Plusieurs approches ont proposé la composition de modèles en comparant uniquement la syntaxe des éléments. Nous les exposons une par une.

- **Approche de comparaison de modèles Ngram [Manning, 1999]**

Un algorithme, appelé Ngram, a été défini dans [Manning, 1999]. Il est dédié à la comparaison syntaxique de modèles indépendants du méta-modèle. Cet algorithme génère une mesure de similarité comprise entre 0 et 1 de deux chaînes de caractères en se basant sur le calcul du nombre de leurs sous-chaînes de caractères (de taille N) identiques.

Cette approche identifie l'identité syntaxique dans le cas où la mesure est égale à 1, et l'inclusion syntaxique, dans le cas où la mesure est comprise strictement entre 0 et 1. Si la mesure est égale à 0, alors les deux chaînes de caractères sont différentes.

- **Approche de composition de modèles Epsilon Merging Language [Dimitrios et al., 2006a] [Dimitrios et al., 2006b] et [Dimitrios et al., 2006c]**

Un langage dédié à la composition de modèles a été proposé par [Dimitrios et al., 2006a] [Dimitrios et al., 2006b] et [Dimitrios et al., 2006c]. Il s'agit de EML (Epsilon Merging Language) dont le caractère générique permet de composer des modèles de divers méta-modèles. EML fait partie de la plateforme Epsilon [Dimitrios et al., 2006c] supportée par l'environnement Eclipse. Il intègre un langage de comparaison de modèles ECL (Epsilon Comparaison Language) et un langage de transformation de modèles ETL (Epsilon Transformation Language).

Une spécification EML est un ensemble de règles décrivant comment composer les modèles. Il s'agit d'abord des règles de comparaison des éléments de modèles pour identifier les éléments correspondants et non correspondants, ensuite, des règles de fusion des éléments de modèles afin de fusionner les éléments correspondants, et enfin, des règles de transformation qui ne s'appliquent que sur les paires d'éléments qui ne sont pas correspondants.

Les règles EML partagent certaines fonctionnalités communes, exploitées pour favoriser la réutilisation d'informations et donc minimiser l'intervention du développeur lors de l'écriture de la composition. Le langage EML propose pour cela le mécanisme d'héritage entre les règles et la notion de règle abstraite.

Les auteurs proposent dans [Dimitrios et al., 2006a] une étude de cas qui comporte un ensemble de règles pour la composition de diagrammes de classes UML. Le code ci-dessous montre un exemple de règles dans EML illustrant la comparaison, la fusion et la transformation de deux classes (Left !Class et Right !Class). La règle de comparaison retourne vrai si et seulement si les deux classes sont toutes les deux concrètes, ou toutes les deux abstraites, et si leurs noms, et leurs espaces de noms sont les mêmes.

La règle de fusion crée, dans le modèle composite résultat, une classe dont le nom et l'espace de noms viennent de la classe de gauche, les propriétés sont l'union des propriétés de deux classes en entrée. La règle de transformation crée l'objet destination, dont le nom, l'espace de nom et les propriétés proviennent de l'objet source.

```
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;


  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();


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 3-4 Extrait de spécification EML : composition de diagrammes de classes UML [Dimitrios et al., 2006a]

Cette approche syntaxique fusionne uniquement les éléments qui ont des noms identiques. EML ne suppose pas composer les modèles en conflits, cela veut dire qu'il ne propose pas de solution pour l'identification ou la résolution des incohérences entre les modèles à fusionner.

- **Approche de composition de modèles de [Rungworawut, 2010]**

Les travaux de [Rungworawut, 2010] proposent la composition de patterns avec des modèles du logiciel (diagrammes de classes UML) existants. En effet, un pattern est une solution réutilisable pour résoudre des problèmes récurrents de conception de logiciels. Les patrons appropriés sont ensuite appliqués à un problème de conception logiciel récurrent, qu'il faut composer avec les diagrammes existants.

Ces travaux proposent un framework formel de composition, qui fournit des règles de composition sous forme d'une spécification formelle, basée sur les mathématiques (ensemble, quantificateurs, union, etc.). Lors de la comparaison, ils ne prennent en compte que l'identité des éléments.

- **Approche de composition de modèles de l'OMG [UML, 2011]**

L'OMG a défini dans la spécification d'UML 2.4 [UML, 2011] la fusion de packages. Un package peut contenir des classes, attributs, opérations, associations, etc. Le « package merge » (cf. Figure 3-5) définit comment les contenus des deux packages A et B seront fusionnés. Il prend en entrée les deux packages et génère le package résultat.

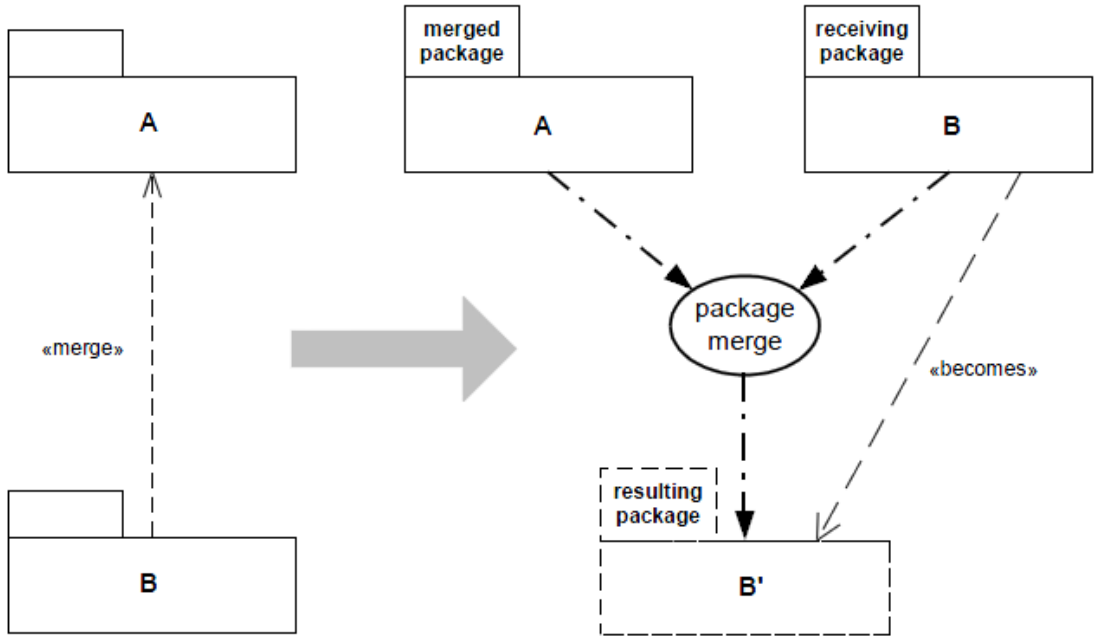


Figure 3-5 Vue conceptuelle du « package merge » [UML, 2011]

Le package source (package B) est appelé *package récepteur*. C'est celui dont le contenu est étendu après l'opération de fusion. Le package destination (package A) est appelé *package fusionné*. C'est le paquetage qui contient les éléments supplémentaires ajoutés au package récepteur.

PackageMerge est défini par un ensemble de règles de transformations et de contraintes. Les règles de transformation assurent la mise en correspondance des éléments et la fusion des éléments correspondants. Dans les cas où certains éléments dans deux packages représentent la même chose, leur contenu est fusionné en un seul élément résultant conformément aux règles du package merge. Les contraintes permettent d'assurer la validité de la fusion. Les règles sont définies en langage naturel.

La fusion n'est appliquée que si les deux packages à fusionner contiennent deux éléments avec le même nom. Nous parlons ici de comparaison syntaxique qui teste uniquement l'identité des éléments.

3.2.3. Approches semi-hybrides

D'autres travaux ont proposé des approches de composition de modèles semi-hybrides, combinant deux ou trois types de comparaison. Nous les présentons ci-dessous :

- **Approche de composition de modèles : Opérateurs Match et Merge de model management [Bernstein et al., 2000] [Madhavan et al, 2001] [Bernstein, 2003] [Pottinger et al., 2003]**

Les auteurs de [Bernstein et al., 2000] [Bernstein, 2003] proposent un framework appelé *model management* qui contient des opérateurs pour la gestion de modèles, écrits dans différents langages. Ces opérateurs incluent *Match* qui retourne le mapping entre les éléments de modèles, *Apply* qui applique un ensemble de fonctions sur tous les éléments du modèle, *Diff* qui prend en entrée deux modèles et un mapping et retourne un modèle contenant les éléments appartenant au premier modèle et n'appartenant pas au second modèle, et *Merge* qui combine les modèles après que les correspondances soient établies.

L'opérateur *Match* définit le mapping entre les éléments de modèles, en se basant sur le traitement des correspondances des schémas des bases de données, proposé par la même équipe dans [Madhavan et al, 2001]. Ils proposent un algorithme appelé Cupid, qui procède par déduction des correspondances à partir d'une mesure. Cette dernière est calculée en trois phases.

- La première phase est une comparaison linguistique (syntaxique et sémantique). Elle utilise un thesaurus pour aider à l'identification des synonymes. Avant de comparer la syntaxe des noms des éléments, Cupid élimine les ponctuations, les prépositions et les conjonctions et explicite les abréviations et acronymes.
- La seconde phase est une comparaison structurelle globale des éléments des schémas, basée sur la similarité de leurs contextes ou proximités. Exemples : deux éléments sont en correspondance car leurs parents sont en correspondance et les autres fils des parents sont en correspondance.
- La troisième phase est l'étape de la génération du modèle de correspondances. Les correspondances sont créées en choisissant la paire d'éléments qui a la plus grande mesure de similarité.

L'opérateur *Merge* [Pottinger et al., 2003] prend en entrée deux modèles A et B, un mapping MapAB et une désignation du modèle préféré de A ou B, et il retourne un troisième modèle qui respecte le modèle de Mapping. Merge propose des règles de fusion en langage informel et formel ainsi qu'un algorithme de fusion.

Cette approche est intéressante du fait qu'elle s'intéresse à l'identité, l'abréviation, l'acronymie, la synonymie et la structure globale des éléments. Cependant, elle ne couvre pas toutes les autres hétérogénéités et ne compare pas la structure locale des éléments de modèles.

- **Approche de composition de modèles : Opérateur Merge de MOMENT [Boronat et al., 2005] [Boronat et al., 2007]**

Dans cette approche, les auteurs [Boronat et al., 2005] ont proposé un framework générique, appelé MOMENT (MOdel manageMENT), intégré dans la plateforme Eclipse, et qui fournit un ensemble d'opérateurs génériques pour manipuler les modèles exprimés dans différents langages, à travers

Eclipse Modeling Framework (EMF). Les opérateurs simples sont l'union, l'intersection, la fusion, la différence entre les modèles, la transformation d'un ensemble de modèles vers un autre, la navigation à travers le mapping, etc. Les opérateurs complexes peuvent être définis par la composition des autres opérateurs.

Dans MOMENT, l'opérateur Merge reste totalement réutilisable pour tout méta-modèle. MOMENT fournit les opérateurs génériques avec des constructeurs nécessaires pour spécifier un modèle en tant qu'ensemble d'éléments, basés sur une spécification axiomatisée d'un ensemble de théories. La spécification de l'opérateur Merge peut également être personnalisée à un méta-modèle en ajoutant de nouveaux axiomes.

L'opérateur Merge prend deux modèles en entrée et produit un troisième. Si A et B sont des modèles dans un méta-modèle spécifique, l'application de l'opérateur Merge sur A et B produit un modèle C qui comprend les éléments de A conjointement avec les éléments de B, c'est-à-dire l'union de A et de B. En tenant compte du fait que les doublons ne sont pas autorisés dans le modèle, l'union est disjointe.

L'opérateur Merge implique trois autres opérateurs. L'opérateur Equals définit une relation d'équivalence entre les éléments appartenant à différents modèles, mais au même méta-modèle. L'opérateur Resolve sert à résoudre les conflits qui surviennent lorsque des éléments sont sémantiquement équivalents mais représentés syntaxiquement différemment. La stratégie de résolution des conflits s'appuie sur la notion de modèle préféré. En effet, lorsque l'opérateur Merge est appliqué à deux modèles, il faut que l'un soit choisi comme préféré. Lorsqu'il y a un conflit, les éléments appartenant au modèle préféré l'emportent. Des rafraîchissements sont nécessaires pour copier des éléments non-dupliqués dans le modèle fusionné. Ceci est assuré par l'opérateur Refresh.

Dans [Boronat et al., 2007], les auteurs présentent une spécification de l'opérateur, qu'ils ont implémentée dans EMF. Ils l'appliquent pour la fusion des diagrammes de classes UML partiels (à partir des cas d'utilisation de Diagramme de Cas d'Utilisation) dans un contexte de processus de développement logiciel dirigé par les cas d'utilisation. Chaque cas d'utilisation est transformé en un diagramme de classes UML partiel, qui représente une partie du diagramme de classes. L'ensemble de ces diagrammes de classes partiels doivent être fusionnés.

Dans l'algorithme proposé de la comparaison de modèles, les relations d'équivalence qui relient deux éléments prennent en compte le type d'un concept (par exemple, les deux doivent être des classes), son nom, et son conteneur (sa structure globale). Les noms de deux éléments qui ont le même type sont analysés en trois étapes : deux éléments peuvent être égaux s'ils ont exactement le même nom, sinon, deux éléments peuvent être égaux si leurs noms sont définis comme des synonymes dans un thésaurus, sinon, ils peuvent être égaux si une fonction heuristique pour comparer des chaînes établit qu'elles sont semblables avec une distance acceptable. Dans cette approche, la comparaison de la structure locale des éléments de modèles n'est pas prise en compte.

- **Approche de comparaison de modèles de [Uhrig, 2008]**

Les travaux de [Uhrig, 2008] proposent une méthode de comparaison de diagrammes de classes UML. L'idée consiste à calculer une métrique, appelée coût minimal, de chaque élément du premier diagramme avec tous les éléments du deuxième diagramme et en prendre le minimum. Le coût

dépend des noms, des attributs (noms et types), des opérations (noms et types de retour) des deux éléments, et des associations reliées avec chaque élément (noms et cardinalités).

Ce travail réalise une comparaison syntaxique en testant l'identité des noms des termes. Il inclut aussi une comparaison structurelle locale et globale.

- **Approche de composition de Atlas Model Weaver (AMW) [Didonet et al., 2005a] [Didonet et al., 2005b] [Didonet et al, 2006] [Bézivin et al., 2006]**

Atlas Model Weaver (AMW) est un module de AMMA (ATLAS Model Management Architecture) [Didonet et al., 2005a], une plateforme de gestion de modèles. Ce module est destiné particulièrement à la création de relations (i.e. des liens) entre les éléments de modèles (ou méta-modèles).

AMW traite les diagrammes de classes. A présent, AMW utilise EMF (Eclipse Modeling Framework) comme le handler (gestionnaire) de ses modèles. Il utilise un langage, appelé langage de tissage (weaving language), contenant une partie fournissant les concepts génériques de base permettant de créer les liens entre les modèles. Ces liens sont sauvegardés dans les modèles de tissage (weaving models). Le méta-modèle de tissage de base de AMW (cf. Figure 3-6) contient les concepts de tissage de base. WElement est l'élément basique de tous les éléments du méta-modèle de tissage, WModel représente la racine du modèle de tissage. WLink représente des liens entre les éléments des modèles. WLinkEnd indique le type des éléments pouvant être composés [Didonet et al., 2005b] [Didonet et al, 2006] [Bézivin et al., 2006].

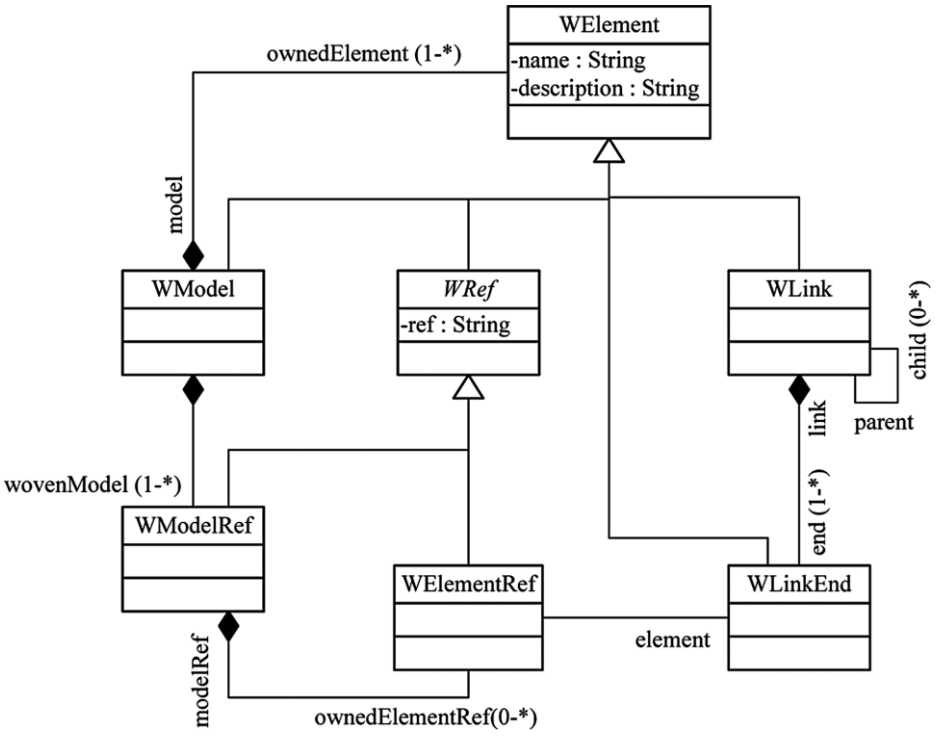


Figure 3-6 Méta-modèle de tissage de base de AMW [Bézivin et al., 2006]

Les liens créés par les concepts n'ont aucune sémantique. Pour la tâche de composition, AMW fournit la capacité d'étendre le méta-modèle de tissage de base afin de construire les nouveaux concepts de tissage spécifiques au domaine de composition de modèles (i.e. fusion, remplacement, union,

surcharge, etc.). Ce sont les concepts WLink et WLinkEnd que l'on peut étendre pour ajouter les nouveaux types de relation de composition. Cette extension permet d'avoir un langage de tissage dédié à la composition de modèles. De façon similaire, pour étendre le méta-modèle de tissage de base, on peut créer aussi une famille de méta-modèles de tissage pour la comparaison, la traduction et même la transformation de modèles, etc.

Les auteurs se sont basés sur l'opérateur *Match* proposé par [Bernstein, 2003] pour trouver les liens de mapping entre les éléments de modèles, et donc créer le modèle de tissage. Les auteurs dans [Jossic et al., 2007] ont proposé une valeur de similarité qui caractérise l'équivalence entre les éléments. Le calcul de la similarité est exprimé par un algorithme. Cette valeur est égale à 1 s'il s'agit exactement des mêmes entités, c'est-à-dire (i) il existe une similarité des chaînes et une similarité de type (les deux sont des classes ou les deux sont des attributs, etc.) ou (ii) ces entités ont le même contenant, les mêmes descendants (équivalence structurelle globale) et les mêmes caractéristiques structurelles locales. La valeur de similarité vaut 0 sinon.

3.2.4. Approches hybrides

Cette section présente le détail des approches de composition hybrides proposant la combinaison des quatre types de comparaison (syntaxique, sémantique, structurel local et structurel global).

- **Approche de composition de modèles de [Haddar et al., 2002]**

Dans [Haddar et al., 2002], les auteurs présentent une approche formelle pour la composition de Représentations Conceptuelles (RCs) (modèles appartenant à différents méta-modèles), exprimées éventuellement dans des formalismes différents (entité/association, orienté objet, etc.), pour avoir une seule représentation conceptuelle globale. Le processus de composition suivi comporte les phases suivantes : la phase de pré-composition, de comparaison et de fusion.

La phase de pré-composition traduit les RCs initiales dans un Modèle Commun (MC). Le MC choisi est le langage UML. La transformation des RCs vers le MC UML est réalisée au niveau du méta-modèle, suivant des règles de transformation exprimées avec le langage de spécification formel Object-Z [Smith, 2000].

La phase de comparaison a été réalisée pour les noms de termes. Cette étape inclut (1) la comparaison syntaxique qui teste l'*identité* de deux noms, et (2) la comparaison sémantique qui, en s'appuyant sur une ontologie de domaine, teste la *synonymie*, la *polysémie* et le lien de *généralisation/spécialisation* entre les termes. La comparaison inclut aussi (3) l'aspect structurel. Ceci consiste à comparer deux classes en comparant leurs structures locales (attributs spécifiques et hérités) et leurs structures globales (ensemble de classes qui sont reliées à une classe (spécifiques et hérités)). Dans cette approche, la comparaison prend en compte même (4) l'aspect dynamique. Il s'agit de comparer deux classes en comparant leurs opérations (spécifiques et héritées) et leurs diagrammes d'état qui décrivent le comportement de leurs objets face aux événements qu'ils peuvent subir tout au long de leurs cycles de vie.

Les résultats de la comparaison sont enregistrés dans une table de comparaison (modèle de correspondances), pour une utilisation dans l'étape de fusion.

Phase de fusion : la fusion de RCs consiste à prendre deux à deux les représentations conceptuelles et à créer une nouvelle représentation conceptuelle, bien formée et cohérente, contenant toutes les informations des RCs d'origine. La fusion est guidée par la table de comparaison qui montre toutes les correspondances détectées pendant la phase de comparaison.

L'originalité de ces travaux provient, d'abord, du fait que les règles de comparaison et de fusion ont été exprimées par un langage de spécification formel Object-Z. Ce langage offre les concepts orientés objet et constitue une extension du langage Z. De plus, ces travaux ont inclus aussi les aspects comportementaux des représentations conceptuelles dans le processus de comparaison. Enfin, l'association des différents types de comparaison utilisés, permet d'avoir une comparaison trouvant un nombre important de correspondances. Toutefois, cette approche n'assure pas tous les aspects syntaxiques tels que l'inclusion, l'acronymie et l'abréviation et tous les aspects sémantiques tels que l'inverse et la disjonction.

- **Approche de composition de modèles de [France et al., 2004] [Reddy et al., 2006] [Fleurey et al, 2007a] [Fleurey et al., 2007b]**

Les travaux de [France et al., 2004] [Reddy et al., 2006] [Fleurey et al, 2007a] offrent une approche générique de composition de modèles indépendants d'un langage de modélisation. Dans cette approche, les auteurs proposent un méta-modèle générique décrivant le comportement d'un opérateur de composition, fondé sur la notion de directives de composition. Cette approche peut être spécialisée pour un méta-modèle particulier afin d'ajouter des capacités particulières pour la composition de modèles dans un langage particulier.

Notons que cette approche a commencé historiquement par la perspective de la composition dans l'approche de modélisation orientée aspect [Reddy et al., 2006] [France et al., 2007], c'est-à-dire la composition de modèles d'aspect modélisant les préoccupations transverses d'une application, avec un modèle de base appelé *modèle primaire* qui représente le cœur fonctionnel de l'application. Les auteurs présentent le méta-modèle de composition des modèles orientés aspects dans [Reddy et al., 2005] [Reddy et al., 2006] [France et al. 2007] et les directives de composition des modèles orientés aspects dans [Reddy et al., 2006].

La composition de modèles est réalisée en deux phases : une phase de comparaison et une phase de fusion.

Phase de comparaison : cette phase est spécifique au langage de modélisation. Les auteurs proposent la comparaison des diagrammes de classes UML orientés aspects [Reddy et al., 2006]. Le mécanisme de détermination des éléments à fusionner se base sur la comparaison de leurs signatures. Une signature est un ensemble de valeurs extraites de certaines propriétés de l'élément du modèle. L'ensemble des propriétés utilisées pour déterminer une signature est appelé type de la signature. Par exemple, le type de la signature d'une classe UML consiste en les propriétés name et isAbstract. La signature d'une classe concrète, par exemple Etudiant, sera « name = Etudiant », « isAstract = false ». La signature de l'opération *Update(x : int, y : int)* est l'ensemble {*update, (x : int, y : int)*}. Dans ce cas, deux opérations sont en correspondance si elles ont le même nom et les mêmes paramètres. L'approche propose une comparaison syntaxique en testant l'identité des éléments et une comparaison structurelle locale et globale des éléments de modèles.

Phase de fusion : cette phase est indépendante du langage de modélisation. Les éléments de modèles qui ont la même valeur de signature vont être fusionnés. Dans le cas contraire, ils seront simplement copiés vers le modèle final. Pendant le processus de fusion, il est probable qu'auront lieu des conflits. Cette approche utilise deux types de directives de composition pour les résoudre. Les *directives de pré-Fusion* spécifient des modifications sur les modèles avant qu'ils ne soient fusionnés. Ces changements peuvent forcer ou rejeter les correspondances entre les éléments de modèles. Par exemple, dans le cas de deux classes qui représentent le même concept et qui apparaissent dans deux modèles avec deux noms différents (synonymie), une directive de pré-fusion de type *Rename* est appliquée dans un modèle pour changer le nom de la classe de telle sorte qu'elle corresponde à la classe de l'autre modèle. Les *directives de post-Fusion* spécifient des modifications sur le modèle composé, comme le changement de nom, ajout, suppression ou remplacement des éléments de modèle.

Cette approche a été implémentée dans un outil Open source appelé Kompose [Fleurey et al., 2007b]. Kompose[19] a été implémenté en utilisant le langage Kermeta et a été intégré dans l'environnement de développement intégré Eclipse.

L'intérêt de cette approche, est qu'elle permet de concevoir des opérateurs de composition spécifiques à chaque langage de modélisation, tout en gardant un noyau générique commun. Cependant, même si elle combine quatre types de comparaison, elle ne couvre pas tous les types de comparaison les modèles tels que la disjonction, l'inverse, l'inclusion, etc.

- **Approche de composition de modèles de [Anwar et al., 2006] [Anwar et al., 2007a] [Anwar et al., 2007b] [Anwar et al., 2008a][Anwar et al., 2008b] et [Anwar et al., 2010]**

Dans [Anwar et al., 2006] [Anwar et al., 2007a] [Anwar et al., 2007b] [Anwar et al., 2008a][Anwar et al., 2008b] et [Anwar et al., 2010], les auteurs proposent la composition des modèles orientés points de vue. La modélisation orientée points de vue permet de modéliser séparément les besoins des acteurs interagissant avec le système étudié. Cette approche a été mise en œuvre dans le profil VUML (*V*iew based *U*nified *M*odeling *L*anguage) qui offre un formalisme pour modéliser un système logiciel par une approche combinant objets et points de vue.

Le processus de composition de modèles est défini en trois phases : une phase manuelle de pré-composition, une phase automatique de composition et une phase semi-automatique de post-composition.

Phase de pré-composition : L'objectif principal de cette phase est d'harmoniser les modèles de conception par point de vue de façon à éliminer les éventuels conflits résultant des modélisations séparées. Dans cette étape, une comparaison sémantique est réalisée manuellement, elle permet de détecter la synonymie et la polysémie. Pour résoudre ces conflits, la décision du concepteur de procéder par renommage des éléments est nécessaire.

Phase de composition : la deuxième phase du processus de composition, automatisable, est réalisée en deux étapes. La première étape, de mise en correspondance, est guidée par un ensemble de règles dont le but est d'identifier les correspondances entre les éléments des modèles partiels. Cette phase compare syntaxiquement (teste l'identité), et structurellement local et global les éléments de modèles.

[19] http://www.kermeta.org/docs/wink/kompose.swf

Les différentes relations de correspondance, créées au cours de cette étape, sont stockées dans un modèle séparé appelé *modèle de correspondances*.

La deuxième étape, l'étape de fusion, consiste à composer les modèles initialement mis en correspondance. Cette étape est réalisée par l'application d'un ensemble de règles (fusion, et translation). L'idée est de parcourir les liens de correspondance établis dans l'étape précédente et de déterminer selon le cas, les règles de fusion à appliquer. Les éléments qui ne sont pas reliés sont simplement retranscrits dans le modèle VUML par le biais de règles de translation.

Phase de post-composition : une fois le modèle VUML généré, la troisième phase consiste à vérifier les propriétés syntaxiques et sémantiques propres au profil VUML (par exemple : Une classe stéréotypée par 'view' a au plus un parent).

Pour automatiser le processus de composition, les auteurs ont décidé de l'ancrer dans une approche IDM en explicitant les transformations sous forme de règles. Ils ont classé les différentes règles de transformation en règles de correspondance, règles de composition et règles de translation. En effet, la composition de modèles par point de vue est clairement une suite de transformations de modèles (transformations exogènes puisque les méta-modèles en entrée et en sortie sont différents). En effet, la première transformation permet de créer un modèle de correspondances dans lequel sont stockées les différentes relations de correspondance entre les modèles. La deuxième transformation est composée de deux transformations parallèles : la première permet de construire un modèle intermédiaire en utilisant le modèle de correspondances et les règles de composition, et la deuxième a pour objectif d'affiner ce modèle préliminaire et de créer le modèle composé VUML en utilisant des règles de translation.

Les auteurs ont étendu le méta-modèle de correspondances AMW pour définir les différentes relations entre les éléments des modèles partiels. Les règles sont décrites de manière informelle. L'approche est ensuite formalisée en exprimant ces règles sous forme de définitions mathématiques. Dans un contexte de développement dirigé par les modèles, la transformation de modèles est décrite formellement par des opérations de transformation de graphes.

L'implémentation de l'approche a été réalisée en utilisant la plate-forme de développement open source Eclipse. Ce choix est justifié par l'existence d'un grand nombre d'outils et de technologies autour de l'IDM tels que les API graphiques comme GEF (Graphical Editing Framework) qui permet de visualiser graphiquement les modèles manipulés et les technologies de gestion de modèles comme EMF (Eclipse Modeling Framework) et GMF (Graphical Modeling Framework) qui a permis d'implémenter le profil VUML. Les auteurs ont choisi le langage de transformation ATL (Atlas Transformation Language) qui est considéré maintenant comme un standard de transformation dans Eclipse.

Cette approche ne couvre pas tous les aspects de comparaison syntaxique et sémantique, à savoir l'abréviation, l'acronymie, l'inclusion, la disjonction, l'inverse et l'hyponymie. De plus, même la résolution des conflits de synonymie et de polysémie s'effectue manuellement. La principale limite de cette approche est la résolution manuelle de conflit au niveau des modèles sources avant la composition.

- **Approche de composition de modèles de [Oliveira et al., 2007] [Oliveira et al., 2008] [Oliveira et al., 2009]**

Dans cette approche, les auteurs de [Oliveira et al., 2007] [Oliveira et al., 2008] proposent un guide pour la composition de diagrammes de classes UML. Ils considèrent la composition de modèles comme un cas particulier de la transformation de modèles, qui prend en entrée deux modèles MA et MB et combine leurs éléments en un modèle MAB. L'approche de composition de modèles proposée par ces travaux est décrite par quatre phases.

Phase initiale : la première étape est l'identification et l'analyse des modèles en entrée. Le but est de définir les types des éléments des deux modèles, par exemple stéréotypes, classes, ou associations, afin que l'opérateur de matching décide, dans l'étape suivante, si les éléments sont en mesure d'être comparés ou non. **Phase de comparaison :** le but de cette phase est de trouver les éléments de modèles équivalents. Les auteurs proposent des règles de matching représentées dans un langage mathématique formel. La comparaison consiste à calculer une mesure en fonction du [Oliveira et al., 2009] : Degré d'équivalence syntaxique : il s'agit de comparer syntaxiquement les éléments des deux modèles. Pour ce faire, ils se basent sur l'algorithme de [Manning, 1999], cité précédemment avec un N=2. Degré d'équivalence sémantique : il s'agit d'identifier les synonymes en se référant à un dictionnaire de synonymes. Degré de similarité basé sur les signatures : les auteurs ont étendu l'approche de [Reddy et al., 2006] pour comparer les signatures des éléments des modèles. Ils calculent un degré de similarité entre les signatures des éléments, contrairement à [Reddy et al., 2006] qui teste uniquement si deux éléments sont en correspondance. Degré de similarité de l'alignement des ontologies : il s'agit de la comparaison de deux ontologies équivalentes aux modèles. Les travaux s'appuient sur la stratégie proposée par [Felicissimo, 2004] qui est implémentée par l'outil CATO [Breitman et al., 2005]. **Phase de fusion :** les éléments de modèles doivent être proprement fusionnés en appliquant des règles de transformation. Afin de produire le modèle résultat à partir de la composition de modèles, et afin qu'il soit sans conflits, il faut appliquer un ensemble de règles. Ces règles peuvent être : la création et la suppression d'éléments, l'ajout et la suppression à partir d'un package, le renommage des éléments du modèle, la fusion de deux modèles selon une stratégie de fusion. Ces règles sont exprimées dans un langage informel. **Phase post-composition** : l'objectif de cette phase est de découvrir les erreurs de conception, les propriétés indésirables et les conflits. Tous les modèles composés sont vérifiés par rapport à des règles de bonne formation (well-formedness) en vue d'identifier les modèles qui ne sont pas bien formés, tels que : une classe avec une élévation du nombre d'attributs, une opération avec un nombre excessif de paramètres, un nom d'attribut trop long, etc. Pour chaque problème découvert, sont identifiées des règles de transformation, afin de résoudre les problèmes et les conflits.

Le principal avantage de cette approche est la combinaison de plusieurs stratégies de comparaison, qui permetent d'augmenter la chance de trouver toutes les correspondances entre les deux modèles. Néanmoins, même si elle utilise les quatre types de comparaison, cette approche ne couvre pas l'abréviation, l'acronymie, la polysémie, la disjonction, l'inverse et l'hyponymie.

3.3. Comparaison et évaluation des approches de composition de modèles

Nous avons établi un tableau comparatif (cf. Tableau 3-1) basé sur les critères définis précédemment, qui présente une synthèse des approches citées ci-dessus.

Le tableau comparatif est divisé en trois parties, chacune correspondant au type d'approche étudiée, signalée dans la première colonne. La seconde colonne identifie l'approche par son nom et/ou sa référence. Nous trouvons dans les colonnes suivantes, les six autres critères de comparaison, avec pour le critère spécifiant le type de comparaison, une division en quatre sous-types: syntaxique (St), sémantique (Sm), structurel global (Gl) et structurel local (Lc).

Type d'approche	Approche	Type de modèles à composer	Type comparaison				Fusion	Type de composition	Langage de composition	Outil
			St	Sm	Gl	Lc				
Approches syntaxiques	Ngram [Manning, 1999]	Indépendants du méta-modèle	x	-	-	-	-	Algorithme qui génère une mesure	Algorithmique	-
	EML [Dimitrios et al., 2006a]	Indépendants du méta-modèle (MM)/ Diagrammes de classes	x	-	-	-	x	Règles de comparaison, fusion et de transformation	EML	Implémenté sous Eclipse
	[Rungworawut, 2010]	Diagramme de classes et pattern	x	-	-	-	x	Règles de comparaison et de fusion	Langage formel	-
	[UML, 2011]	Packages UML (diagrammes de classes)	x	-	-	-	x	Règles de transformation	Langage informel	-
Approches semi-hybrides	Opérateurs Match et Merge de Model Management [Bernstein, 2003]	Indépendants du M.M	x	x	x	-	x	-Mesure -Algorithme de comparaison -Règles de fusion -Algorithme de fusion	-Algorithmique -Langage informel -Langage formel	-
	Opérateur Merge de MOMENT [Boronat et al., 2007]	Indépendants du M.M/Diagrammes de classes UML (partiels)	x	x	x	-	x	Algorithme	Algorithmique	Framework MOMENT sous Eclipse
	[Uhrig et al, 2008]	Diagrammes de classes UML	x	-	x	x	-	Métrique	Mathématique	-
	[Didonet et al., 2005a]	Diagrammes de classes UML	x	-	x	x	x	-Tissage -Algorithme qui génère une mesure	-Algorithmique -Langage de tissage	Implémenté sous Eclipse
Approches hybrides	[Haddar et al., 2002]	Indépendants du M.M	x	x	x	x	x	Règles de comparaison et de fusion	Langage formel : Object-Z	-
	[France et al., 2004]	Indépendants du M.M/ Diagrammes de classes (M. orientés aspect)	x	x	x	x	x	Directives de composition	Langage (méta-modèle) de composition	Outil open source KOMPOSE implémenté sous Eclipse
	[Anwar et al., 2007a]	Diagrammes de classes UML (M. orientés point de vue)	x	x	x	x	x	-Règles de composition -Règle de transformation	Informel, formel	Règles en ATL sous Eclipse
	[Oliveira et al., 2009]	Diagrammes de classes UML	x	x	x	x	x	-Mesure de similarité -Règles de comparaison et fusion	-Mathématique -Formel -Informel	-

Tableau 3-1 Synthèse des travaux de composition de modèles classés selon les critères de comparaison

Légende du tableau :

- : non supporté ou non décrit dans la littérature

x : pris en compte par l'approche

Nous remarquons que les types de composition les plus utilisés sont les règles de comparaison, de fusion, et les mesures. Viennent par la suite, l'algorithme et les règles de transformation. Le tissage est utilisé dans une seule approche.

Nous constatons aussi que tous les travaux n'ont pas formalisé leur solution. Les autres l'ont exprimée par des langages informels, langages spécifiques pour la composition ou en algorithmique. De plus, toutes les approches n'ont pas proposé un outil de composition de modèles.

Nous nous penchons, plus particulièrement, sur le critère Type de comparaison. Le schéma de la Figure 3-7, permet de mettre en évidence, les limites des approches de composition de modèles et, ainsi, situer l'objectif de nos travaux. L'axe des abscisses représente les approches étudiées, et l'axe des ordonnées représente le détail des types de comparaison. La présence du point exprime que l'approche utilise le type de comparaison.

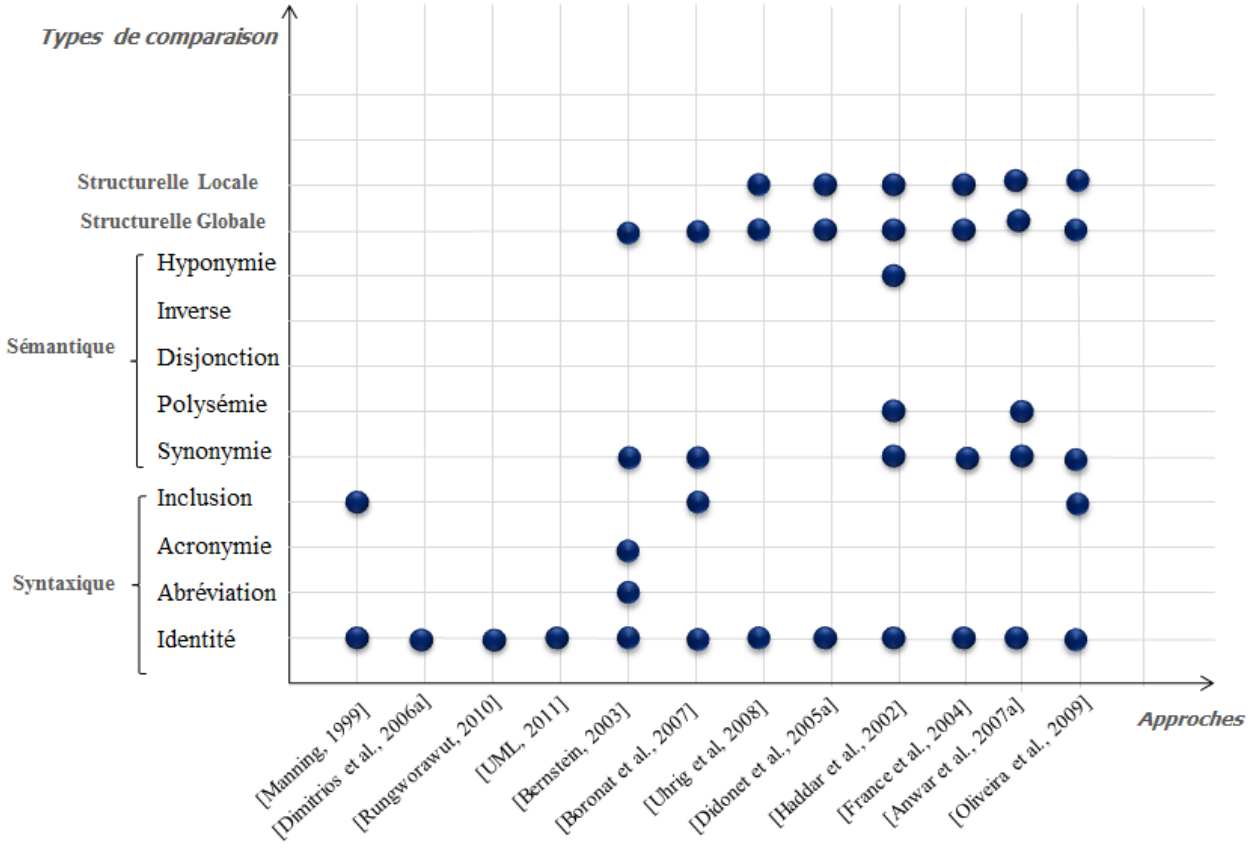


Figure 3-7 Synthèse des approches existantes selon le type de comparaison

Les approches syntaxiques de [Manning, 1999], [Dimitrios et al., 2006a], [Rungworawut, 2010] et [UML, 2011] se limitent à l'identité syntaxique des éléments. Une seule d'entre elles ([Manning, 1999]) rajoute la détection de l'inclusion.

Les approches semi-hybrides [Uhrig et al, 2008] et [Didonet et al., 2005a] ne réalisent pas la comparaison sémantique et se limitent à tester l'identité syntaxique des éléments de modèles. Les deux approches semi-hybrides restantes ne prennent pas en compte l'aspect structurel local, et se contentent de la synonymie dans la comparaison sémantique. Les opérateurs Match et Merge de

Model Management [Bernstein, 2003] ne détectent pas l'inclusion ; quant à l'opérateur Merge de MOMENT [Boronat et al., 2007], il ne traite pas l'abréviation et l'acronymie.

Nous constatons, aussi, qu'aucune approche hybride ne traite l'abréviation et l'acronymie dans la comparaison syntaxique, ainsi que l'inverse et la disjonction dans la comparaison sémantique.

Notre étude de l'existant a montré qu'il n'existe aucune approche détectant les correspondances sémantiques telles que la disjonction et l'inverse. Nous constatons, d'ailleurs, qu'aucune approche ne couvre tous les types de comparaison. Nous pouvons le vérifier dans le diagramme de la Figure 3-7 par le fait qu'aucune approche n'offre une continuité de points.

Notons que le fait de ne pas combiner l'ensemble des types de comparaison existants, diminue les chances de trouver de « vraies » correspondances, ou de « vraies » différences (c'est-à-dire deux éléments réellement équivalents, ou deux éléments réellement différents entre les modèles). En effet, l'absence d'utilisation d'un type de comparaison (syntaxique, sémantique, structurel global ou local), peut engendrer des conflits (problèmes et incohérences) pouvant apparaître après la composition de modèles dans le modèle résultat. En effet, des éléments jugés différents lors de la comparaison mais qui sont réellement équivalents, créent des éléments redondants dans le modèle résultat. Ou encore, des éléments jugés équivalents mais qui sont réellement différents, créent une perte d'information dans le modèle résultat.

Afin d'y remédier, nous proposons une approche combinant tous les types de comparaison et couvrant tous les aspects syntaxique, sémantique, structurel global et structurel local, ce qui va permettre d'augmenter la probabilité d'avoir de réelles correspondances et de réelles différences.

Conclusion

Dans ce chapitre, nous avons présenté les concepts de base de la composition de modèles, les hétérogénéités pouvant exister entre des modèles à composer, les travaux existants proposant la composition de modèles, ainsi qu'une synthèse sous forme de comparaison entre eux. Des limites sont apparues et qui se concrétisent en la non détection de toutes les correspondances, ce qui peut engendrer des conflits dans le modèle résultat.

Notre but est donc de proposer une approche de composition hybride qui intègre les aspects sémantique, syntaxique, structurel local et global. Nous explicitons, dans le chapitre suivant, les détails de notre solution, à savoir le système de composition de modèles en décrivant le processus de composition de modèles ainsi que les règles de composition exprimées en langage informel et formel

Chapitre 4

Approche Co-Models de Composition hybride de modèles UML

4. Approche Co-Models de Composition hybride de modèles UML

Introduction

Les modèles à composer sont généralement construits indépendamment les uns des autres. Nous pouvons retrouver des informations communes dans les modèles, mais modélisées de façon différente. De ce fait, des conflits peuvent apparaître dans le modèle résultat. Ainsi, notre problématique [Benabdellah et al., 2009] consiste à réaliser la composition de modèles, en empêchant l'apparition de conflits dans le système résultat.

Notre objectif est donc de proposer une approche de composition de modèles permettant de générer de réelles correspondances et de réelles différences afin de ne pas avoir de redondance et de perte d'information dans le modèle résultat. Nous prenons alors compte de l'aspect sémantique et utilisons ainsi le concept de l'ontologie.

Nous présentons, dans ce chapitre, notre approche de composition de modèles, qui consiste en un système de composition contenant (1) un système de comparaison qui se base sur des règles de comparaison et (2) un système de fusion/translation qui se base sur des règles de fusion et de translation. Nous proposons aussi un processus qui décrit les étapes de composition de modèles. Enfin, nous présentons l'ensemble des règles de composition en langage informel, puis en langage formel afin de fournir une description précise de la composition de modèles.

4.1. Système Co-Models de composition de modèles

Afin de répondre à notre problématique, notre approche consiste d'abord à introduire la sémantique afin de nous approcher du raisonnement humain pour la comparaison de modèles ; ensuite, à couvrir tous les éléments des types de comparaison (acronymie, inclusion, inverse, abréviation, etc.) afin de détecter toutes les correspondances ; et enfin, de combiner tous les types de comparaison (syntaxique, sémantique, structurel global et structurel local) dans le but d'augmenter la probabilité d'avoir de vraies correspondances et de vraies différences. Par exemple, si deux classes ont la même sémantique, la même structure locale (attributs et opérations) et la même structure globale (associations, relations de généralisation et classes qui les entourent), alors elles sont forcément équivalentes. De la même manière, si deux attributs ont le même nom, le même type et les mêmes multiplicités (structure locale) mais pas la même structure globale (c'est-à-dire n'appartiennent pas à deux classes équivalentes), alors ils sont forcément différents.

En ce qui concerne la fusion/translation, nous avons élaboré un ensemble de directives permettant d'indiquer des actions à appliquer pour décider comment les éléments vont apparaître dans le modèle résultat. Par exemple, comment des éléments synonymes, ou des éléments en relation de polysémie, ou deux associations avec une composition différente, vont-ils apparaître dans le modèle résultat ?

Nous proposons ainsi un système de composition [Benabdellah et al., 2011b], que nous baptisons « Co-Models » pour **Co**mposition of **Models,** qui prend en entrée deux modèles M1 et M2 et produit en sortie le modèle résultat MR. Dans notre approche, nous considérons que les modèles sources ont le même niveau de détail.

Comme nous l'avons modélisé dans la Figure 4-1, ce système comporte deux sous-systèmes : le premier est le système de comparaison de modèles et le deuxième est le système de fusion/translation de modèles.

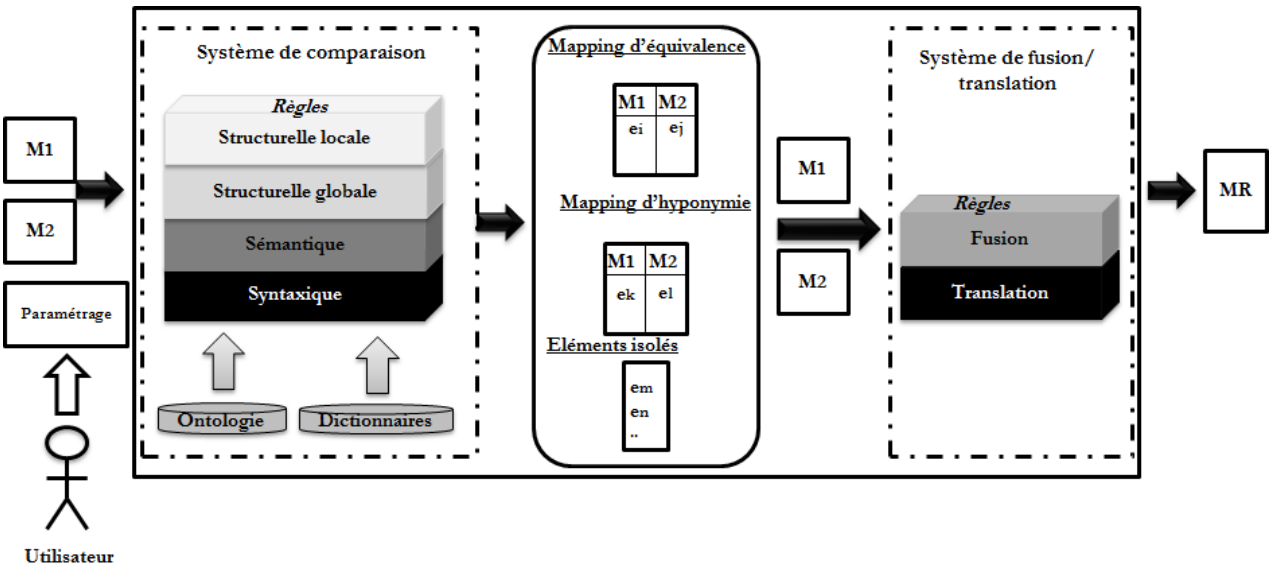


Figure 4-1 Système de composition de modèles

4.1.1. Système de comparaison

Le système de comparaison [Benabdellah et al., 2010c] [Benabdellah et al., 2010e] prend en entrée les deux modèles sources M1 et M2 et produit en sortie (1) le mapping entre les éléments du modèle M1 et du modèle M2, et (2) les éléments qui ne sont pas équivalents, que nous appelons isolés. Ce système permet de détecter les correspondances entre les éléments des modèles, à savoir l'équivalence (mapping d'équivalence) ou l'hyponymie (mapping d'hyponymie). Deux éléments sont hyponymes si l'un d'entre eux est représenté par une classe dans un modèle, et l'autre, par sa sous-classe dans l'autre modèle (une définition plus précise de l'hyponymie est présentée dans les règles de comparaison dans la partie 4.2.5.1).

Notre approche de comparaison étant **hybride** [Benabdellah et al., 2010d] [Benabdellah et al., 2010f] [Benabdellah et al., 2011a], elle intègre les aspects sémantique, syntaxique et structurel local et global, afin d'augmenter la probabilité d'avoir de « réels » mapping et de « réels » éléments isolés. Autrement dit, accroître la probabilité pour que les éléments jugés équivalents (appartenant au mapping d'équivalence) soient réellement des éléments correspondants, pour que les éléments jugés hyponymes (appartenant au mapping d'hyponymie) soient réellement des éléments hyponymes et pour que les éléments jugés isolés soient réellement des éléments non équivalents.

C'est un système basé sur quatre types de règles : celles qui comparent l'aspect syntaxique, celles qui comparent l'aspect sémantique et celles qui comparent l'aspect structurel local et global des éléments de modèles. Afin de prendre en compte l'aspect sémantique, nous avons eu besoin de stratégies s'appuyant sur des propriétés de nature sémantique. Pour cela, le système de comparaison fait référence à un dictionnaire de synonymes[20] et à une ontologie de domaine. Cette dernière va

[20] http://www.synonymes.com/

permettre de fournir des informations sémantiques pertinentes et de prise de décision lors de la comparaison. Par exemple, si nous considérons deux éléments A et C du modèle M1, et deux éléments B et D du modèle M2, le système peut décider, en se référant à l'ontologie de domaine, que A et B sont disjoints, C et D sont équivalents, etc. Dans notre approche, nous considérons que nous disposons de l'ontologie de domaine. Pour ce qui est de la comparaison syntaxique, nous utilisons un dictionnaire d'acronymes[21] et un dictionnaire d'abréviation[22].

Le système de composition de modèles peut être paramétré par l'utilisateur qui précise dans quel cas il considère que deux classes de modèles peuvent être mises en correspondance. A défaut de paramétrage de l'utilisateur, une configuration par défaut est appliquée. Le détail sur le paramétrage est expliqué dans la section 4.1.3.3.

4.1.2. Système de fusion/translation de modèles

C'est un système qui prend en entrée les mappings d'équivalence et d'hyponymie, la liste des éléments isolés et les deux modèles sources, et produit en sortie le modèle résultat (cf. Figure 4-1). Ce système à base de règles de fusion et de translation permet de fusionner les éléments en correspondance et de translater (ajouter) les éléments isolés dans le modèle cible.

Ces règles contiennent aussi des directives qui permettent de déterminer comment vont apparaître, dans le modèle résultat, les éléments présentés de manière différente dans les modèles sources. Pour les éléments équivalents qui ont :

- Une hétérogénéité syntaxique ou sémantique : nous utilisons la solution du modèle principal. En effet, un des deux modèles sources est choisi comme modèle principal, dans le but d'y prendre les noms des éléments en cas de différents entre eux. Par exemple, M1 étant le modèle principal, « Personnel » une classe de M1 et équivalente à la classe « Employé » de M2, alors la classe résultante des deux classes fusionnées prendra le nom de la classe appartenant au modèle principal, soit « Personnel ». L'utilisateur choisit parmi les deux modèles sources, le modèle principal, qui peut être sélectionné en fonction de la taille du modèle : comportant le plus grand nombre de classes (choix par défaut), ou choisi du fait qu'il ait été conçu par l'utilisateur de Co-Models.
  Dans le cas de l'inclusion entre deux éléments, le nom de l'élément résultat est celui possédant le plus de caractères (le nom le plus long). Par exemple, l'attribut résultat des deux attributs équivalents « dateExp » du modèle M1 et « dateExpiration » du modèle M2, sera « dateExpiration ». Cette approche permet de garder le nom fournissant le plus d'informations.
- Une hétérogénéité structurelle locale : les directives de fusion permettent de décider comment la structure locale de l'élément résultat va apparaître dans le modèle résultat. Par exemple, si deux associations sont équivalentes avec une composition différente (l'une est une composition et l'autre une association) alors l'association résultat ne sera pas une composition.
  Ces directives pour l'hétérogénéité structurelle locale ne concernent que les associations et les classes. Pour les autres éléments (attributs, opérations et paramètres), ces directives ne sont

[21] http://acronymes.info/
[22] http://www.meltingmots.com/

pas appliquées, car si les éléments n'ont pas la même structure locale, nous ne les considérons pas équivalents dans le système de comparaison.

- Une hétérogénéité structurelle globale : les directives de fusion permettent de décider comment la structure globale de l'élément résultat va apparaître dans le modèle résultat. Ces directives pour l'hétérogénéité structurelle globale ne concernent que les classes. Pour les autres éléments (associations, relations de généralisation, attributs, opérations et paramètres), ces directives ne sont pas appliquées car si les éléments n'ont pas la même structure globale, nous ne les considérons pas équivalents.

En ce qui concerne les éléments isolés en relation de polysémie, il faut les renommer. En effet, un modèle (le modèle résultat) ne peut pas avoir deux classes avec le même nom ou une classe avec des attributs qui ont le même nom. Par exemple « Famille » et « Famille » étant deux classes en relation de polysémie, il faut les renommer en « Famille1 » et « Famille2 » pour ne pas retrouver deux classes « Famille » avec le même nom (cet exemple ne fait pas partie de notre exemple illustratif présenté dans le chapitre 3).

Le système de fusion/translation prend aussi en entrée les modèles sources afin de prendre des informations non présentes dans le mapping, telles que les multiplicités des associations, la navigabilité des associations, etc.

Nous avons présenté notre système à base de règles de composition de modèles. Nous détaillons, dans la section suivante, le processus qui décrit les étapes de comparaison, de fusion et de translation.

4.1.3. Processus de composition de modèles

Le processus général de composition de deux modèles comprend principalement deux étapes que nous décrivons ci-dessous.

4.1.3.1. Processus de comparaison

Dans notre approche, nous comparons uniquement les éléments de même nature (deux classes, deux attributs, etc.). Le processus de comparaison est modélisé, en Figure 4-2, sous forme de diagramme d'activités. Il englobe les comparaisons syntaxique, sémantique, structurelle locale, structurelle globale et l'hyponymie des classes. Une étape préliminaire est appliquée afin d'éliminer la ponctuation des éléments de modèles. Par exemple l'attribut «Carte_Identité_Nationale» de M2 devient « CarteIdentitéNationale».

Le processus débute en testant la subsomption des classes deux à deux. Il s'agit d'abord de chercher s'il existe une relation de subsomption (héritage) entre elles en se référant à une ontologie. Nous choisissons de ne comparer deux classes syntaxiquement et sémantiquement, qu'après avoir vérifié qu'elles ne sont pas en relation de subsomption. En effet, ceci permet de diminuer le temps de la recherche des classes équivalentes car, à partir du moment où deux classes sont en relation de subsomption, nous ne cherchons plus leur équivalence.

Dans la suite du processus, une comparaison syntaxique des classes deux à deux est effectuée (cf. Figure 4-3), en se référant aux dictionnaires d'acronymes et d'abréviations, si nécessaire. Si les classes ne sont pas syntaxiquement équivalentes, nous les comparons sémantiquement (cf. Figure 4-4), en se référant au dictionnaire de synonymes et/ou à l'ontologie de domaine.

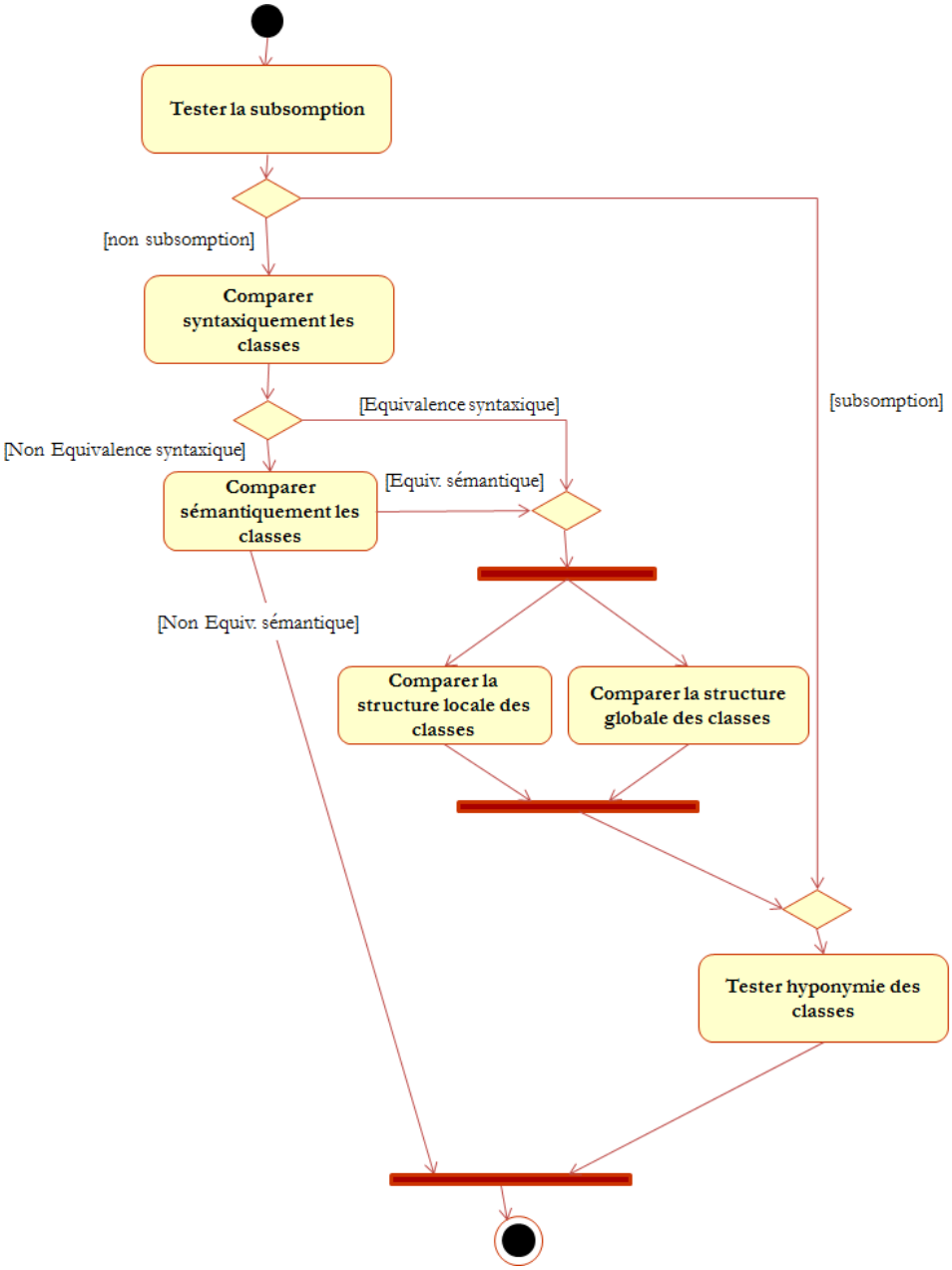


Figure 4-2 Processus de comparaison de modèles

La comparaison syntaxique de deux éléments (classes, attributs, opérations, paramètres ou associations) est présentée dans la Figure 4-3. Nous commençons par tester l'identité des éléments, puis l'inclusion (totale) des éléments qui ne sont pas identiques, puis l'abréviation pour les éléments qui ne sont pas en inclusion, et enfin, l'acronymie des éléments qui ne sont pas en abréviation.

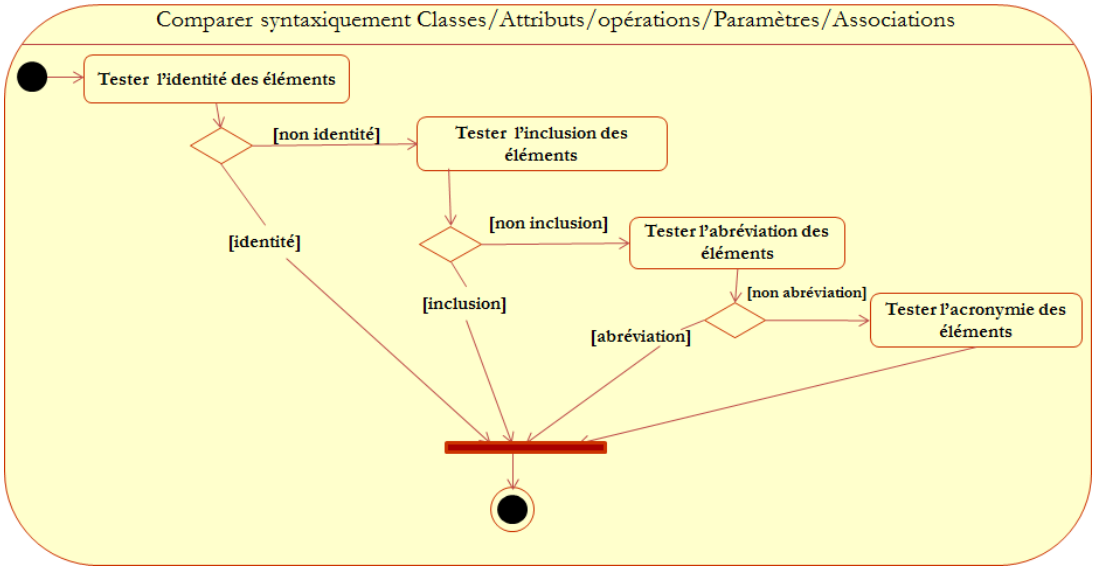


Figure 4-3 Comparaison syntaxique des éléments

La comparaison sémantique de classes (cf. Figure 4-4) commence par le test de la synonymie. Dans le cas où les classes ne sont pas synonymes, nous testons l'équivalence selon une ontologie.

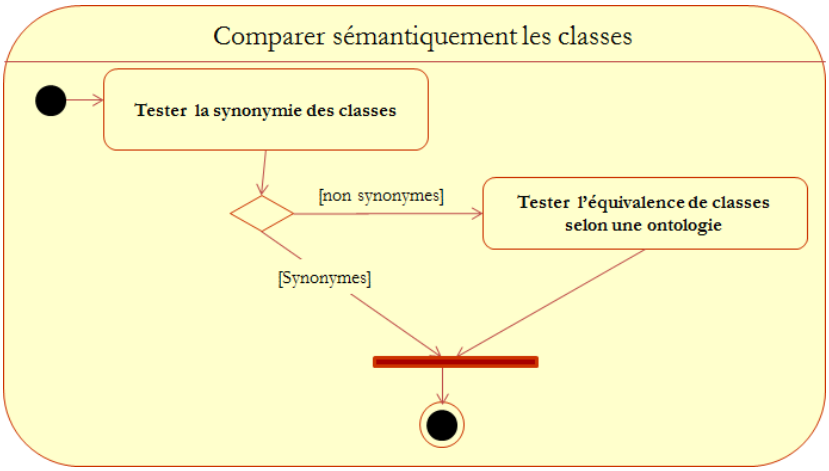


Figure 4-4 Comparaison sémantique des classes

Si les classes sont équivalentes syntaxiquement ou sémantiquement (cf. Figure 4-2), nous comparons ensuite leurs structures locales et globales. La comparaison de la structure locale de deux classes consiste à (cf. Figure 4-5) :

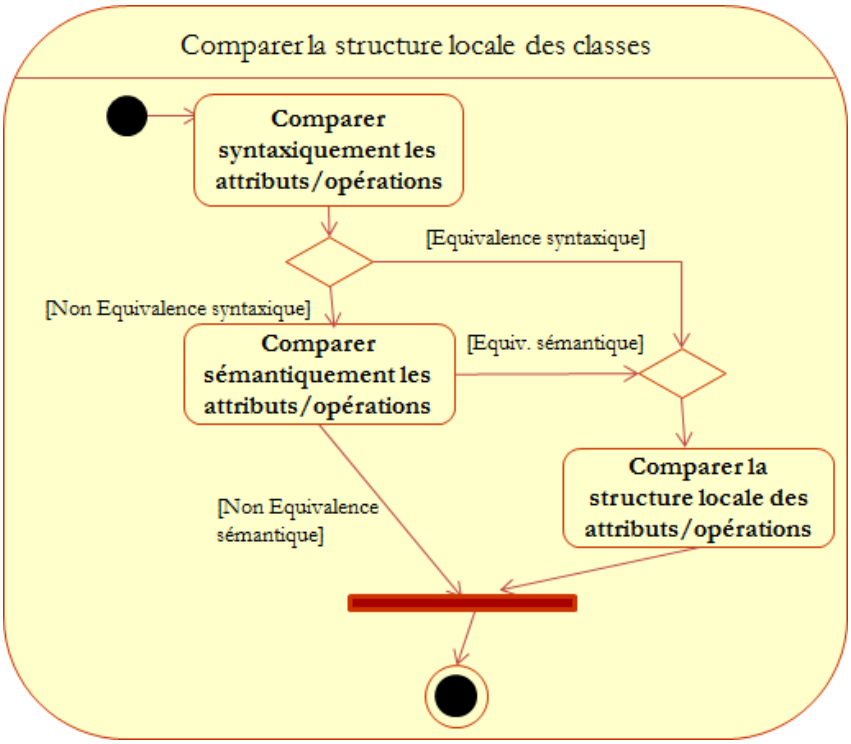


Figure 4-5 Comparaison structurelle locale de classes

- (1) Comparer syntaxiquement leurs attributs deux à deux. Si les attributs ne sont pas syntaxiquement équivalents, nous les comparons sémantiquement. La comparaison sémantique de deux attributs (cf. Figure 4-6) concerne le test de la synonymie, de l'équivalence (selon une ontologie) et de la disjonction. Dans le cas où les attributs sont équivalents syntaxiquement ou sémantiquement, nous comparons ensuite leurs structures locales (types et multiplicités).
- (2) Comparer syntaxiquement leurs opérations deux à deux. Si les opérations ne sont pas syntaxiquement équivalentes, de la même manière que pour les attributs, nous les comparons sémantiquement (cf. Figure 4-7) en testant la synonymie. Dans le cas où les opérations sont équivalentes syntaxiquement ou sémantiquement, nous comparons ensuite leurs structures locales (cf. Figure 4-8). Il faut donc :
  - (a) comparer le type de retour des opérations.
  - (b) comparer syntaxiquement ou sémantiquement tous leurs paramètres deux à deux. La comparaison sémantique des paramètres (cf. Figure 4-9) fait l'objet d'un test de synonymie. Dans le cas où les paramètres sont équivalents syntaxiquement ou sémantiquement, nous comparons ensuite leurs structures locales (leurs types).
  - (c) comparer le nombre de paramètres.

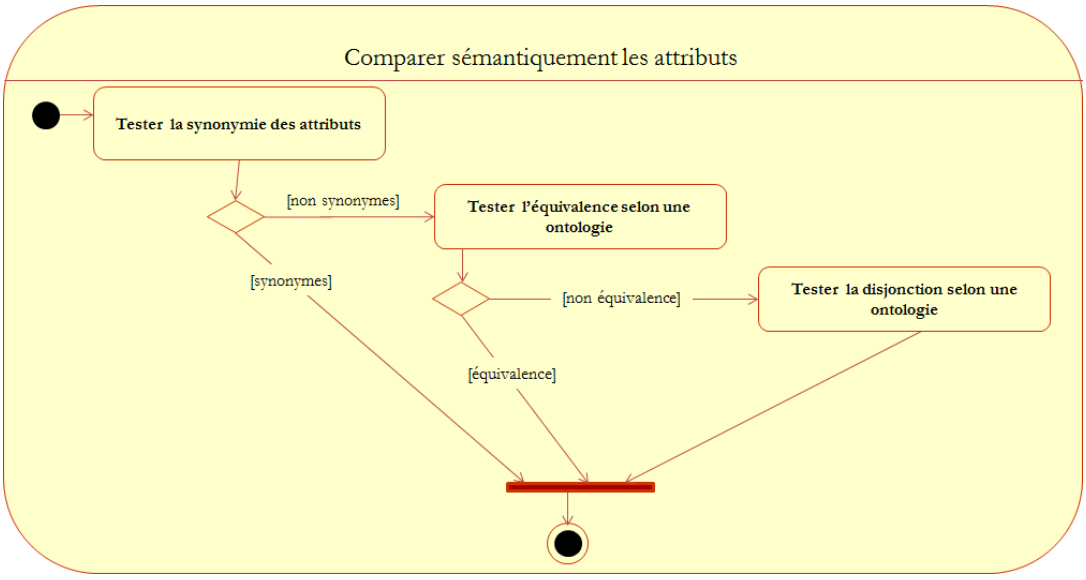


Figure 4-6 Comparaison sémantique des attributs

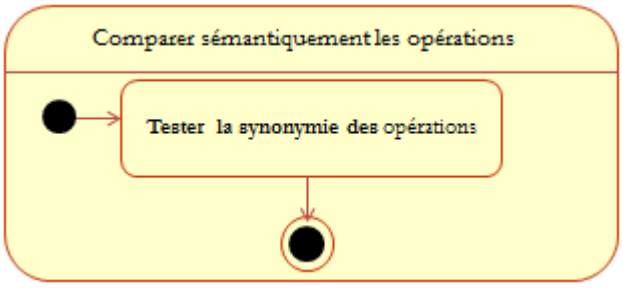


Figure 4-7 Comparaison sémantique des opérations

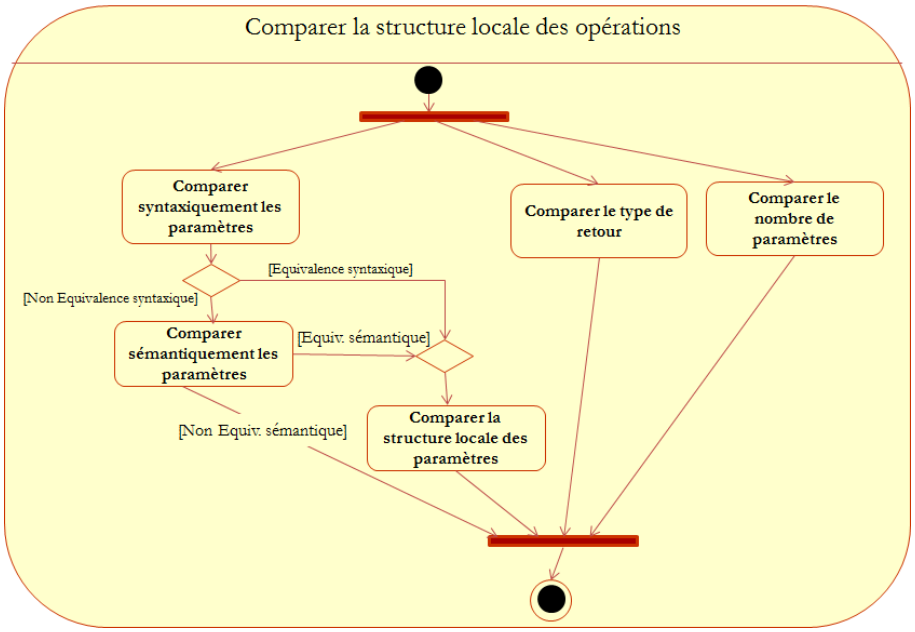


Figure 4-8 Comparaison structurelle locale des opérations

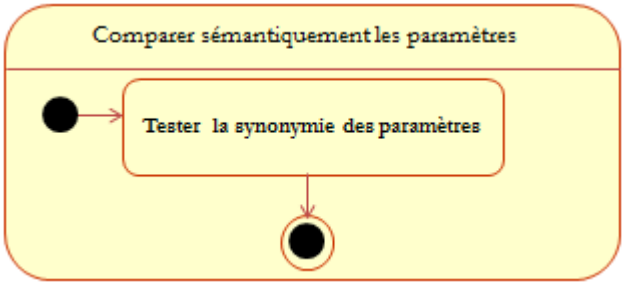


Figure 4-9 Comparaison sémantique des paramètres

En ce qui concerne la comparaison de la structure globale de deux classes (cf. Figure 4-10), elle consiste à comparer toutes les associations et les relations de généralisation qui les entourent.

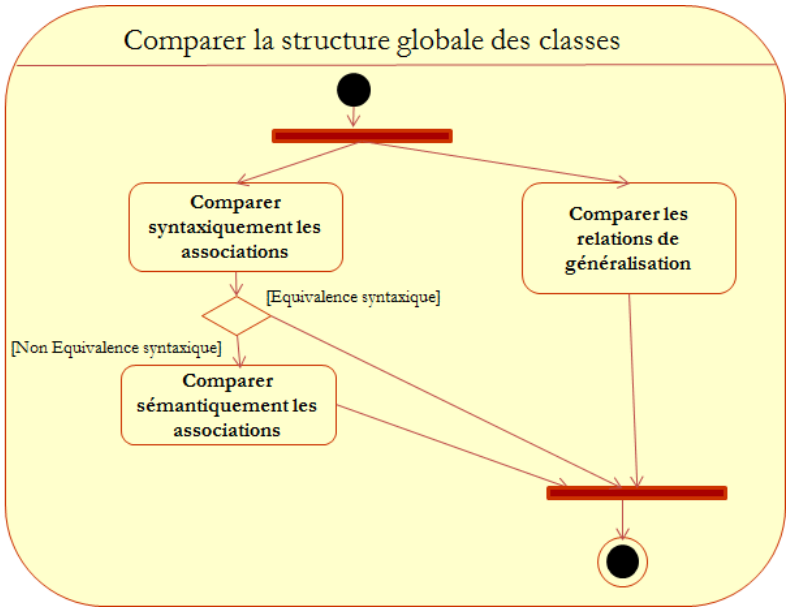


Figure 4-10 Comparaison structurelle globale de classes

La comparaison des associations est réalisée en les comparant syntaxiquement (cf. Figure 4-3), ou sémantiquement dans le cas où elles ne sont pas équivalentes syntaxiquement. Notons que nous ne les comparons pas en structure locale (multiplicités maximale et minimale, navigabilité, composition et extrémités d'association). En effet, nous considérons que deux associations peuvent être équivalentes même si elles sont différentes en structure locale. En d'autres termes, deux associations équivalentes syntaxiquement ou sémantiquement et reliant des classes équivalentes (syntaxiquement ou sémantiquement deux à deux), sont forcément équivalentes, même si elles ont, par exemple, des multiplicités différentes ou une navigabilité différente.

La comparaison sémantique des associations (cf. Figure 4-11) consiste à tester la synonymie, puis l'équivalence selon une ontologie, et enfin l'inverse selon une ontologie.

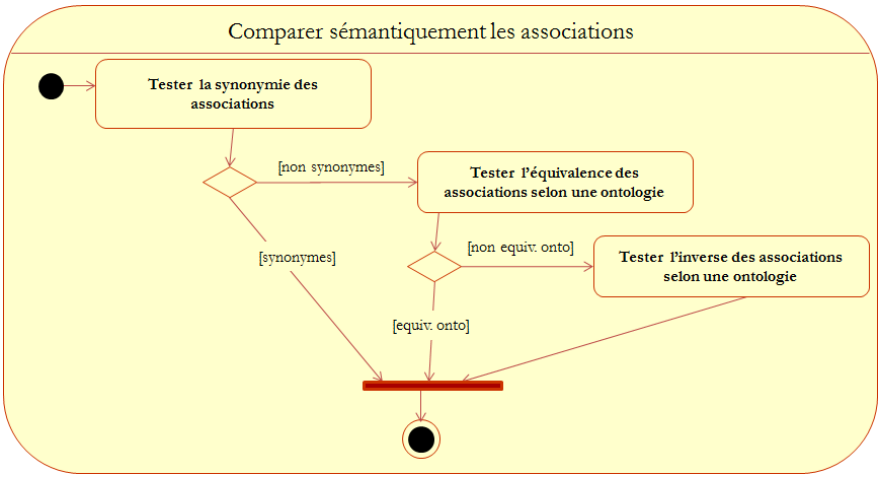


Figure 4-11 Comparaison sémantique des associations

Nous avons décrit, jusqu'ici, toutes les activités du processus de comparaison (cf. Figure 4-2), hormis le test de l'hyponymie. Il commence après avoir terminé la phase de comparaison structurelle locale et globale de toutes les classes. L'hyponymie est testée uniquement pour les classes qui sont en relation de subsomption, étant les seules pouvant être hyponymes.

Une fois l'étape de comparaison achevée, le mapping d'équivalence, le mapping d'hyponymie ainsi que les éléments isolés sont générés. Tous les éléments jugés non équivalents feront partie des éléments isolés. Nous précisons que les classes en relation d'hyponymie, n'étant pas équivalentes avec d'autres classes, apparaîtront donc dans le mapping d'hyponymie et comme éléments isolés.

Notons que nous comparons uniquement les attributs et opérations appartenant à deux classes équivalentes syntaxiquement ou sémantiquement et uniquement les associations/relations de généralisation reliant des classes équivalentes syntaxiquement ou sémantiquement deux à deux. En effet, nous considérons que même si deux attributs/opérations/associations sont équivalents syntaxiquement ou sémantiquement mais ne caractérisent pas les mêmes classes, ils ne sont pas équivalents. Ainsi, nous ne traitons pas la comparaison structurelle globale des attributs/opérations/associations/relations de généralisation, car elle est prise en compte implicitement. La même logique est appliquée pour les paramètres. Nous comparons uniquement les paramètres appartenant à deux opérations équivalentes syntaxiquement ou sémantiquement.

Nous avons présenté le processus de comparaison de modèles. Nous explicitons, dans ce qui suit, le processus de fusion et translation.

4.1.3.2. Processus de fusion/translation

De la même manière que pour le processus de comparaison, nous modélisons, dans la Figure 4-12, le processus de fusion/translation sous forme de diagramme d'activités. Ce processus suit les trois étapes suivantes :

1) Création des classes résultantes (avec attributs et opérations)
2) Création des associations
3) Création des relations de généralisation

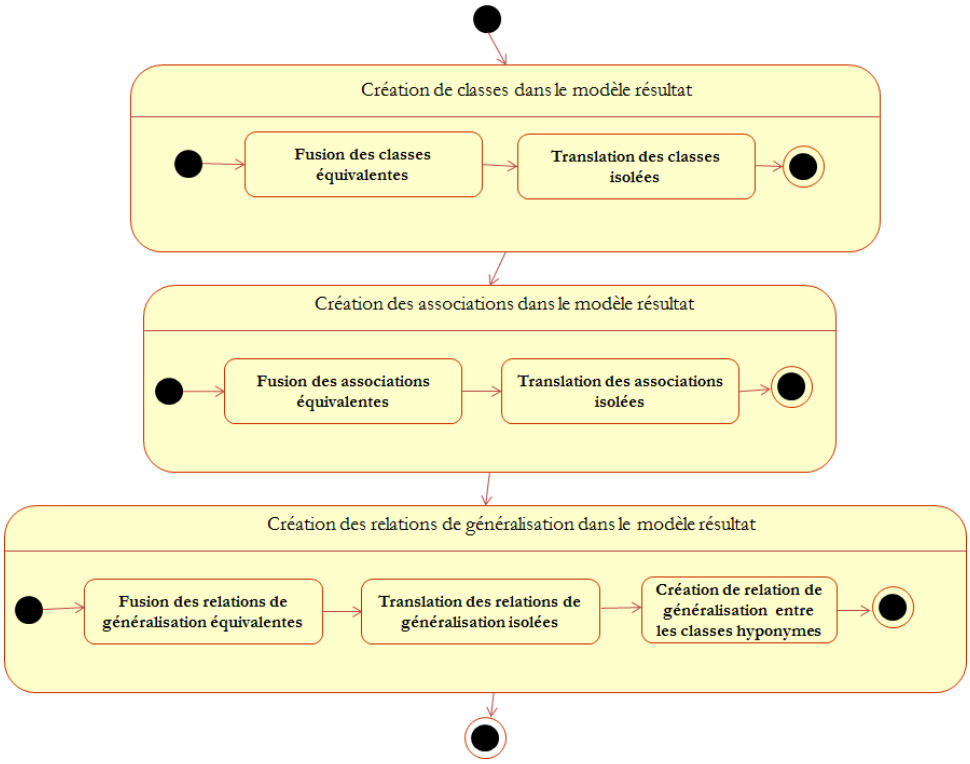


Figure 4-12 Processus de fusion/translation de modèles

La création des classes, dans le modèle cible, concerne la fusion de classes équivalentes, puis la translation des classes isolées.

La fusion des classes équivalentes est modélisée dans la Figure 4-13. Elle consiste en la fusion des attributs et des opérations équivalents, puis la translation des attributs et des opérations isolés.

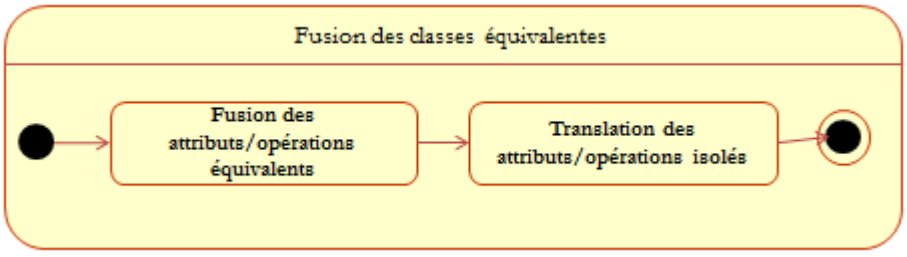


Figure 4-13 Fusion de classes équivalentes

La fusion des opérations inclut la fusion de ses paramètres (cf. Figure 4-14), de même pour la translation, qui inclut la translation de ses paramètres (cf. Figure 4-15).

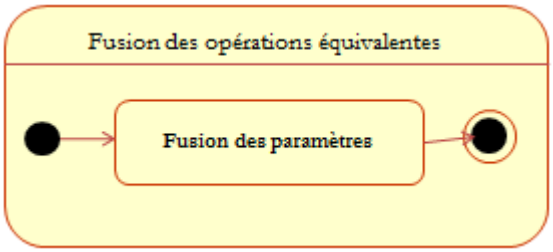


Figure 4-14 Fusion des opérations équivalentes

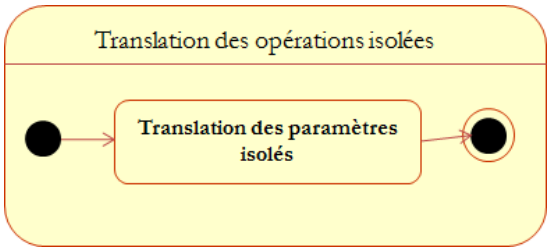


Figure 4-15 Translation des opérations isolées

Concernant la translation des classes isolées, elle inclut (Figure 4-16) la translation des attributs/opérations isolés.

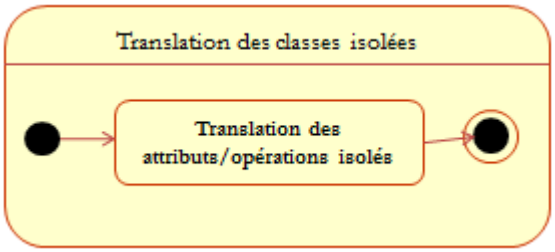


Figure 4-16 Translation de classes isolées

La seconde étape de la création des associations (cf. Figure 4-12), consiste à fusionner les associations équivalentes et à translater celles qui sont isolées.

La troisième étape de la création des relations de généralisation, inclut la fusion des relations de généralisation équivalentes, la translation de celles qui sont isolées ainsi que la création des relations de généralisation reliant les classes hyponymes.

4.1.3.3. Paramétrage du système de composition de modèles

Nous avons choisi de laisser la possibilité à l'utilisateur de Co-Models, de choisir dans quel cas considérer que deux classes de modèles peuvent être mises en correspondance. Ceci permet d'avoir un outil modulable en fonction du besoin de l'utilisateur. En effet, le système peut produire un modèle résultat pour un utilisateur voulant fusionner uniquement les classes qui ont exactement les mêmes attributs et opérations, et exactement le même voisinage ; et pour un autre utilisateur, un modèle résultat fusionnant les classes ayant au moins un attribut ou une opération en commun.

Ces différentes possibilités permettent un paramétrage comportant quatre choix, qui engendrent huit options.

Nous définissons les choix suivants :

- Equivalence de structure locale partielle de deux classes : l'utilisateur souhaite juger équivalentes les classes qui ont au moins un attribut ou une opération en commun.
- Equivalence de structure locale totale de deux classes : l'utilisateur souhaite juger équivalentes les classes qui ont exactement les mêmes attributs et opérations.
- Equivalence de structure globale partielle de deux classes : l'utilisateur souhaite juger équivalentes les classes qui sont liées au moins à un même élément (association, relation de généralisation, classe)
- Equivalence de structure globale totale de deux classes : l'utilisateur souhaite juger équivalentes les classes qui sont liées exactement aux mêmes éléments (associations, relations de généralisation, classes).

La combinaison des choix des équivalences en structure locale et globale donne lieu aux options suivantes :

- Equivalence en structure locale partielle et globale partielle
- Equivalence en structure locale partielle et globale totale
- Equivalence en structure locale totale et globale partielle
- Equivalence en structure locale totale et globale totale
- Equivalence en structure locale partielle
- Equivalence en structure locale totale
- Equivalence en structure globale partielle
- Equivalence en structure globale totale

Une description informelle et formelle des options de paramétrage est exposée dans la section 4.2.5.1 de ce chapitre.

Nous n'offrons la possibilité à l'utilisateur de ne paramétrer que les classes, car nous jugeons que pour que les attributs/opérations/paramètres soient équivalents, il faut qu'ils soient équivalents en structure globale **et** locale. De même pour les relations de généralisation : pour qu'elles soient équivalentes, il faut qu'elles soient équivalentes en structure globale (la structure locale n'existe pas pour la relation de généralisation). Enfin, pour que les associations soient équivalentes, nous exigeons qu'elles soient équivalentes en structure globale et pas nécessairement en structure locale. Cependant, nous considérons que les classes peuvent être équivalentes même si elles ne sont pas équivalentes en structure globale ou en structure locale.

Par défaut, l'option d'équivalence en structure locale partielle et globale partielle est appliquée. Ce choix provient de la logique suivie majoritairement lors des tests manuels de composition de modèles réalisés par des participants (le détail est présenté dans le chapitre 5).

Par ailleurs, l'utilisateur peut aussi paramétrer le système en choisissant un modèle principal parmi les deux modèles sources.

Après avoir présenté notre système à base de règles et le processus général de composition, nous détaillons, dans la section qui suit, les règles de comparaison, de fusion et de translation.

4.1.4. Conclusion

Nous avons présenté le système de composition « Co-Models »**,** qui prend en entrée deux modèles M1 et M2 et produit en sortie le modèle résultat MR. Il se compose d'un système de comparaison qui se base sur des règles de comparaison et d'un système de fusion/translation qui se base sur des règles de fusion et de translation. Nous avons présenté aussi un processus qui décrit les étapes de composition de modèles. Dans la suite du chapitre, nous présentons l'ensemble des règles de composition en langage informel, puis en langage formel.

4.2. Approche de composition par des règles de comparaison, de fusion et de translation

Nous avons défini un cadre mathématique pour exprimer d'une manière formelle notre approche. Nous exploitons la puissance d'expressivité de la logique des prédicats afin de représenter les règles de composition.

Afin d'exprimer ces règles en logique des prédicats, nous avons besoin d'exprimer le modèle UML (représentant les éléments en entrée et en sortie du système Co-Models), l'ontologie de domaine OWL et d'autres ressources (représentant les références du système) en logique des prédicats.

Le mécanisme adopté se base sur des transformations des modèles UML et des ontologies vers une représentation logique. Plus précisément, l'ensemble des transformations se présente comme suit :

- Transformation d'un modèle UML (conforme au méta-modèle UML) vers un modèle logique (conforme au méta-modèle de la logique des prédicats)
- Transformation d'une ontologie (conforme au méta-modèle OWL) vers un modèle logique (conforme au méta-modèle de la logique des prédicats)

La partie suivante vise à présenter les formalisations du modèle UML, d'une ontologie OWL et des dictionnaires, ainsi que des faits supplémentaires utilisés pour l'expression des règles de composition.

Les règles de comparaison [Benabdellah et al., 2010a] [Benabdellah et al., 2010b], de fusion et de translation de modèles sont présentées d'abord en langage informel. Ceci a pour objectif d'en présenter une version simplifiée, illustrée par des exemples. Les exemples, illustrant ce chapitre, sont issus des deux modèles du domaine bancaire, présentés dans le chapitre 3. Les règles sont ensuite présentées en langage formel.

4.2.1. Formalisation d'UML

L'objectif de cette partie est de présenter la transformation d'un modèle UML, plus précisément un diagramme de classes UML, en logique des prédicats. La formalisation de diagramme de classes UML a été proposée dans [Hu, 2008], [Shan et al, 2008] et [Rungworawut, 2010]. Les travaux de [Hu, 2008] et [Rungworawut, 2010] proposent une formalisation basée sur les mathématiques (ensemble, quantificateurs, union, etc.), mais n'étant pas exprimée en logique des prédicats. [Shan et al, 2008] ont formalisé UML en logique des prédicats, mais cela ne correspond pas à nos besoins d'expression.

En effet, cette proposition [Shan et al, 2008] ne permet pas de garantir l'expression de l'unicité des éléments de modèles. Par exemple, une opération x étant exprimée par le prédicat Operation(x), nous pouvons donc retrouver Operation(get) dans une classe dans un modèle, et le même prédicat pour une autre opération get d'une autre classe et d'un autre modèle. Dans notre cas, nous avons besoin d'exprimer l'appartenance de chaque élément, et de déterminer ainsi le modèle auquel il appartient. Le fait qu'une opération « Op » appartient à une classe « C » d'un modèle M sera exprimée par OwnedOperation(Op, C, M, type).

De plus, l'approche [Shan et al, 2008] propose un ensemble de prédicats ne permettant pas d'optimiser l'expression d'un élément. Par exemple, pour exprimer qu'une opération « Op » d'un type « T », appartient à une classe C, il faut utiliser quatre prédicats (contre un dans notre proposition) : Operation (Op), Type (T), ownedOperation(Op, C) et type(Op, T).

Nous définissons une transformation d'un modèle UML vers un modèle en logique des prédicats. La transformation (cf. Figure 4-17) s'effectue dans le contexte de l'IDM, où les modèles UML et logique sont conformes à leur méta-modèle, ces méta-modèles étant conformes au MOF.

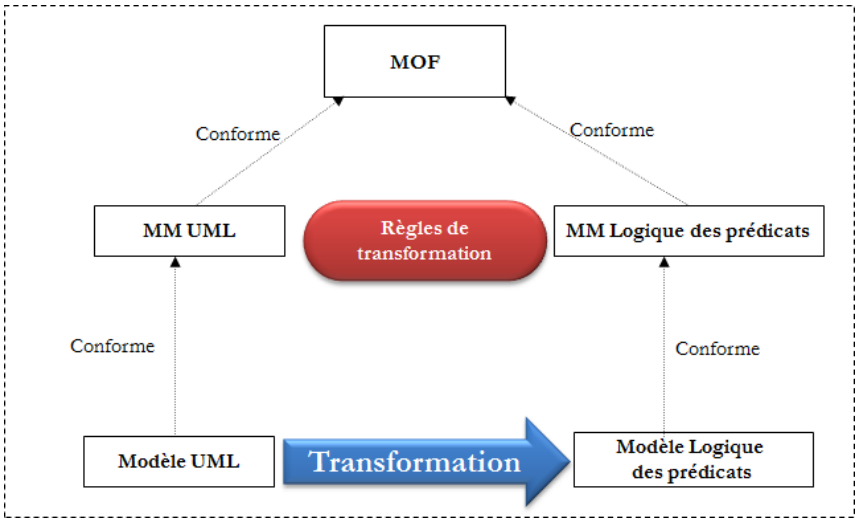


Figure 4-17 Transformation d'un modèle UML vers un modèle logique des prédicats

Avant de présenter les règles de transformation, nous présentons d'abord le méta-modèle UML et celui de la logique des prédicats.

4.2.1.1. Méta-modèle UML

Nous présentons, ci-dessous, l'extrait du méta-modèle du diagramme de classes UML que nous utilisons dans nos travaux. Nous nous sommes référés à la spécification d'UML version 2.4 (Janvier 2011).

Un diagramme de classes se compose de classes (*Class)* nommées, qui peuvent être reliées par des associations (*Association)* nommées, ou par des relations de généralisation (*superClass)*. Une classe peut avoir des attributs (*Property)*, décrits par leur type, leur nom, une multiplicité minimale et une multiplicité maximale, et des opérations (*Operation)* décrites par un nom, un type de retour et des paramètres (*Parameter)* décrits par un nom et un type. L'extrémité d'une association (*Property*) est

caractérisée par un nom (le rôle), une multiplicité minimale (*lower*) et une multiplicité maximale (*upper*), une valeur de composition, une valeur de navigabilité, un type et une association. Les multiplicités maximale et minimale sont par défaut égales à 1.

Classe, Association, Opération, Paramètre et Propriété sont des éléments nommés (*NamedElement*).

Une propriété peut être :

- Un attribut primitif : représente ce qu'on appelle généralement un attribut d'une classe (*ownedAttribute*).
- Un attribut de référence : représente ce qu'on appelle généralement une extrémité d'association (*memberEnd*) liée à une classe (*ownedAttribute*).

Une opération, un paramètre et une propriété sont des éléments typés (*TypedElement*).

Le type (*Type*) d'une opération représente son type de retour. Il s'agit d'un type primitif (int, char, etc.). Le type (*Type*) d'un attribut et d'un paramètre est un type primitif. Enfin, le type (*Type*) d'un attribut de référence est une classe.

Une propriété (extrémité d'association) peut être une composition *(isComposite)* ou une association (par défaut c'est une association).

Une association peut être navigable(*navigableOwnedEnd*) vers une propriété (extrémité d'association).

TypedElement, *Type*, *NamedElement* et *MultyplicityElement* sont des classes abstraites, cela veut dire qu'elles n'apparaissent pas dans un modèle UML.

Afin de satisfaire nos besoins en expression des règles de composition, nous avons étendu l'extrait du méta-modèle UML, en rajoutant la méta-classe *Model* afin de spécifier le modèle auquel appartient un élément.

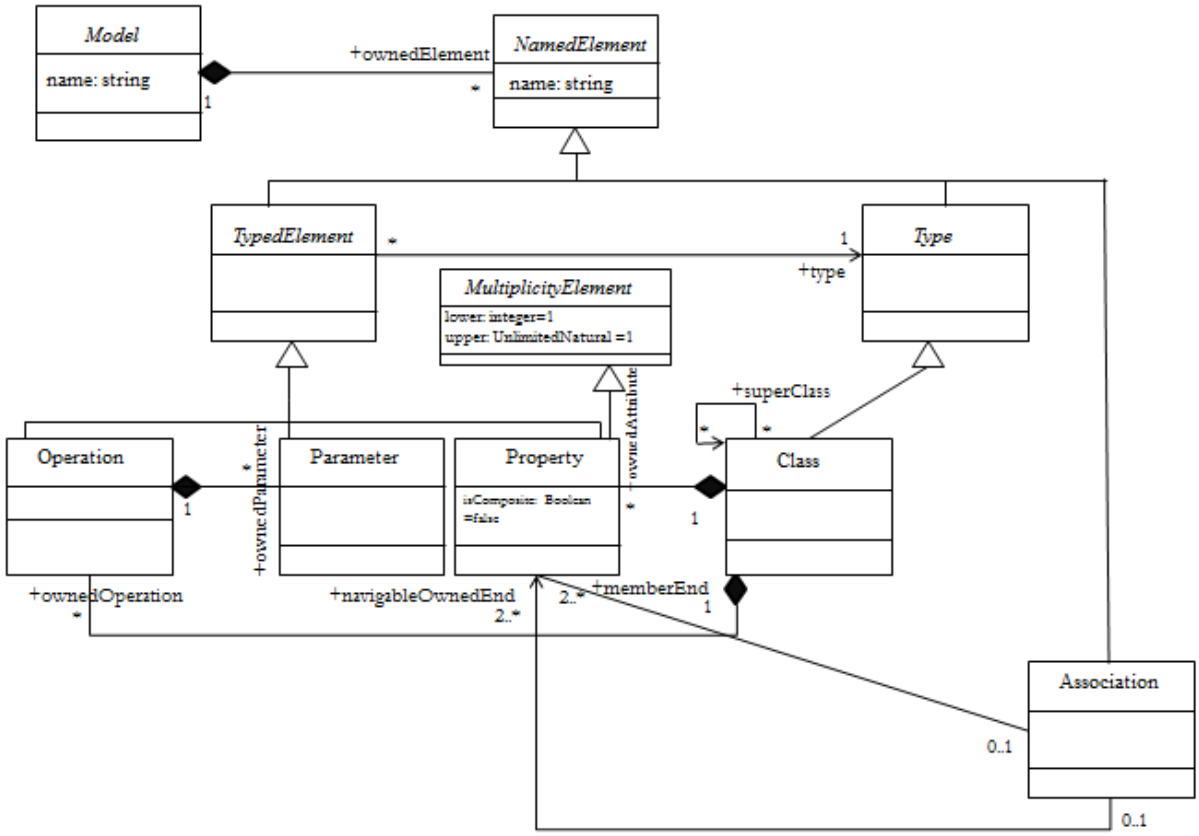


Figure 4-18 Extrait avec extension du méta-modèle UML [UML, 2011]

4.2.1.2. Méta-modèle de la logique des prédicats

Nous avons représenté un extrait du méta-modèle de la logique des prédicats dans la Figure 4-19 ci-dessous.

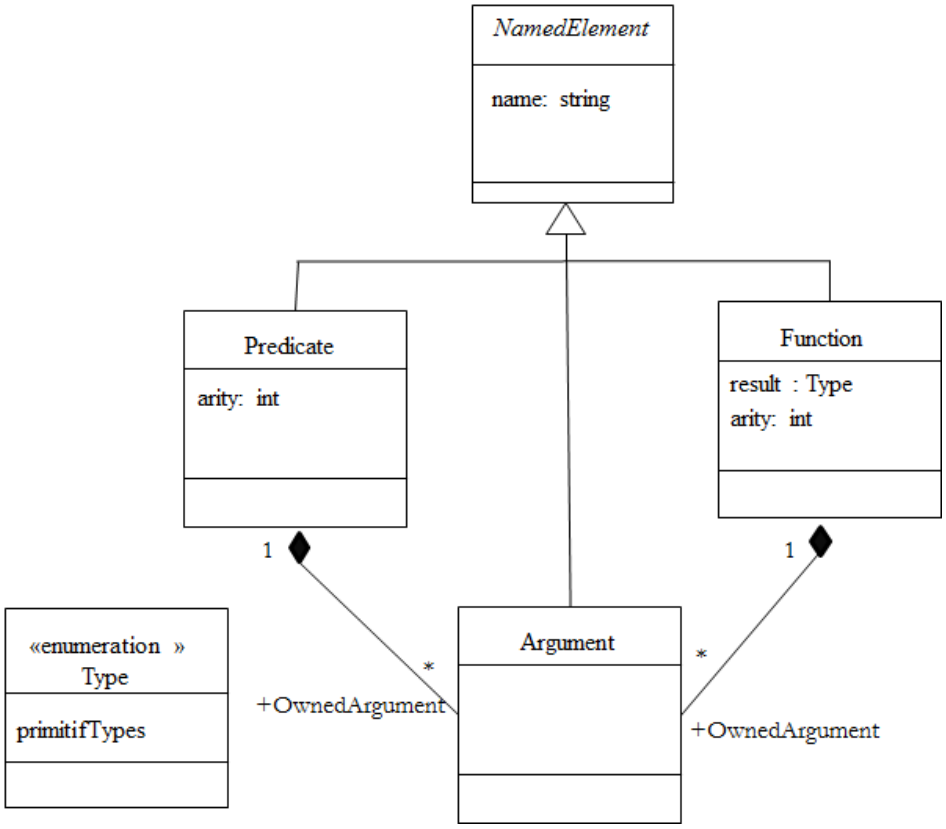


Figure 4-19 Méta-modèle de la logique des prédicats

- Un prédicat (*Predicate*), une fonction (*Function*) et un argument (*Argument*) sont des éléments nommés (*NamedElement*) caractérisés par leurs noms (*name*).
- Un prédicat et une fonction se composent (*ownedArguments*) d'arguments (*Argument*).
- Le nombre d'arguments détermine l'arité (*arity*) du prédicat et de la fonction.
- Une fonction est caractérisée aussi par son résultat (*result*) de type primitif.

4.2.1.3. Transformation de l'UML vers la logique des prédicats

Une transformation d'un modèle UML vers un modèle en logique des prédicats nécessite la définition des règles de transformation, qui déterminent les correspondances entre les éléments des méta-modèles UML et logique des prédicats. Nous présentons, dans le Tableau 4-1, chaque règle avec un exemple de transformation au niveau des modèles.

Règle 1	- La méta-classe *Class* sera transformée en un prédicat. - *« Class »* sera transformée en nom *name* du prédicat. - Les noms *name* des méta-classes *Class* et *Model* seront transformés dans cet ordre en arguments du prédicat *Class*. L'ordre des arguments est celui de la transformation (c'est-à-dire que le nom de *Class* est transformé en premier argument et le nom de *Model* est transformé en deuxième argument).	
Exemple	La classe « Personne » du modèle M2.	Class(Personne,M2)
Règle 2	- Le méta-rôle de la méta-association *OwnedAttribute* sera transformé en un prédicat. - *« OwnedAttribute »* sera transformé en nom *name* du prédicat. - Les noms *name* des méta-classes *Property, Class*, *Model*, *Type*, et les méta-attributs *lower* et *upper* seront transformés dans cet ordre en arguments du prédicat *OwnedAttribute*.	

	L'ordre des arguments est celui de la transformation.	
Exemple	L'attribut « nom » de la classe « Personne » appartenant au modèle M2, ayant pour type string et pour multiplicité minimale et maximale 1.	OwnedAttribute(nom,Personne,M2,string, 1,1)
Règle 3	- Le méta-rôle de la méta-association *OwnedOperation* sera transformé en un prédicat. - *« OwnedOperation »* sera transformé en nom *name* du prédicat. - Les noms *name* des méta-classes *Operation, Class*, *Model*, *Type* seront transformés dans cet ordre en arguments du prédicat *OwnedOperation*. L'ordre des arguments est celui de la transformation.	
Exemple	L'opération « débiter » de la classe « Compte » appartenant au modèle M1 ayant pour type de retour float	OwnedOperation(débiter, Compte,M1, float)
Règle 4	- Le méta-rôle de la méta-association *memberEnd* sera transformé en un prédicat. - *« memberEnd »* sera transformé en nom *name* du prédicat. - Les noms *name* des méta-classes *Association, Property,* le méta-attribut *isComposite,* et le rôle de la méta-association *navigableOwnedEnd* seront transformés dans cet ordre en arguments du prédicat *memberEnd*. L'ordre des arguments est celui de la transformation. Comme cité dans la règle 2 : - Le méta-rôle de la méta-association *OwnedAttribute* sera transformé en un prédicat. - *« OwnedAttribute »* sera transformé en nom *name* du prédicat. - Les noms *name* des méta-classes *Property, Class*, *Model*, *Type*, et les méta-attributs *lower* et *upper* seront transformés dans cet ordre en arguments du prédicat *OwnedAttribute*. L'ordre des arguments est celui de la transformation.	
Exemple	L'association « Joint » du modèle M1, relie « Compte » et « CarteCrédit », l'extrémité d'association « Cpt » du côté de « Compte » et l'extrémité d'association « Crt» du côté de « CarteCrédit»	OwnedAttribute(Cpt, CarteCrédit, M1, Compte,1,1) OwnedAttribute(Crt, Compte, M1, CarteCrédit,1,*) memberEnd(Joint,Cpt, false,false) memberEnd(Joint,Crt,false,false)
Règle 5	- Le méta-rôle de la méta-association *OwnedParameter* sera transformé en un prédicat. - *« OwnedParameter »* sera transformé en nom *name* du prédicat. - Les noms *name* des méta-classes *Parameter, Operation, Class*, *Model*, *Type* seront transformés dans cet ordre, en des arguments du prédicat *OwnedParameter*. L'ordre des arguments est celui de la transformation.	
Exemple	Le paramètre « solde » de l'opération « débiter » appartenant à la classe « CompteBancaire » du modèle M2 et qui a pour type float	OwnedParameter(solde, débiter, CompteBancaire, M2, float)
Règle 6	- Le méta-rôle de la méta-association *superClass* sera transformé en un prédicat. - *« superClass»* sera transformé en nom *name* du prédicat. - Les noms *name* des méta-classes *Class et Model* seront transformés en des arguments du prédicat *superClass*. L'ordre des arguments est celui de la transformation.	
Exemple	La classe « Personne » est une super-classe de la classe « Client » appartenant au modèle M2.	superClass(Personne,Client,M2)

Tableau 4-1 Règles de transformation de modèle UML vers un modèle en logique prédicats

4.2.2. Formalisation d'une ontologie OWL

L'objectif de cette partie est de représenter une ontologie OWL en logique des prédicats. Pour cela, nous définissons une transformation d'une ontologie OWL vers un modèle en logique des prédicats. Comme le cas présenté précédemment, la transformation s'effectue dans un contexte IDM, où les modèles OWL et logique sont conformes à leurs méta-modèles et ces derniers sont conformes au MOF. Nous présentons, dans ce qui suit, le méta-modèle OWL ainsi que les règles de transformation.

4.2.2.1. Méta-modèle OWL

Afin de définir le méta-modèle OWL (cf. Figure 4-20), nous utilisons la spécification de l'OMG, ODM (Ontology Definition Metamodel), version 1.0 (Mai 2009).

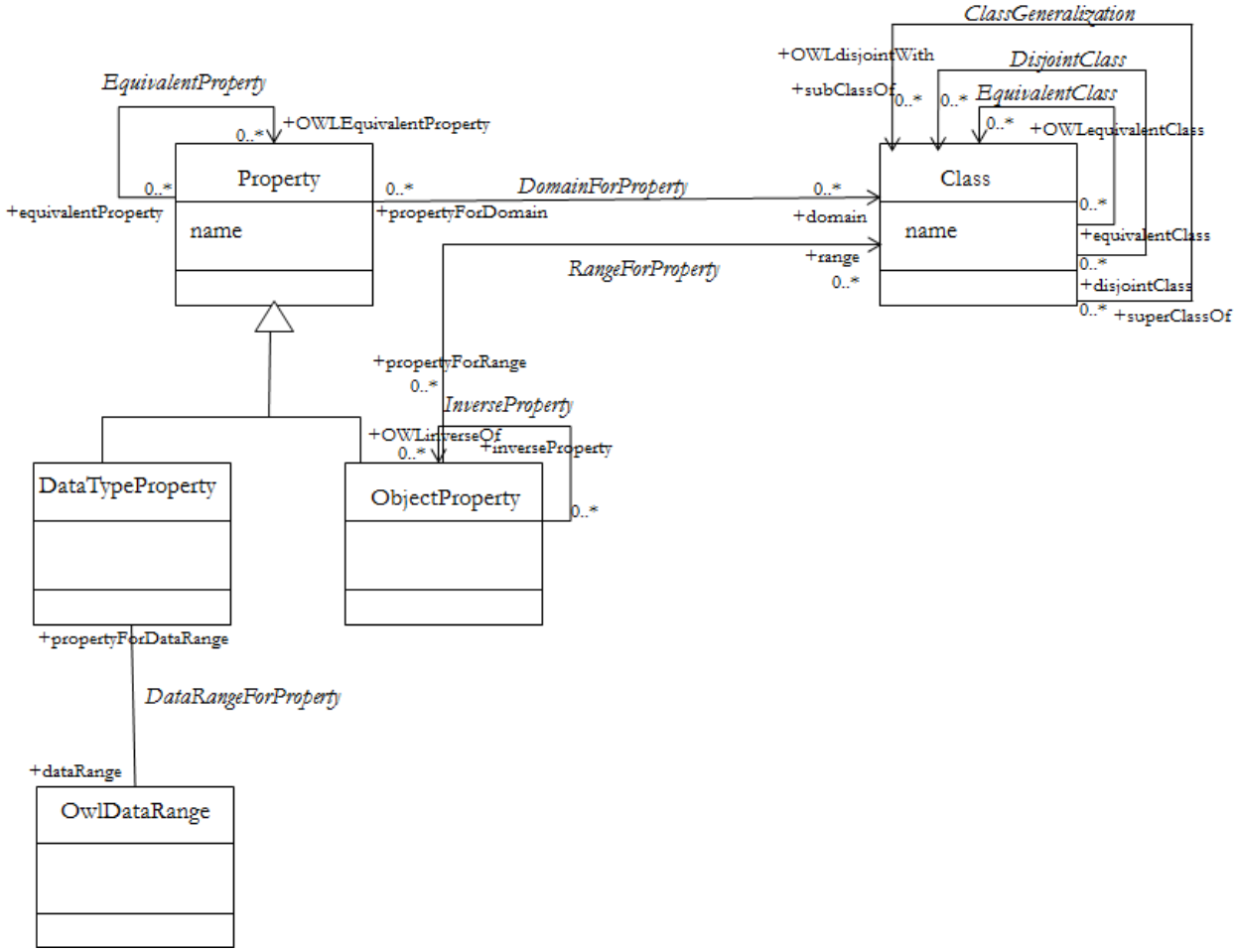


Figure 4-20 Extrait du méta-modèle OWL [ODM, 2009]

- Une classe fournit le mécanisme d'abstraction afin de grouper les ressources ayant des caractéristiques similaires. Elle est représentée par la méta-classe *Class*.
- Une classe peut avoir une ou plusieurs super et sous-classes. La relation *ClassGeneralization* relie une super-classe à une sous-classe.
- Une classe peut avoir une ou plusieurs classes équivalentes. Ceci est représenté par la relation *EquivalentClass* qui relie deux classes équivalentes.

- Une classe peut avoir une ou plusieurs classes disjointes. Ceci est représenté par la relation *DisjointClass* qui relie deux classes disjointes.
- Une propriété est décrite comme une relation entre une paire de ressources.
- Il existe deux types de propriétés : propriété de données et propriété d'objet.
- Les propriétés de données (*DataTypeProperty*) sont utilisées pour relier des individus à des valeurs de donnée. En d'autres termes, une propriété de données relie une classe et un type de donnée (int, char, etc.).
- Une propriété d'objet (*ObjectProperty*) relie un individu à d'autres individus.
- La source (*domain*) de la propriété (de donnée ou d'objet) est représentée par la relation *DomainForProperty* qui relie les deux méta-classes *Property* et *Class.*
- La cible *(range)* de la propriété de données est représentée par la relation *DataRangeForProperty* qui relie les deux méta-classes *DataTypeProperty* et *OwlDataRange* (le type).
- La cible *(range)* de la propriété d'objet est représentée par la relation *RangeForProperty* qui relie les deux méta-classes *ObjectProperty* et *Class.*
- Une propriété (de données ou d'objet) peut avoir une ou plusieurs propriétés équivalentes. Ceci est représenté par la relation *EquivalentProperty* qui relie deux propriétés équivalentes.
- Une propriété d'objet peut avoir une ou plusieurs propriétés d'objet inverses. Ceci est représenté par la relation *InverseProperty* qui relie deux propriétés inverses.
- Une propriété et une classe sont caractérisées par un nom, représenté par le méta-attribut *name*.

4.2.2.2. Transformation de l'ontologie OWL vers la logique des prédicats

Nous présentons, dans le Tableau 4-2, les règles de transformation d'une ontologie OWL vers un modèle en logique des prédicats. Notons que nous avons rajouté le terme « Onto » aux prédicats pour les différencier de ceux du modèle UML. (Exemple : le prédicat Class est présent dans les méta-modèles UML et OWL, donc *Class* de l'ontologie devient *OntoClass*).

Règle 1	- La méta-classe *Class* sera transformée en un prédicat. - *« OntoClass »* sera transformé en nom du prédicat *name.* - Le nom *name* de la méta-classe *Class* sera transformé en argument du prédicat Onto*Class.*	
Exemple	La classe « Personne »	OntoClass(Personne)
Règle 2	- Le rôle de la méta-association e*quivalentClass* sera transformé en un prédicat. - *« equivalentOntoClass »* sera transformé en nom du prédicat *name.* - Les noms *name* de la méta-classe *Class* seront transformés en deux arguments du prédicat e*quivalentOntoClass.*	
Exemple	Les deux classes « Employé » et « Personnel » sont équivalentes	equivalentOntoClass(Employé, Personnel)
Règle 3	- Le rôle de la méta-association *disjointClass* sera transformé en un prédicat. - *« disjointOntoClass »* sera transformé en nom du prédicat *name.* - Les noms *name* de la méta-classe *Class* seront transformés en deux arguments du prédicat *disjointOntoClass.*	
Exemple	Les deux classes « Marié» et « Célibataire » sont disjointes	disjointOntoClass(Marié, Célibataire)
Règle 4	- Le rôle de la méta-association *SuperClassOf* sera transformé en un prédicat. - *« OntoSuperClassOf »* sera transformé en nom du prédicat *name.* - Les noms *name* de la méta-classe *Class* seront transformés en deux arguments du prédicat *OntoSuperClassOf.*	

Exemple	La classe « Personne » est une super-classe de la classe « Client »	OntoSuperClassOf(Client, Personne)
Règle 5	- Le rôle de la méta-association *equivalentProperty* sera transformé en un prédicat. - *« equivalentOntoProperty »* sera transformé en nom du prédicat *name.* - Les noms *name* de la méta-classe *Property* seront transformés en deux arguments du prédicat *equivalentOntoProperty.*	
Exemple	Les propriétés d'objet « avoir » et « posséder » sont équivalentes.	equivalentOntoProperty(Avoir, Posséder)
Règle 6	- Le rôle de la méta-association *inverseProperty* sera transformé en un prédicat. - *« inverseOntoProperty »* sera transformé en nom du prédicat *name.* - Les noms *name* de la méta-classe *ObjectProperty* seront transformés en deux arguments du prédicat *inverseOntoProperty.*	
Exemple	Les propriétés d'objet « Gérer» et « EstGéréPar » sont inverses.	inverseOntoProperty(Gérer, EstGéréPar)
Règle 7	- Le rôle de la méta-association *domain* sera transformé en un prédicat. - *« domainOnto »* sera transformé en nom du prédicat *name.* - Les noms *name* des méta-classes *Property* et *Class* seront transformés dans cet ordre, en deux arguments du prédicat *domainOnto.* - Le rôle de la méta-association *range* sera transformé en un prédicat. - *« rangeOnto»* sera transformé en nom du prédicat *name.* - Les noms *name* des méta-classes *ObjectProperty* et *Class* seront transformés dans cet ordre, en deux arguments du prédicat *rangeOnto.* L'ordre des arguments est celui de la transformation.	
Exemple	La propriété d'objet « alimenter » a pour classe source « Versement » et classe cible «Compte»	domainOnto(alimenter,Versement) rangeOnto(alimenter,Compte)
Règle 8	- Le rôle de la méta-association *dataRange* sera transformé en un prédicat. - *« dataRangeOnto »* sera transformé en nom du prédicat *name.* - Les noms *name* des méta-classes *DataTypeProperty, Class* et *OwlDataRange* seront transformés dans cet ordre, en deux arguments du prédicat *dataRangeOnto.* L'ordre des arguments est celui de la transformation.	
Exemple	La propriété de donnée « avoirNom » de la classe « Personne » ayant pour type chaîne de caractères	dataRangeOnto(avoirNom , Personne,string)

Tableau 4-2 Règles de transformation d'une ontologie vers un modèle en logique des prédicats

Après avoir présenté le modèle UML et l'ontologie OWL, nous présentons, dans cette section, une formalisation des dictionnaires utilisés comme référence de notre système de composition.

4.2.3. Formalisation des dictionnaires

Nous représentons ci-dessous une formalisation des dictionnaires d'acronymes, d'abréviations et de synonymes.

- *DicAcronyms (elt1, elt2)* : signifie que *elt1* est un acronyme de *elt2 ou elt2* est un acronyme de *elt2.*
- *DicAbbreviation(elt1, elt2)* : signifie que *elt1* est une abréviation de *elt2* ou *elt2* est une abréviation de *elt1.*
- *DicSynonymy(elt1, elt2) :* signifie que *elt1* est un synonyme de *elt2.*

4.2.4. Base des faits

L'ensemble des prédicats, récupérés après les transformations, représentent les faits que nous utilisons pour exprimer les règles de composition. Afin de compléter l'expression des règles, nous définissons une fonction et un ensemble de faits provenant des langages de programmation.

Nous définissons une fonction qui fait l'objet de requête pour récupérer le nombre de paramètres d'une opération.

- *NbParameters(Op,C,M)* : retourne le nombre de paramètres de l'opération Op de la classe C du modèle M.

4.2.4.1. Faits provenant des langages de programmation

Nous avons défini un ensemble de faits issus de fonctions de base des langages de programmation, telles que l'égalité de deux chaînes de caractères, l'inclusion de deux chaînes de caractères, l'égalité et le sous-typage des types, l'égalité des nombres, etc.

- *String(elt) :* signifie que *elt* est une chaîne de caractères.
- *EqualString(ch1, ch2)*: signifie que les chaînes de caractères *ch1* et *ch2* sont égales.
- *InclusionString(ch1,ch2) :* signifie que la chaîne de caractères ch1 est incluse totalement (inclusion totale) dans la chaîne de caractères ch2.
- *EquivType(t1,t2)* : signifie que les types x et y sont équivalents. Deux types sont dit équivalents si et seulement si ils sont égaux ou l'un est un sous-type de l'autre (exemple : int est un sous-type de float).
- *UpperType (t1,t2) :* signifie que t2 est un sous-type de t1
- *LowerType(t1,t2) :* signifie que t1 est un sous-type de t2
- *EqualType(t1,t2) :* signifie que les types t1 et t2 sont égaux
- *EqualNb* (n1,n2) : signifie que le nombre n1 est égal au nombre n2.
- *Greater(n1,n2) :* signifie que le nombre n1 est plus grand que le nombre n2.

4.2.4.2. Faits utilisés dans les règles de fusion/translation

Nous avons défini également une base de faits, afin de pouvoir exprimer des règles de fusion/translation.

- *MainModel(M)* : signifie que M est le modèle principal.
- *BelongingToMainModel(elt,M) :* signifie que l'élément elt appartient au modèle principal M.
- *FusionResultElt(elt1,elt2,elt3) :* signifie que *elt3* est l'élément résultat de la fusion de *elt1* et *elt2*.
- *ResultElt(elt1,elt2) :* signifie que *elt2* est l'élément résultat de *elt1*. L'élément *elt1* a pu être translaté ou fusionné avec un autre élément.
- *IsolatedAtt(Att,C, M) :* signifie que l'attribut *Att* de la classe *C* du modèle *M* est un élément isolé.
- *IsolatedClass(C, M) :* signifie que la classe *C* du modèle *M* est un élément isolé.
- *IsolatedOp(Op,C, M) :* signifie que l'opération *Op* de la classe *C* du modèle *M* est un élément isolé.
- *IsolatedPm(Pm,Op,C, M) :* signifie que le paramètre Pm de l'opération *Op* de la classe *C* du modèle *M* est un élément isolé.

- *IsolatedAss(P,Ass,C,M) :* signifie que l'extrémité d'association *P* de l'association *Ass,* qui a pour type la classe *C* du modèle *M* est un élément isolé.
- *IsolatedElt(elt,M) :* signifie que l'élément *elt* du modèle *M* est un élément isolé.
- *RemoveAtt(Att,C,M) :* signifie la suppression de l'attribut Att de la classe C appartenant au modèle M
- *RemoveOp(Op,C,M) :* signifie la suppression de l'opération Op de la classe C appartenant au modèle M
- *RemoveAss(Ass,C1,C2,M) :* signifie la suppression de l'association Ass reliant les classes C1 et C2 appartenant au modèle M
- *RemoveRG(C1,C2,M) :* signifie la suppression de la relation de généralisation reliant la super-classe C2 et la sous-classe C1 appartenant au modèle M
- *Rename(ch1,ch2) :* signifie que la chaîne de caractères *ch1* est renommée en *ch2*.

Nous définissons, dans ce qui suit, les règles de composition en logique des prédicats des éléments de deux modèles M1 et M2.

4.2.5. Règles de composition

4.2.5.1. Règles de comparaison

- **Règle d'équivalence de deux attributs**

Description informelle

Deux attributs sont équivalents si et seulement si :

- ils sont équivalents syntaxiquement ou sémantiquement (synonymes, équivalents selon une ontologie ou disjoints),
- et ils ont un type équivalent et les mêmes multiplicités minimale et maximale. Nous sommes dans le cas, de ce que nous appelons équivalence en structure locale d'attributs. Deux types sont dit équivalents, si et seulement si, ils sont égaux ou l'un est inclus dans l'autre (exemple : int est inclus dans float, char est inclus dans string),
- et ils appartiennent à deux classes équivalentes syntaxiquement ou sémantiquement. Il s'agit de l'équivalence en structure globale d'attributs.

Exemple : l'attribut « code » de la classe « CarteCrédit » de M1 et l'attribut « code » de la classe « Carte » de M2 sont équivalents.

Représentation formelle

$$\mathbf{Equivalence\_Attributes}(\mathbf{Att_i}, \mathbf{Att_j}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2}) \Leftrightarrow [(\mathbf{Equivalence\_Semantic\_Attributes}(\mathbf{Att_i}, \mathbf{Att_j}) \vee \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_i}, \mathbf{Att_j})) \wedge \mathbf{OwnedAttribute}(\mathbf{Att_i}, \mathbf{C_i}, \mathbf{M1}, \mathbf{T1}, \mathbf{L1}, \mathbf{U1}) \wedge \mathbf{OwnedAttribute}(\mathbf{Att_j}, \mathbf{C_j}, \mathbf{M2}, \mathbf{T2}, \mathbf{L2}, \mathbf{U2}) \wedge \mathbf{EquivType}(\mathbf{T1}, \mathbf{T2}) \wedge \mathbf{EqualNb}(\mathbf{L1}, \mathbf{L2}) \wedge \mathbf{EqualNb}(\mathbf{U1}, \mathbf{U2})]$$

Deux attributs Att_i et Att_j, appartenant respectivement aux classes C_i et C_j et aux modèles M1 et M2, sont équivalents si et seulement si ils sont équivalents syntaxiquement ou sémantiquement, ont un

type équivalent (T1 et T2) et ont les mêmes multiplicités minimales (L1 et L2) et maximales (U1 et U2).
L'équivalence de la structure globale est assurée implicitement. En effet, nous ne comparons que les attributs et opérations appartenant à deux classes équivalentes syntaxiquement ou sémantiquement.

- **Règle d'équivalence sémantique de deux attributs**

Description informelle

Deux attributs sont équivalents sémantiquement si et seulement si ils sont synonymes, équivalents selon une ontologie ou disjoints.

Exemple : l'attribut « poste » de la classe « Personnel » de M1 et l'attribut « fonction » de la classe « Employé » de M2 sont équivalents sémantiquement.

Représentation formelle

$$\mathbf{Equivalence\_Semantic\_Attributes}(\mathbf{Att_i}, \mathbf{Att_j}) \Leftrightarrow$$

//Synonymie

$$[(\mathbf{DicSynonymy}(\mathbf{Att_i}, \mathbf{Att_j}) \vee (\exists \mathbf{e}\ \mathbf{String}(\mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_i}, \mathbf{e}) \wedge \mathbf{DicSynonymy}(\mathbf{e}, \mathbf{Att_j})) \vee (\exists \mathbf{c}\ \mathbf{String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_j}, \mathbf{c}) \wedge \mathbf{DicSynonymy}(\mathbf{c}, \mathbf{Att_i})) \vee (\exists \mathbf{e}\ \mathbf{String}(\mathbf{e}) \wedge \exists \mathbf{c}\ \mathbf{String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_j}, \mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_i}, \mathbf{c}) \wedge \mathbf{DicSynonymy}(\mathbf{e}, \mathbf{c})))$$

//Disjonction

$$\vee (\mathbf{Disjunction\_Attributes}(\mathbf{Att_i}, \mathbf{Att_j}) \vee (\exists \mathbf{e}\ \mathbf{String}(\mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_i}, \mathbf{e}) \wedge \mathbf{Disjunction\_Attributes}(\mathbf{e}, \mathbf{Att_j})) \vee (\exists \mathbf{c}\ \mathbf{String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_j}, \mathbf{c}) \wedge \mathbf{Disjunction\_Attributes}(\mathbf{c}, \mathbf{Att_i})) \vee (\exists \mathbf{e}\ \mathbf{String}(\mathbf{e}) \wedge \exists \mathbf{c}\ \mathbf{String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_j}, \mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_i}, \mathbf{c}) \wedge \mathbf{Disjunction\_Attributes}(\mathbf{e}, \mathbf{c})))$$

//Equivalence selon une ontologie

$$\vee (\mathbf{equivalentOntoProperty}(\mathbf{Att_i}, \mathbf{Att_j}) \vee (\exists \mathbf{e}\ \mathbf{String}(\mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_i}, \mathbf{e}) \wedge \mathbf{equivalentOntoProperty}(\mathbf{e}, \mathbf{Att_j})) \vee (\exists \mathbf{c}\ \mathbf{String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_j}, \mathbf{c}) \wedge \mathbf{equivalentOntoProperty}(\mathbf{c}, \mathbf{Att_i})) \vee (\exists \mathbf{e}\ \mathbf{String}(\mathbf{e}) \wedge \exists \mathbf{c}\ \mathbf{String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Att_j}, \mathbf{e}) \wedge \mathbf{Syntactic\_Equivalence}(\mathbf{Att_i}, \mathbf{c}) \wedge \mathbf{equivalentOntoProperty}(\mathbf{e}, \mathbf{c})))]$$

Deux attributs Att_i et Att_j sont équivalents sémantiquement si et seulement si Att_i (ou une chaîne de caractères équivalente syntaxiquement à Att_i) est synonyme, équivalente selon une ontologie ou disjoint à Att_j (ou une chaîne de caractères équivalente syntaxiquement à Att_j).

- **Règle de disjonction de deux attributs**

Description informelle

Deux attributs sont disjoints si et seulement si, dans l'ontologie de domaine, deux classes de même nom que les attributs, sont disjointes.

Exemple : l'attribut « célibataire » de la classe « Personnel » de M1 et l'attribut hérité « marié » de la classe « Employé » de M2 sont disjoints.

Représentation formelle

$$\mathbf{Disjunction\_Attrbutes}(\mathbf{Att_i}, \mathbf{Att_j})$$
$$\Leftrightarrow [\exists \mathbf{C_i}\ \mathbf{OntoClass}(\mathbf{C_i}) \wedge \exists \mathbf{C_j}\ \mathbf{OntoClass}(\mathbf{C_j}) \wedge \mathbf{disjointOntoClass}(\mathbf{C_i}, \mathbf{C_j})$$
$$\wedge\ \mathbf{EqualString}(\mathbf{C_i}, \mathbf{Att_i}) \wedge \mathbf{EqualString}(\mathbf{C_j}, \mathbf{Att_j})]$$

Deux attributs Att_i et Att_j sont disjoints si et seulement si il existe dans l'ontologie deux classes C_i et C_j (avec les mêmes noms que les attributs) qui sont disjointes.

- **Equivalence syntaxique de deux éléments**

Description informelle

Deux éléments sont syntaxiquement équivalents si et seulement si ils sont identiques, ou si il existe une relation d'inclusion totale, d'acronymie ou d'abréviation entre eux.

Représentation formelle

$$\mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{elt_i}, \mathbf{elt_j})$$
$$\Leftrightarrow [\mathbf{Syntactic\_Identity}\ (\mathbf{elt_i}, \mathbf{elt_j})\ \vee\ \mathbf{InclusionString}\ (\mathbf{elt_i}, \mathbf{elt_j})$$
$$\vee\ \mathbf{InclusionString}\ (\mathbf{elt_j}, \mathbf{elt_i}) \vee \mathbf{DicAcronyms}(\mathbf{elt_i}, \mathbf{elt_j})\ \vee\ \mathbf{DicAbbreviation}(\mathbf{elt_i}, \mathbf{elt_j})]$$

Deux éléments elt_i et elt_j sont syntaxiquement équivalents si et seulement si l'une de ces conditions est satisfaite :

- Identité syntaxique.
- Inclusion syntaxique: un élément est inclus syntaxiquement (inclusion totale) dans l'autre.
- Acronymie : un élément elt_i est l'acronyme de l'élément elt_j si et seulement si il existe une relation qui les relie dans le dictionnaire d'acronymes.
- Abréviation : un élément elt_i est l'abréviation de l'élément elt_j s'il existe une relation qui les relie dans le dictionnaire d'abréviations.

- **Identité syntaxique de deux éléments**

Description informelle

Deux éléments sont identiques syntaxiquement si et seulement si ils ont le même nom.

Représentation formelle

$$\mathbf{Syntactic\_Identity}(\mathbf{elt_i}, \mathbf{elt_j}) \Leftrightarrow \mathbf{EqualString}\ (\mathbf{elt_i}, \mathbf{elt_j})$$

Deux éléments elt_i et elt_j sont identiques syntaxiquement si et seulement si ils ont le même nom.

- **Règle d'équivalence de deux opérations**

Description informelle

Deux opérations sont équivalentes si et seulement si:

- elles sont équivalentes syntaxiquement ou sémantiquement,
- elles ont la même structure locale (appelée aussi signature d'opération),
- et elles appartiennent à deux classes équivalentes syntaxiquement ou sémantiquement (équivalence structure globale).

Exemple : l'opération « consulterSolde » de la classe « Compte » de M1 et l'opération « consulter » de la classe « CompteBancaire» de M2 sont équivalentes.

Représentation formelle

$$\begin{aligned}&\mathbf{Equivalence\_Operations}(\mathbf{Op_i}, \mathbf{Op_j}, \mathbf{C_i}, \mathbf{C_i}, \mathbf{M1}, \mathbf{M2})\\ &\quad \Leftrightarrow [(\mathbf{Equivalence\_Semantic\_Operations}(\mathbf{Op_i}, \mathbf{Op_j})\\ &\quad \vee\ \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{Op_i}, \mathbf{Op_j}))\\ &\quad \wedge \mathbf{EquivSignature\_Operations}(\mathbf{Op_i}, \mathbf{Op_j}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2})]\end{aligned}$$

Deux opérations Op_i et Op_j appartenant respectivement aux classes C_i et C_j et aux modèles M1 et M2 sont équivalentes si et seulement si elles sont équivalentes syntaxiquement ou sémantiquement, et si elles ont la même signature.

- **Règle d'équivalence de signature de deux opérations**

Description informelle

Deux opérations ont la même signature si et seulement si leurs types de retour sont équivalents, si elles ont le même nombre de paramètres et si leurs paramètres sont deux à deux équivalents.

Exemple : l'opération « créditer» de la classe « Compte » de M1 et l'opération « créditer» de la classe « CompteBancaire» de M2 ont la même signature.

Représentation formelle

$$\mathbf{EquivSignature\_Operations}(\mathbf{Op_i}, \mathbf{Op_j}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2}) \Leftrightarrow$$

//type

$$[[\mathbf{OwnedOperation}(\mathbf{Op_i}, \mathbf{C_i}, \mathbf{M1}, \mathbf{T1}) \wedge \mathbf{OwnedOperation}(\mathbf{Op_j}, \mathbf{C_j}, \mathbf{M2}, \mathbf{T2}) \wedge \mathbf{EquivType}(\mathbf{T1}, \mathbf{T2})] \wedge$$

//Nombre de paramètres

$$\mathbf{EqualNb}(\mathbf{NbParameters}(\mathbf{Op_i}, \mathbf{C_i}, \mathbf{M1}), \mathbf{NbParameters}(\mathbf{Op_j}, \mathbf{C_j}, \mathbf{M2})) \wedge$$

//Les mêmes paramètres

$$\begin{aligned}&[[\forall\ \mathbf{Pm_k}\ \mathbf{OwnedParameter}(\mathbf{Pm_k}, \mathbf{Op_i}, \mathbf{C_i}, \mathbf{M1}, -) \Rightarrow (\exists\ \mathbf{Pm_l} \mathbf{OwnedParameter}(\mathbf{Op_j}, \mathbf{Pm_l}, \mathbf{C_j}, \mathbf{M2}, -)\\ &\quad \wedge \mathbf{Equivalence\_Parameters}(\mathbf{Pm_k}, \mathbf{Pm_l}, \mathbf{Op_i}, \mathbf{Op_j}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2}))]\\ &\quad \wedge [\forall \mathbf{Pm_l} \mathbf{OwnedParameter}(\mathbf{Op_j}, \mathbf{Pm_l}, \mathbf{C_j}, \mathbf{M2}, -)\\ &\quad \Rightarrow (\exists\ \mathbf{Pm_k}\ \mathbf{OwnedParameter}(\mathbf{Pm_k}, \mathbf{Op_i}, \mathbf{C_i}, \mathbf{M1}, -)\\ &\quad \wedge \mathbf{Equivalence\_Parameters}(\mathbf{Pm_k}, \mathbf{Pm_l}, \mathbf{Op_i}, \mathbf{Op_j}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2}))]]]\end{aligned}$$

Deux opérations Op_i et Op_j appartenant respectivement aux classes C_i et C_j et aux modèles M1 et M2 ont la même signature si et seulement si leurs types de retour sont équivalents, elles ont le même nombre de paramètres et leurs paramètres respectifs Pm_k et Pm_l sont deux à deux équivalents.

- **Règle d'équivalence sémantique de deux opérations**

Description informelle

Deux opérations sont équivalentes sémantiquement si et seulement elles sont synonymes selon un dictionnaire de synonymes.

Représentation formelle

$$\begin{aligned}&\textbf{Equivalence\_Semantic\_Operations}(\textbf{Op}_\textbf{i}, \textbf{Op}_\textbf{j})\\ &\quad\Leftrightarrow [\textbf{DicSynonymy}(\textbf{Op}_\textbf{i}, \textbf{Op}_\textbf{j}) \vee (\exists \textbf{e}\ \textbf{String}(\textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Op}_\textbf{i}, \textbf{e})\\ &\quad\wedge \textbf{DicSynonymy}(\textbf{e}, \textbf{Op}_\textbf{j})) \vee (\exists \textbf{c}\ \textbf{String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Op}_\textbf{j}, \textbf{c})\\ &\quad\wedge \textbf{DicSynonymy}(\textbf{c}, \textbf{Op}_\textbf{i})) \vee (\exists \textbf{e}\ \textbf{String}(\textbf{e}) \wedge \exists \textbf{c}\ \textbf{String}(\textbf{c})\\ &\quad\wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Op}_\textbf{j}, \textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Op}_\textbf{i}, \textbf{c})\\ &\quad\wedge \textbf{DicSynonymy}(\textbf{e}, \textbf{c}))]\end{aligned}$$

Deux opérations Op_i et Op_j sont équivalentes sémantiquement si et seulement si Op_i (ou une chaîne de caractères équivalente syntaxiquement à Op_i) est synonyme à Op_j (ou une chaîne de caractères équivalente syntaxiquement à Op_j).

- **Règle d'équivalence de deux paramètres**

Description informelle

Deux paramètres sont équivalents si et seulement si :

- ils sont équivalents syntaxiquement ou sémantiquement,
- ils ont un type équivalent (équivalence structure locale),
- et ils appartiennent à deux opérations équivalentes (équivalence structure globale).

Exemple : Les paramètres « solde » de l'opération « débiter » de la classe « Compte » de M1 et le paramètre l'opération « solde » de l'opération « débiter » de la classe « CompteBancaire » de M2 sont équivalents.

Représentation formelle

$$\begin{aligned}&\textbf{Equivalence\_Parameters}(\textbf{Pm}_\textbf{i}, \textbf{Pm}_\textbf{j}, \textbf{OP}_\textbf{i}, \textbf{OP}_\textbf{j}, \textbf{C}_\textbf{i}, \textbf{C}_\textbf{j}, \textbf{M1}, \textbf{M2})\\ &\quad\Leftrightarrow [(\textbf{Equivalence\_Semantic\_Parameters}(\textbf{Pm}_\textbf{i}, \textbf{Pm}_\textbf{j})\\ &\quad\vee\ \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Pm}_\textbf{i}, \textbf{Pm}_\textbf{j})) \wedge (\textbf{OwnedParameter}(\textbf{Pm}_\textbf{i}, \textbf{Op}_\textbf{i}, \textbf{C}_\textbf{i}, \textbf{M1}, \textbf{T1})\\ &\quad\wedge \textbf{OwnedParameter}(\textbf{Pm}_\textbf{j}, \textbf{Op}_\textbf{j}, \textbf{C}_\textbf{j}, \textbf{M2}, \textbf{T2}) \wedge \textbf{EquivType}(\textbf{T1}, \textbf{T2}))]\end{aligned}$$

Deux paramètres Pm_i et Pm_j, appartenant respectivement aux deux opérations Op_i et Op_j, aux classes C_i et C_j et aux modèles M1 et M2, sont équivalents si et seulement si ils sont équivalents syntaxiquement ou sémantiquement et ont un type équivalent.

De la même manière que les attributs et les opérations, l'équivalence de la structure globale n'est pas exprimée. Nous comparons uniquement les paramètres appartenant à deux opérations équivalentes syntaxiquement ou sémantiquement. La règle de l'équivalence des paramètres est appliquée uniquement lors de son appel par la fonction d'équivalence d'opérations.

- **Règle d'équivalence sémantique de deux paramètres**

Description informelle

Deux paramètres sont équivalents sémantiquement si et seulement si ils sont synonymes.

Représentation formelle

$$\textbf{Equivalence\_Semantic\_Parameters}(\textbf{Pm}_i, \textbf{Pm}_j) \Leftrightarrow [\textbf{DicSynonymy}(\textbf{Pm}_i, \textbf{Pm}_j) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Pm}_i, \textbf{e}) \wedge \textbf{DicSynonymy}(\textbf{e}, \textbf{Pm}_j)) \vee (\exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Pm}_j, \textbf{c}) \wedge \textbf{DicSynonymy}(\textbf{c}, \textbf{Pm}_i)) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Pm}_j, \textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Pm}_i, \textbf{c}) \wedge \textbf{DicSynonymy}(\textbf{e}, \textbf{c}))]$$

Deux paramètres Pm_i et Pm_j sont équivalents sémantiquement si et seulement si Pm_i (ou une chaîne de caractères équivalente syntaxiquement à Pm_i) est synonyme à Pm_j (ou une chaîne de caractères équivalente syntaxiquement à Pm_j).

- **Règle d'équivalence de deux associations**

Description informelle

Hypothèse : Nous supposons que les associations sont nommées pour pouvoir comparer leurs noms.

Deux associations sont équivalentes si et seulement si elles :

- sont équivalentes syntaxiquement ou sémantiquement (synonymes, inverses ou équivalentes en se référant à une ontologie).
- relient des classes équivalentes syntaxiquement ou sémantiquement deux à deux (équivalence en structure globale)

Similairement, l'équivalence de la structure globale n'est pas exprimée car nous ne comparons que les associations reliant des classes équivalentes syntaxiquement ou sémantiquement.

Exemple : l'association « Joint » reliant les classes « Compte » et « CarteCrédit » de M1 et l'association « Joint » reliant les classes « CompteBancaire » et « Carte » de M2 sont équivalentes.

Représentation formelle

$$\textbf{Equivalence\_Associations}(\textbf{Ass}_i, \textbf{Ass}_j) \Leftrightarrow [\textbf{Equivalence\_Semantic\_Associations}(\textbf{Ass}_i, \textbf{Ass}_j) \vee \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_i, \textbf{Ass}_j)]$$

Deux associations Ass_i et Ass_j sont équivalentes si et seulement si elles sont équivalentes syntaxiquement ou sémantiquement.

- **Règle d'équivalence sémantique de deux associations**

Description informelle

Deux associations sont équivalentes sémantiquement si et seulement elles sont synonymes, inverses ou équivalentes selon une ontologie.

Exemple : l'association « Gérer » reliant les classes « Personnel » et « Client » de M1 et l'association « EstGérérPar» reliant les classes « Employé » et « Client » de M2 sont équivalentes sémantiquement.

Représentation formelle

$$\textbf{Equivalence\_Semantic\_Associations}(\textbf{Ass}_i, \textbf{Ass}_j) \Leftrightarrow$$
$$\textbf{//Synonymie}$$
$$[(\textbf{DicSynonymy}(\textbf{Ass}_i, \textbf{Ass}_j) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_i, \textbf{e}) \wedge \textbf{DicSynonymy}(\textbf{e}, \textbf{Ass}_j)) \vee (\exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_j, \textbf{c}) \wedge \textbf{DicSynonymy}(\textbf{c}, \textbf{Ass}_i)) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_j, \textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_i, \textbf{c}) \wedge \textbf{DicSynonymy}(\textbf{e}, \textbf{c}))$$
$$\textbf{//inverse}$$
$$\vee (\textbf{inverseOntoProperty}(\textbf{Ass}_i, \textbf{Ass}_j) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_i, \textbf{e}) \wedge \textbf{inverseOntoProperty}(\textbf{e}, \textbf{Ass}_j)) \vee (\exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_j, \textbf{c}) \wedge \textbf{inverseOntoProperty}(\textbf{c}, \textbf{Ass}_i)) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_j, \textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_i, \textbf{c}) \wedge \textbf{inverseOntoProperty}(\textbf{e}, \textbf{c})))$$
$$\textbf{//Equivalence}$$
$$\vee (\textbf{equivalentOntoProperty}(\textbf{Ass}_i, \textbf{Ass}_j) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_i, \textbf{e}) \wedge \textbf{equivalentOntoProperty}(\textbf{e}, \textbf{Ass}_j)) \vee (\exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_j, \textbf{c}) \wedge \textbf{equivalentOntoProperty}(\textbf{c}, \textbf{Ass}_i)) \vee (\exists \textbf{e String}(\textbf{e}) \wedge \exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\textbf{Ass}_j, \textbf{e}) \wedge \textbf{Syntactic\_Equivalence}(\textbf{Ass}_i, \textbf{c}) \wedge \textbf{equivalentOntoProperty}(\textbf{e}, \textbf{c})))]$$

Deux associations Ass_i et Ass_j sont équivalentes sémantiquement si et seulement si l'une de ces trois conditions est satisfaite :

- Ass_i (ou une chaîne de caractères équivalente syntaxiquement à Ass_i) est synonyme (selon un dictionnaire) à Ass_j (ou une chaîne de caractères équivalente syntaxiquement à Ass_j)
- Il existe deux propriétés d'objet dans l'ontologie, avec les mêmes noms que Ass_i et Ass_j (ou des chaînes de caractères équivalentes syntaxiquement à Ass_i et Ass_j) qui sont inverses.
- Il existe deux propriétés d'objet dans l'ontologie, avec les mêmes noms que Ass_i et Ass_j (ou des chaînes de caractères équivalentes syntaxiquement à Ass_i et Ass_j) qui sont équivalentes.

- **Règle d'équivalence de deux relations de généralisation**

Description informelle

Deux relations de généralisation sont équivalentes si et seulement si elles relient des classes équivalentes syntaxiquement ou sémantiquement deux à deux (équivalence en structure globale).

Représentation formelle

$$\begin{aligned}&\textbf{Equivalence\_Gen}(C_i, C_j, C_m, C_n, M1, M2)\\&\quad\Leftrightarrow [\textbf{superClass}(C_m, C_i, M1) \wedge \textbf{superClass}(C_n, C_j, M2)\\&\quad\wedge (\textbf{Equivalence\_Semantic\_Classes}(C_i, C_j) \vee \textbf{Equivalence\_Syntactic\_Elts}(C_i, C_j))\\&\quad\wedge (\textbf{Equivalence\_Semantic\_Classes}(C_m, C_n) \vee \textbf{Equivalence\_Syntactic\_Elts}(C_m, C_n))]\end{aligned}$$

Deux relations de généralisation reliant respectivement la super-classe C_m avec sa sous-classe C_i, et la super-classe C_n avec sa sous-classe C_j sont équivalentes si et seulement si C_i et C_j sont équivalentes syntaxiquement ou sémantiquement et C_m et C_n sont équivalentes syntaxiquement ou sémantiquement.

- **Règles de subsomption de classes**

Description informelle

Deux classes sont en relation de subsomption s'il existe dans l'ontologie de domaine deux classes portant leur nom, reliées par une relation de généralisation.

Exemple : la classe « Client » de M1 et la classe « Personne » de M2 sont en relation de subsomption.

Représentation formelle

$$\begin{aligned}&\textbf{Subsumption\_Classes}(C_i, C_j, M1, M2)\\&\quad\Leftrightarrow [\textbf{Class}(C_i, M1) \wedge \textbf{Class}(C_j, M2) \wedge ((\textbf{OntoSuperClassOf}(C_i, C_j) \vee (\exists \textbf{e String}(\textbf{e})\\&\quad\wedge \textbf{Equivalence\_Syntactic\_Elts}(C_i, \textbf{e}) \wedge \textbf{OntoSuperClassOf}(\textbf{e}, C_j)) \vee (\exists \textbf{c String}(\textbf{c})\\&\quad\wedge \textbf{Equivalence\_Syntactic\_Elts}(C_j, \textbf{c}) \wedge \textbf{OntoSuperClassOf}(C_i, \textbf{c})) \vee (\exists \textbf{e String}(\textbf{e})\\&\quad\wedge \exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(C_j, \textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(C_i, \textbf{c})\\&\quad\wedge \textbf{OntoSuperClassOf}(\textbf{c}, \textbf{e}))) \vee (\textbf{OntoSuperClassOf}(C_j, C_i) \vee (\exists \textbf{e String}(\textbf{e})\\&\quad\wedge \textbf{Equivalence\_Syntactic\_Elts}(C_i, \textbf{e}) \wedge \textbf{OntoSuperClassOf}(C_j, \textbf{e})) \vee (\exists \textbf{c String}(\textbf{c})\\&\quad\wedge \textbf{Equivalence\_Syntactic\_Elts}(C_j, \textbf{c}) \wedge \textbf{OntoSuperClassOf}(\textbf{c}, C_i)) \vee (\exists \textbf{e String}(\textbf{e})\\&\quad\wedge \exists \textbf{c String}(\textbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(C_j, \textbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(C_i, \textbf{c})\\&\quad\wedge \textbf{OntoSuperClassOf}(\textbf{e}, \textbf{c}))))]\end{aligned}$$

Deux classes C_i et C_j appartenant respectivement à M1 et M2 sont en relation de subsomption si et seulement si l'une de ces conditions est satisfaite :

- Il existe dans l'ontologie de domaine une super-classe C_j (ou une chaîne de caractères équivalente syntaxiquement à C_j) et sa sous-classe C_i (ou une chaîne de caractères équivalente syntaxiquement à C_i)
- Il existe dans l'ontologie de domaine une super-classe C_i (ou une chaîne de caractères équivalente syntaxiquement à C_i) et sa sous-classe C_j (ou une chaîne de caractères équivalente syntaxiquement à C_j)

- **Règle d'équivalence de deux classes**

Description informelle

Deux classes sont équivalentes si et seulement si :

- elles sont équivalentes syntaxiquement ou sémantiquement (synonymes ou équivalentes en se référant à une ontologie),
- elles ne sont pas en relation de subsomption
- et elles vérifient une des options suivantes (choisie par l'utilisateur lors du paramétrage du système):
  - Les classes sont équivalentes en structure locale partielle
  - Les classes sont équivalentes en structure locale totale
  - Les classes sont équivalentes en structure globale partielle
  - Les classes sont équivalentes en structure globale totale
  - Les classes sont équivalentes en structure locale partielle et globale partielle
  - Les classes sont équivalentes en structure locale partielle et globale totale
  - Les classes sont équivalentes en structure locale totale et globale partielle
  - Les classes sont équivalentes en structure locale totale et globale totale.

Etant donné qu'une classe peut avoir des liens de généralisation /spécialisation avec d'autres classes, l'ensemble de tous ses attributs (qui forme sa structure locale), ne se limite pas à ceux qui sont spécifiques à la classe, mais inclut aussi ceux qui sont hérités. Il est donc nécessaire, avant de chercher la correspondance entre les attributs de deux classes, de chercher l'ensemble complet de leurs attributs. Ceci s'applique aussi aux opérations, aux relations de généralisations et aux associations.

Représentation formelle

$$\mathbf{Equivalence\_Classes}(C_i, C_j, M1, M2, opt) \Leftrightarrow [\mathbf{Class}(C_i, M1) \wedge \mathbf{Class}(C_j, M2) \wedge (\mathbf{Equivalence\_Semantic\_Classes}(C_i, C_j) \vee \mathbf{Equivalence\_Syntactic\_Elts}(C_i, C_j)) \wedge (\neg \mathbf{Subsumption\_Classes}(C_i, C_j, M1, M2)) \wedge \mathbf{ValidationOption}(C_i, C_j, M1, M2, opt)]$$

$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option1) \Leftrightarrow \mathbf{LPS\_EquivClasses}(C_i, C_j, M1, M2)$$
$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option2) \Leftrightarrow \mathbf{LTS\_EquivClasses}(C_i, C_j, M1, M2)$$
$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option3) \Leftrightarrow \mathbf{GTS\_EquivClasses}(C_i, C_j, M1, M2)$$
$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option4) \Leftrightarrow \mathbf{GPS\_EquivClasses}(C_i, C_j, M1, M2)$$
$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option5) \Leftrightarrow (\mathbf{LPS\_EquivClasses}(C_i, C_j, M1, M2) \wedge \mathbf{GPS\_EquivClasses}(C_i, C_j, M1, M2))$$
$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option6) \Leftrightarrow (\mathbf{LPS\_EquivClasses}(C_i, C_j, M1, M2) \wedge \mathbf{GTS\_EquivClasses}(C_i, C_j, M1, M2))$$
$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option7) \Leftrightarrow (\mathbf{LTS\_EquivClasses}(C_i, C_j, M1, M2) \wedge \mathbf{GPS\_EquivClasses}(C_i, C_j, M1, M2))$$
$$\mathbf{ValidationOption}(C_i, C_j, M1, M2, option8) \Leftrightarrow (\mathbf{LTS\_EquivClasses}(C_i, C_j, M1, M2) \wedge \mathbf{GTS\_EquivClasses}(C_i, C_j, M1, M2))$$

Deux classes C_i et C_j appartenant respectivement aux modèles M1 et M2 sont équivalentes si et seulement si elles sont équivalentes syntaxiquement ou sémantiquement, ne sont pas en relation de subsomption et vérifient une des options suivantes (opt) choisie par l'utilisateur :

- Les classes C_i et C_j sont équivalentes en structure locale partielle
- Les classes C_i et C_j sont équivalentes en structure locale totale
- Les classes C_i et C_j sont équivalentes en structure globale partielle

- Les classes C_i et C_j sont équivalentes en structure globale totale
- Les classes C_i et C_j sont équivalentes en structure locale partielle et globale partielle
- Les classes C_i et C_j sont équivalentes en structure locale partielle et globale totale
- Les classes C_i et C_j sont équivalentes en structure locale totale et globale partielle
- Les classes C_i et C_j sont équivalentes en structure locale totale et globale totale

- **Règle d'équivalence sémantique de deux classes**

Description informelle

Deux classes sont équivalentes sémantiquement si et seulement si elles sont synonymes ou équivalentes selon une ontologie.

Exemple : les classes « Personnel » de M1 « Employé » de M2 sont équivalentes selon une ontologie et donc équivalentes sémantiquement.

Représentation formelle

$$\textbf{Equivalence\_Semantic\_Classes}(\mathbf{C_i}, \mathbf{C_j}) \Leftrightarrow$$

//Synonymie

$$[(\textbf{DicSynonymy}(\mathbf{C_i}, \mathbf{C_j}) \vee (\exists \mathbf{e}\ \textbf{String}(\mathbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{e}) \wedge \textbf{DicSynonymy}(\mathbf{e}, \mathbf{C_j}))$$
$$\vee (\exists \mathbf{c}\ \textbf{String}(\mathbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{c}) \wedge \textbf{DicSynonymy}(\mathbf{c}, \mathbf{C_i}))$$
$$\vee (\exists \mathbf{e}\ \textbf{String}(\mathbf{e}) \wedge \exists \mathbf{c}\ \textbf{String}(\mathbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{e})$$
$$\wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{c}) \wedge \textbf{DicSynonymy}(\mathbf{e}, \mathbf{c}))]$$

//Equivalence

$$\vee (\text{equivalentOntoClass}(\mathbf{C_i}, \mathbf{C_j}) \vee (\exists \mathbf{e}\ \textbf{String}(\mathbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{e})$$
$$\wedge \text{equivalentOntoClass}(\mathbf{e}, \mathbf{C_j})) \vee (\exists \mathbf{c}\ \textbf{String}(\mathbf{c}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{c})$$
$$\wedge \text{equivalentOntoClass}(\mathbf{c}, \mathbf{C_i})) \vee (\exists \mathbf{e}\ \textbf{String}(\mathbf{e}) \wedge \exists \mathbf{c}\ \textbf{String}(\mathbf{c})$$
$$\wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{e}) \wedge \textbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{c})$$
$$\wedge \text{equivalentOntoClass}(\mathbf{e}, \mathbf{c}))))]$$

Deux classes C_i et C_j sont équivalentes sémantiquement si et seulement si une de ces conditions est satisfaite :

- C_i (ou une chaîne de caractères équivalente syntaxiquement à C_i) est synonyme de C_j (ou une chaîne de caractères équivalente syntaxiquement à C_j) selon un dictionnaire de synonymes.
- Il existe deux classes dans l'ontologie, avec les mêmes noms que C_i et C_j (ou des chaînes de caractères équivalentes syntaxiquement à C_i et C_j) qui sont équivalentes.

- **Règle d'équivalence en structure globale partielle (GPS : Global Partiel Structural) de deux classes**

Description informelle

Deux classes sont équivalentes en structure globale partielle si et seulement si un élément (association, relation de généralisation, classe) lié à une des classes, est équivalent à un élément lié à l'autre classe.

Exemple : Les classes « Personnel » de M1 et « Employé » de M2 sont équivalentes en structure globale partielle, car l'association « Gérer » de M1 qui relie « Personnel » à « Client » est équivalente à l'association « EstGéréPar » de M2 qui relie « Employé » à « Client », et les classes « Client » de M1 et « Client » de M2 sont équivalentes sémantiquement.

Représentation formelle

$$\begin{aligned}&\mathbf{GPS\_EquivClasses}(C_i, C_j, M1, M2)\\ &\Leftrightarrow [[[(\exists\, C_m, Ass_k\ (\mathbf{Class}(C_m, M1) \wedge \mathbf{OwnedAttribute}(P_i, C_m, M1, C_i, -, -)\\ &\wedge \mathbf{OwnedAttribute}(P_m, C_i, M1, C_m, -, -) \wedge \mathbf{memberEnd}(Ass_k, P_i, -, -)\\ &\wedge \mathbf{memberEnd}(Ass_k, P_m, -, -))\ \wedge\ (\exists C_n, Ass_l\ (\mathbf{Class}(C_n, M2)\\ &\wedge \mathbf{OwnedAttribute}(P_j, C_n, M2, C_j, -, -) \wedge \mathbf{OwnedAttribute}(P_n, C_j, M2, C_n, -, -)\\ &\wedge \mathbf{memberEnd}(Ass_l, P_j, -, -) \wedge \mathbf{memberEnd}(Ass_l, P_n, -, -))))]\\ &\wedge \mathbf{Equivalence\_Associations}(Ass_k, Ass_l)\ \wedge [\mathbf{Equivalence\_Semantic\_Classes}(C_m, C_n)\\ &\vee\ \mathbf{Equivalence\_Syntactic\_Elts}(C_m, C_n)]]\end{aligned}$$

$$\begin{aligned}&\vee [[(\exists\, C_o\ (\mathbf{Class}(C_o, M1) \wedge \mathit{superClass}(C_o, C_i, M1)))\ \wedge\ (\exists C_p\ (\mathbf{Class}(C_p, M2) \wedge \mathit{superClass}(C_p, C_j, M2))) \wedge\\ &\mathbf{Equivalence\_Gen}(C_i, C_j, C_o, C_p, M1, M2)] \vee [(\exists\, C_o\ (\mathbf{Class}(C_o, M1) \wedge \mathit{superClass}(C_i, C_o, M1)))\ \wedge\\ &(\exists C_p\ (\mathbf{Class}(C_p, M2) \wedge \mathit{superClass}(C_j, C_p, M2))) \wedge \mathbf{Equivalence\_Gen}(C_o, C_p, C_i, C_j, M1, M2)]]\end{aligned}$$

Deux classes C_i et C_j, appartenant respectivement aux modèles M1 et M2, sont équivalentes en structure globale partielle (cf. Figure 4-21) si et seulement si une de ces conditions est satisfaite :

- Il existe deux classes C_m et C_n, une association Ass_k qui relie C_i à C_m et une association Ass_l qui relie C_j à C_n, C_m et C_n sont équivalentes syntaxiquement ou sémantiquement, et Ass_k et Ass_l sont équivalentes.
- Il existe deux classes C_o et C_p, avec une relation de généralisation entre C_i et C_o, et une relation de généralisation entre C_j et C_p, telle que les deux relations de généralisation sont équivalentes.

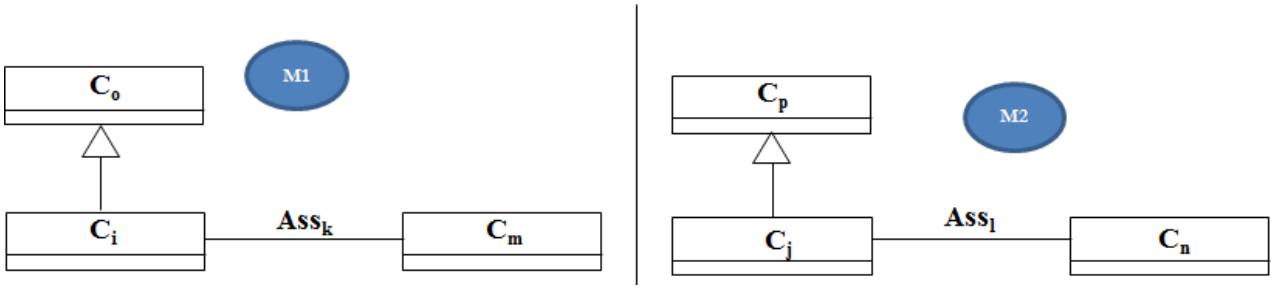


Figure 4-21 Schéma illustratif d'équivalence en structure globale de deux classes

- **Règle d'équivalence en structure globale totale (GTS : Global Total Structural : GTS) de deux classes**

Description informelle

Deux classes sont équivalentes en structure globale totale si et seulement si tous les éléments (association, relation de généralisation, classe) liés à chaque classe sont équivalents à tous les éléments liés à l'autre.

Exemple : Les classes « CarteCrédit » de M1 et « Carte » de M2 sont équivalentes en structure globale totale, car les associations « Joint » et « Joint » sont équivalentes et relient deux classes équivalentes « Compte » et « CompteBancaire ».

Représentation formelle

$$\mathbf{GTS\_EquivClasses}(C_i, C_j, M1, M2) \Leftrightarrow$$

$$\begin{aligned}
&[[(\forall C_m, Ass_k\ (\mathbf{Class}(C_m, M1) \wedge \mathbf{OwnedAttribute}(P_i, C_m, M1, C_i, -, -) \\
&\quad \wedge \mathbf{OwnedAttribute}(P_m, C_i, M1, C_m, -, -) \wedge \mathbf{memberEnd}(Ass_k, P_i, -, -) \\
&\quad \wedge \mathbf{memberEnd}(Ass_k, P_m, -, -))) \Rightarrow (\exists C_n, Ass_l\ (\mathbf{Class}(C_n, M2) \\
&\quad \wedge \mathbf{OwnedAttribute}(P_j, C_n, M2, C_j, -, -) \wedge \mathbf{OwnedAttribute}(P_n, C_j, M2, C_n, -, -) \\
&\quad \wedge \mathbf{memberEnd}(Ass_l, P_j, -, -) \wedge \mathbf{memberEnd}(Ass_l, P_n, -, -)) \\
&\quad \wedge \mathbf{Equivalence_Associations}(Ass_k, Ass_l) \wedge [\mathbf{Equivalence_Semantic_Classes}(C_m, C_n) \\
&\quad \vee \mathbf{Equivalence_Syntactic_Elts}(C_m, C_n)])] \\
&\wedge [(\forall C_n, Ass_l\ (\mathbf{Class}(C_n, M2) \wedge \mathbf{OwnedAttribute}(P_j, C_n, M2, C_j, -, -) \\
&\quad \wedge \mathbf{OwnedAttribute}(P_n, C_j, M2, C_n, -, -) \wedge \mathbf{memberEnd}(Ass_l, P_j, -, -) \\
&\quad \wedge \mathbf{memberEnd}(Ass_l, P_n, -, -))) \Rightarrow (\exists C_m, Ass_k\ (\mathbf{Class}(C_m, M1) \\
&\quad \wedge \mathbf{OwnedAttribute}(P_i, C_m, M1, C_i, -, -) \wedge \mathbf{OwnedAttribute}(P_m, C_i, M1, C_m, -, -) \\
&\quad \wedge \mathbf{memberEnd}(Ass_k, P_i, -, -) \wedge \mathbf{memberEnd}(Ass_k, P_m, -, -)) \\
&\quad \wedge \mathbf{Equivalence_Associations}(Ass_k, Ass_l) \wedge [\mathbf{Equivalence_Semantic_Classes}(C_m, C_n) \\
&\quad \vee \mathbf{Equivalence_Syntactic_Elts}(C_m, C_n)])]
\end{aligned}$$

$$\begin{aligned}
&[[(\forall C_o\ (\mathbf{Class}(C_o, M1) \wedge \mathbf{superClass}(C_o, C_i, M1))) \Rightarrow (\exists C_p\ (\mathbf{Class}(C_p, M2) \wedge \mathbf{superClass}(C_p, C_j, M2)) \wedge \\
&\mathbf{Equivalence_Gen}(C_i, C_j, C_o, C_p, M1, M2))] \vee [(\forall C_o\ (\mathbf{Class}(C_o, M1) \wedge \\
&\mathbf{superClass}(C_i, C_o, M1))) \Rightarrow (\exists C_p\ (\mathbf{Class}(C_p, M2) \wedge \mathbf{superClass}(C_j, C_p, M1)) \wedge \\
&\mathbf{Equivalence_Gen}(C_o, C_p, C_i, C_j, M1, M2))]]
\end{aligned}$$

$$\begin{aligned}
&\wedge [[(\forall C_p\ (\mathbf{Class}(C_p, M2) \wedge \mathbf{superClass}(C_p, C_j, M2))) \Rightarrow (\exists C_o\ (\mathbf{Class}(C_o, M1) \wedge \mathbf{superClass}(C_o, C_i, M1)) \\
&\quad \wedge \mathbf{Equivalence_Gen}(C_i, C_j, C_o, C_p, M1, M2))] \vee [(\forall C_p\ (\mathbf{Class}(C_p, M2) \\
&\quad \wedge \mathbf{superClass}(C_j, C_p, M2))) \Rightarrow (\exists C_o\ (\mathbf{Class}(C_o, M1) \wedge \mathbf{superClass}(C_i, C_o, M1)) \\
&\quad \wedge \mathbf{Equivalence_Gen}(C_o, C_p, C_i, C_j, M1, M2))]]]
\end{aligned}$$

Deux classes C_i et C_j, appartenant respectivement aux modèles M1 et M2, sont équivalentes en structure globale totale (cf. Figure 4-21) si et seulement si les quatre conditions ci-dessous sont satisfaites :

- Pour toute association Ass_k et pour toute classe C_m, avec Ass_k une association qui relie C_i à C_m, il existe une association Ass_l et une classe C_n, avec Ass_l une association qui relie C_j à C_n, Ass_k et Ass_l sont équivalentes et C_m et C_n sont équivalentes syntaxiquement ou sémantiquement
- Pour toute association Ass_l et pour toute classe C_n, avec Ass_l une association qui relie C_j à C_n, il existe une association Ass_k et une classe C_m, avec Ass_k une association qui relie C_i à C_m, Ass_k et Ass_l sont équivalentes et C_m et C_n sont équivalentes syntaxiquement ou sémantiquement.
- Pour toute classe C_o reliée par une relation de généralisation à C_i, il existe une classe C_p reliée par une relation de généralisation à C_j, et les deux relations de généralisation sont équivalentes.
- Pour toute classe C_p reliée par une relation de généralisation à C_j, il existe une classe C_o reliée par une relation de généralisation à C_i, et les deux relations de généralisation sont équivalentes.

- **Règle d'équivalence en structure locale partielle (LPS : Local Partiel Structural) de deux classes**

Description informelle

Deux classes sont équivalentes en structure locale partielle si elles ont au moins un attribut équivalent ou une opération équivalente.

Exemple : Les classes « Compte » de M1 et « CompteBancaire » de M2 sont équivalentes en structure locale partielle.

Représentation formelle

$$\mathbf{LPS\_EquivClasses}(C_i, C_j, M1, M2) \Leftrightarrow [[\exists\, \mathbf{Att_k\ OwnedAttribute}(Att_k, C_i, M1, -, -, -)$$
$$\wedge \exists\, \mathbf{Att_l OwnedAttribute}(Att_l, C_j, M2, -, -, -) \wedge \mathbf{Equivalence\_Attributes}(Att_k, Att_l, C_i, C_j, M1, M2)]$$
$$\vee [\exists\, \mathbf{Op_k OwnedOperation}(OP_k, C_i, M1, -)$$
$$\wedge \exists\, \mathbf{OP_l OwnedOperation}(Op_l, C_j, M2, -) \wedge \mathbf{Equivalence\_Operations}(Op_k, Op_l, C_i, C_j, M1, M2)]]$$

Deux classes C_i et C_j appartenant respectivement aux modèles M1 et M2 sont équivalentes en structure locale partielle (cf. Figure 4-22), si et seulement si il existe au moins un attribut Att_k de C_i (ou une opération Op_k) de C_i qui est équivalent respectivement à un attribut Att_l de C_j (ou à une opération Op_l de C_j).

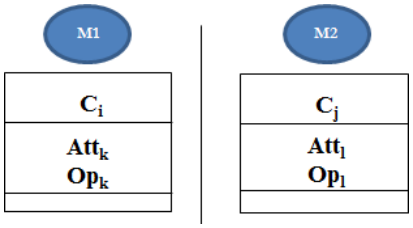


Figure 4-22 Schéma illustratif d'équivalence en structure locale de deux classes

- **Règle d'équivalence en structure locale totale (LTS : Local Total Structural) de deux classes**

Description informelle

Deux classes sont équivalentes en structure locale totale si tous leurs attributs et opérations sont équivalents.

Exemple : Les classes « CarteCrédit » de M1 et « Carte » de M2 sont équivalentes en structure locale totale.

Représentation formelle

$$\mathbf{LTS\_EquivClasses}(C_i, C_j, M1, M2) \Leftrightarrow [[[\forall\, \mathbf{Att_k\ OwnedAttribute}(Att_k, C_i, M1, -, -, -) \Rightarrow$$
$$(\exists\, \mathbf{Att_l\ OwnedAttribute}(Att_l, C_j, M2, -, -, -) \wedge \mathbf{Equivalence\_Attributes}(Att_k, Att_l, C_i, C_j, M1, M2))\ ] \wedge$$
$$[\forall\, \mathbf{Att_l\ OwnedAttribute}(Att_l, C_j, M2, -, -, -) \Rightarrow (\exists\, \mathbf{Att_k}\ (\mathbf{OwnedAttribute}(Att_k, C_i, M1, -, -, -) \wedge$$
$$\mathbf{Equivalence\_Attributes}(Att_k, Att_l, C_i, C_j, M1, M2))]]$$

$$\wedge [[\forall\, \mathbf{Op_k\ OwnedOperation}(\mathbf{Op_k}, \mathbf{C_i}, \mathbf{M1}, -) \Rightarrow (\exists\, \mathbf{Op_l\ OwnedOperation}(\mathbf{Op_l}, \mathbf{C_j}, \mathbf{M2}, -) \wedge \mathbf{Equivalence\_Operations}(\mathbf{Op_k}, \mathbf{Op_l}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2}))] \wedge [\forall\, \mathbf{Op_l\ OwnedOperation}(\mathbf{Op_l}, \mathbf{C_j}, \mathbf{M2}, -) \Rightarrow (\exists\, \mathbf{Op_k}\ (\mathbf{OwnedOperation}(\mathbf{Op_k}, \mathbf{C_i}, \mathbf{M1}, -) \wedge \mathbf{Equivalence\_Operations}(\mathbf{Op_k}, \mathbf{Op_l}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2}))]]$$

Deux classes C_i et C_j, appartenant respectivement aux modèles M1 et M2, sont équivalentes en structure locale totale (cf. Figure 4-22), si et seulement si les quatre conditions ci-dessous sont satisfaites :

- Pour tout attribut Att_k de C_i, il existe un attribut Att_l de C_j qui lui est équivalent
- Pour tout attribut Att_l de C_j, il existe un attribut Att_k de C_i qui lui est équivalent
- Pour toute opération Op_k de C_i, il existe une opération Op_l de C_j qui lui est équivalente
- Pour toute opération Op_l de C_j, il existe une opération Op_k de C_i qui lui est équivalente.

- **Règle d'hyponymie de deux classes**

Description informelle

Deux classes sont hyponymes si, dans une ontologie, il existe une relation de subsomption entre elles.

Exemple : Les classes « Opération » de M1 et « Versement » de M2 sont en relation d'hyponymie, car

- selon l'ontologie de domaine, il existe une relation de subsomption entre elles.
- et il n'existe pas dans M1 une classe équivalente à « Versement » qui hérite de la classe « Opération »
- et il n'existe pas dans M2 une classe équivalente à « Opération » qui a comme sous-classe « Versement ».

Représentation formelle

$$\mathbf{Hyponymy\_Classes}(\mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2}) \Leftrightarrow [\mathbf{Class}(\mathbf{C_i}, \mathbf{M1}) \wedge \mathbf{Class}(\mathbf{C_j}, \mathbf{M2})$$

$$\wedge ((\mathbf{OntoSuperClassOf}(\mathbf{C_i}, \mathbf{C_j}) \vee (\exists \mathbf{e\ String}(\mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{e}) \wedge \mathbf{OntoSuperClassOf}(\mathbf{e}, \mathbf{C_j})) \vee (\exists \mathbf{c\ String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{c}) \wedge \mathbf{OntoSuperClassOf}(\mathbf{C_i}, \mathbf{c})) \vee (\exists \mathbf{e\ String}(\mathbf{e}) \wedge \exists \mathbf{c\ String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{c}) \wedge \mathbf{OntoSuperClassOf}(\mathbf{c}, \mathbf{e})))$$

$$\wedge (\neg\exists\, \mathbf{C_k\ Class}(\mathbf{C_k}, \mathbf{M2}) \wedge \mathbf{Equivalence\_Classes}(\mathbf{C_i}, \mathbf{C_k}, \mathbf{M1}, \mathbf{M2}, \mathbf{opt}) \wedge \mathbf{superClass}(\mathbf{C_j}, \mathbf{C_k}, \mathbf{M2})) \wedge (\neg\exists\, \mathbf{C_l\ Class}(\mathbf{C_l}, \mathbf{M1}) \wedge \mathbf{Equivalence\_Classes}(\mathbf{C_j}, \mathbf{C_l}, \mathbf{M1}, \mathbf{M2}, \mathbf{opt}) \wedge \mathbf{superClass}(\mathbf{C_l}, \mathbf{C_i}, \mathbf{M1})))$$

$$\vee ((\mathbf{OntoSuperClassOf}(\mathbf{C_j}, \mathbf{C_i}) \vee (\exists \mathbf{e\ String}(\mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{e}) \wedge \mathbf{OntoSuperClassOf}(\mathbf{C_j}, \mathbf{e})) \vee (\exists \mathbf{c\ String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{c}) \wedge \mathbf{OntoSuperClassOf}(\mathbf{c}, \mathbf{C_i})) \vee (\exists \mathbf{e\ String}(\mathbf{e}) \wedge \exists \mathbf{c\ String}(\mathbf{c}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_j}, \mathbf{e}) \wedge \mathbf{Equivalence\_Syntactic\_Elts}(\mathbf{C_i}, \mathbf{c}) \wedge \mathbf{OntoSuperClassOf}(\mathbf{e}, \mathbf{c})))$$

$$\wedge (\neg\exists\, \mathbf{C_k\ Class}(\mathbf{C_k}, \mathbf{M2}) \wedge \mathbf{Equivalence\_Classes}(\mathbf{C_i}, \mathbf{C_k}, \mathbf{M1}, \mathbf{M2}, \mathbf{opt}) \wedge \mathbf{superClass}(\mathbf{C_k}, \mathbf{C_j}, \mathbf{M2})) \wedge (\neg\exists\, \mathbf{C_l\ Class}(\mathbf{C_l}, \mathbf{M1}) \wedge \mathbf{Equivalence\_Classes}(\mathbf{C_j}, \mathbf{C_l}, \mathbf{M1}, \mathbf{M2}, \mathbf{opt}) \wedge \mathbf{superClass}(\mathbf{C_i}, \mathbf{C_l}, \mathbf{M1})))]$$

Deux classes C_i et C_j, appartenant respectivement aux modèles M1 et M2, sont hyponymes, si et seulement si l'une de ces conditions est satisfaite :

- Il existe, dans l'ontologie de domaine, une sous-classe C_i (ou une chaîne de caractères équivalente syntaxiquement à C_i) de la super-classe C_j (ou une chaîne de caractères équivalente syntaxiquement à C_j) ; et il n'existe pas dans M2 une classe équivalente à C_i qui hérite de la classe C_j ; et il n'existe pas dans M1 une classe équivalente à C_j qui a comme sous-classe C_i.
- Il existe, dans l'ontologie de domaine, une sous-classe C_j (ou une chaîne de caractères équivalente syntaxiquement à C_j) de la super-classe C_i (ou une chaîne de caractères équivalente syntaxiquement à C_i) ; et il n'existe pas dans M2 une classe équivalente à C_i qui a comme sous-classe C_j ; et il n'existe pas dans M1 une classe équivalente à C_j qui hérie de la classe C_i.

Remarques

a) Polysémie des éléments

Deux classes sont en relation de polysémie, si est seulement si elles ont le même nom (identité syntaxique), mais ont une structure locale et globale différentes. Nous ne considérons pas ces éléments équivalents, donc ils feront partie des éléments isolés. Deux attributs/opérations/paramètres sont en relation de polysémie, si et seulement si ils ont le même nom (identité syntaxique), une structure globale équivalente mais une structure locale différente. Ces éléments feront également partie des éléments isolés.

La polysémie des éléments ne sera pas explicitée en langage formel car, en appliquant toutes les autres règles de comparaison, si deux éléments sont en relation de polysémie, alors ils ne seront pas jugés équivalents et apparaîtront donc en tant qu'éléments isolés.

- **Règle de renommage d'éléments en relation de polysémie**

Description informelle

Dans le cas de deux éléments (classes, attributs, associations, opérations ou paramètres) qui sont en relations de polysémie, il faut les renommer en rajouter respectivement « 1 » et « 2 » aux deux noms.

$$\begin{aligned}&\mathbf{RenamePolysemy}(\mathbf{elt_i}, \mathbf{elt_j}, \boldsymbol{M1}, \boldsymbol{M2})\\ &\quad \Leftrightarrow [(\mathbf{Syntactic\_Identity}\ (\mathbf{elt_i}, \mathbf{elt_j})\\ &\quad \wedge \mathbf{IsolatedElt}\ (\mathbf{elt_i}, \mathbf{M1})\ \mathbf{IsolatedElt}\ (\mathbf{elt_j}, \mathbf{M2})) \Rightarrow (\mathbf{Rename}(\mathbf{elt_i}, \mathbf{elt_i 1})\\ &\quad \wedge \mathbf{Rename}(\mathbf{elt_j}, \mathbf{elt_j 2}))]\end{aligned}$$

Description formelle

Dans le cas de deux éléments elt_i et elt_j qui sont en relation de polysémie, il faut les renommer en rajouter à elt_i et elt_j respectivement les deux chaînes de caractères « 1 » et « 2 ».

b) Unicité d'un élément du mapping

Soient M1 et M2 les modèles à composer. Il faut noter qu'un élément de M1 ne peut être équivalent qu'à un unique élément de M2 et un élément de M2 ne peut être équivalent qu'à un unique élément de M1. En effet, si un élément de M1 est équivalent à deux éléments de modèle M2, cela signifierait que

M2 comporte deux éléments équivalents, ce qui est exclu dans un diagramme de classes UML qui exige l'unicité des éléments.

4.2.5.2. Règles de fusion et de translation

Nous définissons, dans ce qui suit, les règles de fusion et de translation en logique des prédicats des éléments de deux modèles M1 et M2.

Dans notre approche, pour composer deux modèles M1 et M2, nous n'avons pas opté d'ajouter à un modèle les éléments de l'autre, mais nous avons jugé plus simple de créer un nouveau modèle résultat auquel nous rajoutons les éléments de M1 et de M2.

Dans les exemples présentés dans ce qui suit, nous considérons que M1 est le modèle principal. Pour faciliter la compréhension des exemples, nous présentons en rouge les éléments fusionnés et en bleu les éléments translatés.

- **Fusion de classes équivalentes**

Description informelle

Pour chaque correspondance du mapping d'équivalence de deux classes, nous créons, dans le modèle résultat, une classe (cf. Figure 4-23). La classe engendrée a les propriétés suivantes :

- Nom : elle prend le nom de la classe appartenant au modèle principal, sauf le cas de l'inclusion, où elle prend le nom le plus long
- Attributs de la classe résultante :
  - Fusion des attributs équivalents appartenant aux classes équivalentes
  - Translation des attributs isolés appartenant aux classes équivalentes
- Opérations de la classe résultante :
  - Fusion des opérations équivalentes appartenant aux classes équivalentes
  - Translation des opérations isolées appartenant aux classes équivalentes

Exemple : Les deux classes « Compte » de M1 et « CompteBancaire » de M2 sont fusionnées.

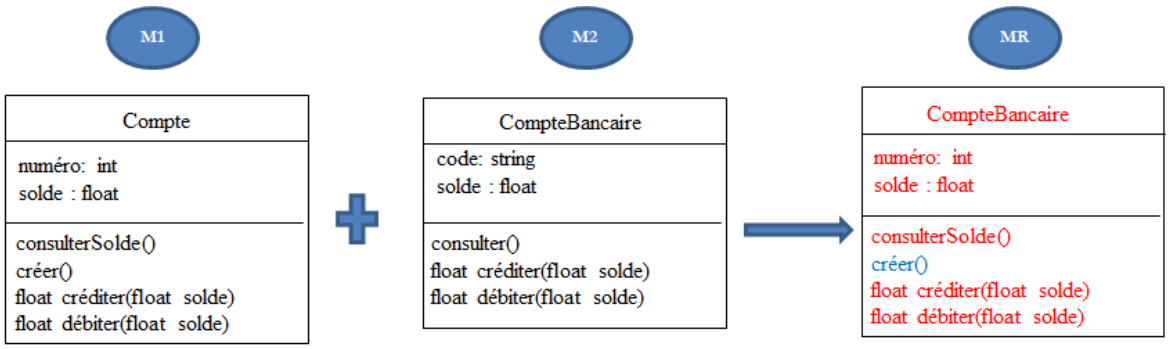


Figure 4-23 Fusion de deux classes

Cas particulier

« Personnel » de M1 et « Employé » de M2 sont des classes équivalentes, « Employé » une sous-classe de « Personne », l'attribut « mail » de « Personnel » est équivalent à « email », attribut de « Employé » hérité de « Personne » (cf. Figure 4-24). Dans ce cas, même si les deux attributs font partie du mapping d'équivalence, il ne faut pas les fusionner.

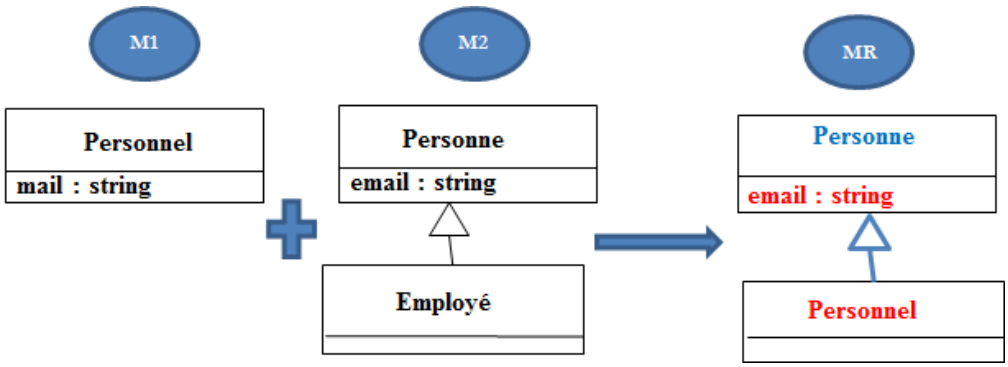


Figure 4-24 Fusion d'attributs et d'opération : idée de solution

En effet, une première idée (cf. Figure 4-24) serait de les fusionner et de retrouver l'attribut résultat dans la super-classe « Personne ». Cependant, dans ce cas, il y aura des conflits dans le modèle résultat MR.

Prenons le cas de la Figure 4-25. Si nous appliquons la règle citée précédemment, la classe « Personne » aura les deux attributs : « email » (résultat de la fusion des attributs de « Personnel » et « Personne ») et « email » (résultat de la fusion des attributs de « Client » et « Personne »).

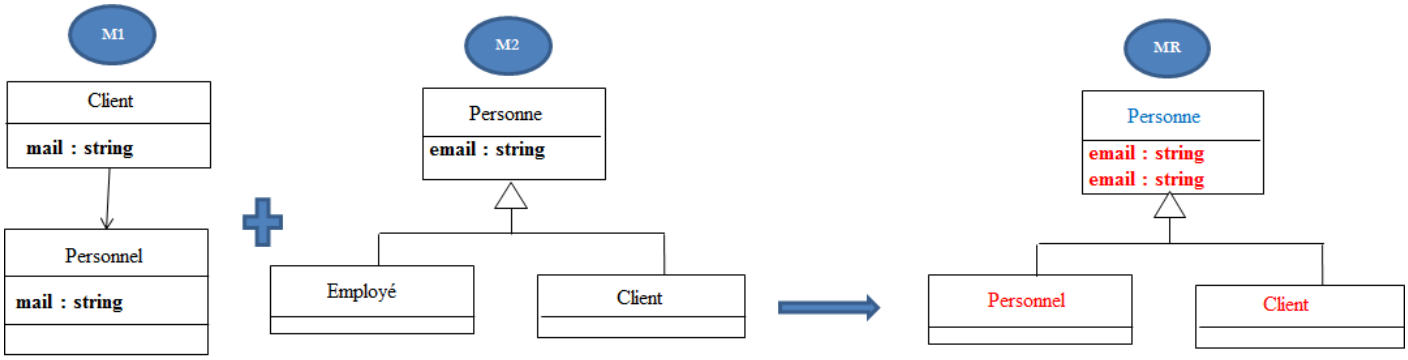


Figure 4-25 Fusion d'attributs : conflit

Afin de gérer ce conflit (cf. Figure 4-26), il faut :

- Ne pas fusionner l'attribut « mail » de « Personnel » avec l'attribut « email » de « Personne »
- Ne pas fusionner l'attribut « mail » de « Client » avec l'attribut « email » de « Personne »
- Translater « email » de M2 dans la classe résultante de « Personne » (lors de la translation de « Personne »)

La même logique est appliquée pour les opérations.

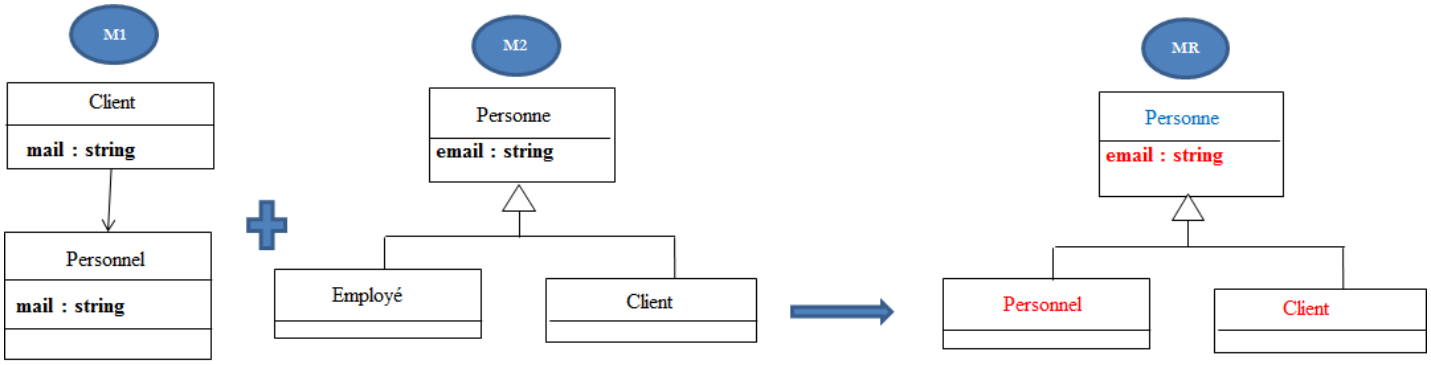


Figure 4-26 Fusion d'attributs : résolution des conflits

Représentation formelle

$$\mathbf{Fusion\_Classes}(C_i, C_j, RC_{ij}, M1, M2, MR) \Leftrightarrow [(\mathbf{Class}(C_i, M1) \wedge \mathbf{Class}(C_j, M2) \wedge \mathbf{Class}(RC_{ij}, MR) \wedge \mathbf{Equivalence\_Classes}(C_i, C_j, M1, M2, -)) \Rightarrow$$

//Nom

$$(((\mathbf{MainModel}(M1) \wedge \mathbf{BelongingToMainModel}(C_i, M1) \wedge \neg\mathbf{InclusionSyn}(C_i, C_j)) \Rightarrow \mathbf{EqualString}(RC_{ij}, C_i)) \wedge ((\mathbf{MainModel}(M2) \wedge \mathbf{BelongingToMainModel}(C_j, M2) \wedge \neg\mathbf{InclusionSyn}(C_j, C_i)) \Rightarrow \mathbf{EqualString}(RC_{ij}, C_j)) \wedge (\mathbf{InclusionSyn}(C_i, C_j) \Rightarrow \mathbf{EqualString}(RC_{ij}, C_j)) \wedge (\mathbf{InclusionSyn}(C_j, C_i) \Rightarrow \mathbf{EqualString}(RC_{ij}, C_i)))$$

//Fusion des attributs

$$\wedge ((\forall\, Att_i\ \mathbf{OwnedAttribute}(Att_i, C_i, M1, -, -, -) \wedge \forall\, Att_j\ \mathbf{OwnedAttribute}(Att_j, C_j, M2, -, -, -) \wedge \mathbf{Equivalence\_Attributes}(Att_i, Att_j, C_i, C_j, M1, M2)) \Rightarrow (\mathbf{Fusion\_Attributes}(Att_i, Att_j, RAtt_{ij}, C_i, C_j, RC_{ij}, M1, M2, MR) \wedge \mathbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, -, -)))$$

//Translation des attributs

$$\wedge ((\forall\, Att_i\ \mathbf{OwnedAttribute}(Att_i, C_i, M1, -, -, -) \wedge \mathbf{IsolatedAtt}(Att_i, C_i, M1)) \Rightarrow (\mathbf{Translation\_Attributes}(Att_i, RAtt_i, C_i, RC_{ij}, M1, MR) \wedge \mathbf{OwnedAttribute}(RAtt_i, RC_{ij}, MR, -, -, -)))$$

$$\wedge ((\forall\, Att_j\ \mathbf{OwnedAttribute}(Att_j, C_j, M2, -, -, -) \wedge \mathbf{IsolatedAtt}(Att_j, C_j, M2)) \Rightarrow (\mathbf{Translation\_Attributes}(Att_j, RAtt_j, C_j, RC_{ij}, M2, MR) \wedge \mathbf{OwnedAttribute}(RAtt_j, RC_{ij}, MR, -, -, -)))$$

//Fusion des opérations

$$\wedge ((\forall\, Op_i\ \mathbf{OwnedOperation}(Op_i, C_i, M1, -) \wedge \forall\, Op_j\ \mathbf{OwnedOperation}(Op_j, C_j, M2, -) \wedge \mathbf{Equivalence\_Operations}(Op_i, OP_j, C_i, C_j, M1, M2)) \Rightarrow (\mathbf{Fusion\_Operations}(Op_i, Op_j, ROp_{ij}, C_i, C_j, RC_{ij}, M1, M2, MR) \wedge \mathbf{OwnedOperation}(ROp_{ij}, RC_{ij}, MR, -)))$$

//Translation des opérations

$$\wedge ((\forall\, \mathbf{Op_i\ OwnedOperation(Op_i, C_i, M1, -)} \wedge \mathbf{IsolatedOp(Op_i, C_i, M1)}) \Rightarrow (\mathbf{Translation\_Operations(Op_i, ROp_i, C_i, RC_{ij}, M1, MR)} \wedge \mathbf{OwnedOperation(ROp_i, RC_{ij}, MR, -)})) \wedge ((\forall\, \mathbf{Op_j\ OwnedOperation(Op_j, C_j, M2, -)} \wedge \mathbf{IsolatedOp(Op_j, C_j, M2)}) \Rightarrow (\mathbf{Translation\_Operation(Op_j, ROp_j, C_j, RC_{ij}, M2, MR)} \wedge \mathbf{OwnedOperation(ROp_j, RC_{ij}, MR, -)}))]$$

La fusion de deux classes équivalentes C_i et C_j, appartenant respectivement aux modèles M1 et M2, crée dans le modèle résultat MR une classe RC_{ij} (cf. Figure 4-27 et Figure 4-28). La classe engendrée a les propriétés suivantes :

- Nom : Elle prend le nom de la classe appartenant au modèle principal, sauf dans le cas de l'inclusion, où elle prend le nom le plus long
- Attributs de la classe résultante :
  - Fusion de tous les attributs équivalents Att_i et Att_j appartenant respectivement aux classes C_i et C_j,
  - Translation des attributs isolés Att_i appartenant à C_i et Att_j appartenant à C_j
- Opérations de la classe résultante :
  - Fusion de toutes les opérations équivalentes Op_i et Op_j appartenant respectivement aux classes C_i et C_j
  - Translation des opérations isolées Op_i appartenant à C_i et Op_j appartenant à C_j

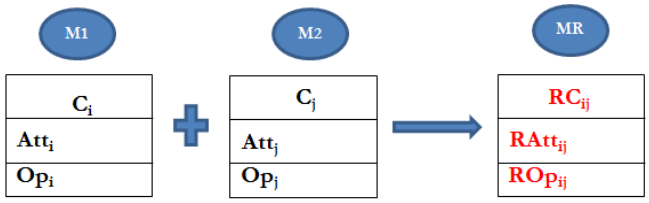


Figure 4-27 Fusion de classes avec fusion d'attributs et d'opérations équivalents

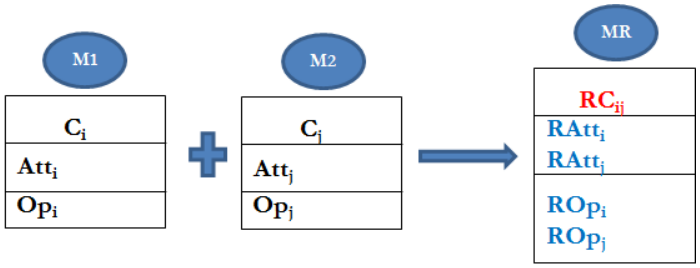


Figure 4-28 Fusion de classes avec translation d'attributs et d'opérations isolés

Le cas particulier, discuté dans la règle informelle de la fusion de classes, est pris en compte implicitement. En effet, dans cette règle formelle, si le cas particulier se présente, on ne crée pas un attribut résultat dans la sous-classe résultante.

- **Translation de classes**

Description informelle

Pour chaque classe isolée, nous créons dans le modèle résultat une classe correspondante (cf. Figure 4-29). La classe engendrée a les propriétés suivantes :

- Nom : garde son nom
- Attributs : garde ses attributs
- Opérations : garde ses opérations

Exemple : La classe « Chéquier » de M2 est translatée dans le MR.

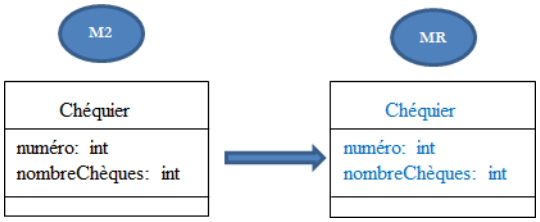


Figure 4-29 Translation de classe

Représentation formelle

$$\textbf{Translation\_Class}(C_i, M, RC_i, MR) \Leftrightarrow [(\textbf{Class}(C_i, M) \wedge \textbf{IsolatedClass}(C_i, M)) \Rightarrow (\textbf{Class}(RC_i, MR) \wedge \textbf{EqualString}(RC_i, C_i) \wedge ((\forall \textbf{Att}_i\ \textbf{OwnedAttribute}(\textbf{Att}_i, C_i, M, -, -, -) \wedge \textbf{IsolatedAtt}(\textbf{Att}_i, C_i, M)) \Rightarrow (\textbf{Translation\_Attribute}(\textbf{Att}_i, \textbf{RAtt}_i, C_i, RC_i, M, MR) \wedge \textbf{OwnedAttribute}(\textbf{RAtt}_i, RC_i, M, -, -, -))) \wedge ((\forall \textbf{Op}_i\ \textbf{OwnedOperation}(\textbf{Op}_i, C_i, M, -) \wedge \textbf{IsolatedOp}(\textbf{Op}_i, C_i, M)) \Rightarrow (\textbf{Translation\_Operation}(\textbf{Op}_i, \textbf{ROp}_i, C_i, RC_i, M, MR) \wedge \textbf{OwnedOperation}(\textbf{ROp}_i, RC_i, MR, -))))]$$

Pour chaque classe isolée C_i appartenant au modèle M, nous créons dans le modèle résultat MR une classe RC_i (cf. Figure 4-30). La classe engendrée a les propriétés suivantes :

- Nom : garde son nom
- Attributs : garde ses attributs Att_i
- Opérations : garde ses opérations Op_i

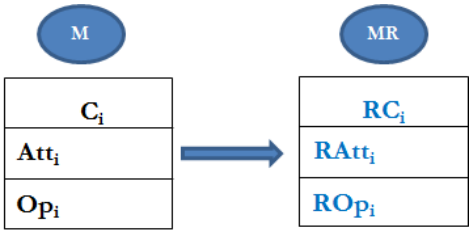


Figure 4-30 Translation de classes avec translation d'attributs et d'opérations

- **Création de relation de généralisation entre les classes hyponymes**

Description informelle

Pour chaque correspondance du mapping d'hyponymie de deux classes C_i et C_j :

- Nous créons, dans le modèle résultat, une relation de généralisation. Les classes qu'elle relie sont la classe résultante de la super-classe et la classe résultante de la sous-classe.
- Nous ne créons pas les classes dans le modèle résultat, car elles seront translatées par les autres règles.
- Nous éliminons les doublons (attributs/opérations/associations/relations de généralisation) de la classe résultante de la sous-classe.

Exemple : Les classes « Opération » de M1 et « Versement » de M2 sont hyponymes. Il faut donc créer, dans le modèle MR, une relation de généralisation entre leurs classes résultantes « Opération » et « Versement ».

Comme le montre la Figure 4-31, il existe des attributs équivalents de « Opération » et «Versement » tels que « date » et « dateVersement », et « montant » (de « Opération ») et « montant » (de «Versement »), qu'il faut supprimer de la classe résultante « Versement ».

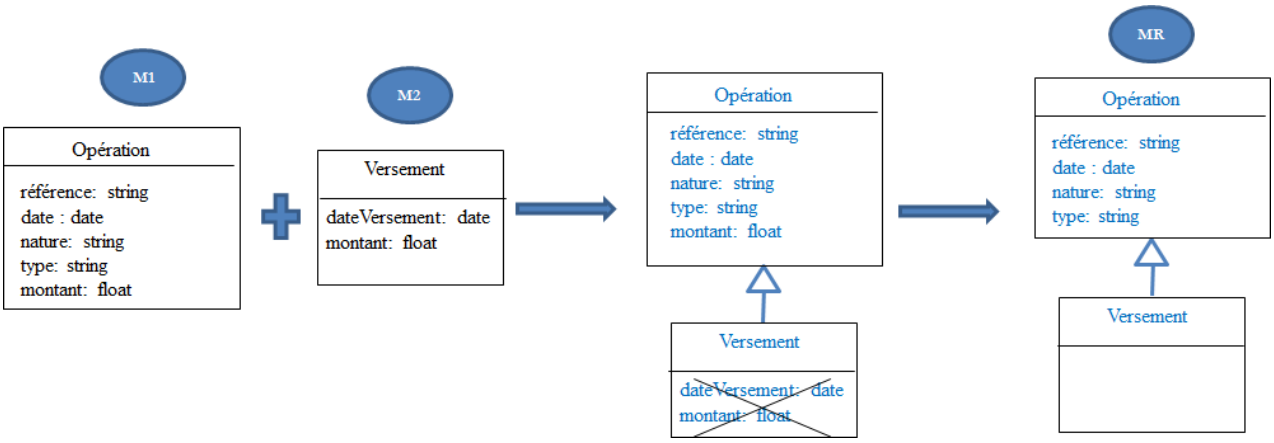


Figure 4-31 Translation des classes hyponymes

Représentation formelle

$$\mathbf{Translation\_Hyponymy\_Classes}(C_i, C_j, RC_i, RC_j, M1, M2, MR) \Leftrightarrow$$

//Création de la relation de généralisation

$$[[(\mathbf{Class}(C_i, M1) \wedge \mathbf{Class}(C_j, M2) \wedge \mathbf{Class}(RC_i, MR) \wedge \mathbf{Class}(RC_j, MR) \wedge \mathbf{Hyponymy\_Classes}(C_i, C_j, M1, M2) \wedge \mathbf{ResultElt}(C_i, RC_i) \wedge \mathbf{ResultElt}(C_j, RC_j)) \Rightarrow \mathbf{superClass}(RC_j, RC_i, MR)] \wedge$$

//Eliminer les attributs en double

$$[(\exists\ RAtt_i\ OwnedAttribute(RAtt_i, RC_i, MR, -, -, -) \wedge \exists\ RAtt_j\ OwnedAttribute(RAtt_j, RC_j, MR, -, -, -) \wedge Equivalence\_Attributes(RAtt_i, RAtt_j, RC_i, RC_j, MR, MR)) \Rightarrow RemoveAtt(RAtt_i, RC_i, MR)] \wedge$$

//Eliminer les opérations en double

$$[(\exists\ ROp_i\ OwnedOperation(ROp_i, RC_i, MR, -) \wedge \exists\ ROP_j\ OwnedOperation(ROp_j, RC_j, MR, -) \wedge Equivalence\_Operations(ROp_i, ROp_j, RC_i, RC_j, MR, MR)) \Rightarrow RemoveOp(ROp_i, RC_i, MR)]]$$

//Eliminer les associations en double

$$[(\exists\ RAss_i, RC_k(OwnedAttribute(RP_i, RC_k, MR, RC_i, -, -) \wedge memberEnd(RAss_i, RP_i, -, -) \wedge OwnedAttribute(RP_k, RC_i, MR, RC_k, -, -) \wedge memberEnd(RAss_i, RP_k, -, -)) \wedge \exists\ RAss_j\ (OwnedAttribute(RP_j, RC_k, MR, RC_j, -, -) \wedge memberEnd(RAss_j, RP_j, -, -) \wedge OwnedAttribute(RP_k, RC_j, MR, RC_k, -, -) \wedge memberEnd(RAss_j, RP_k, -, -)) \wedge Equivalence\_Associations(RAss_i, RAss_j)) \Rightarrow RemoveAss(RAss_i, RC_i, RC_k, MR)] \wedge$$

//Eliminer les relations de généralisation en double – cas 1

$$[(\exists\ RC_{kl}\ (superClass(RC_{kl}, RC_i, MR) \wedge superClass(RC_{kl}, RC_j, MR)) \Rightarrow RemoveRG(RC_i, RC_{kl}, MR)] \wedge$$

//Eliminer les relations de généralisation en double – cas 2

$$[(\exists\ RC_{mn}\ (superClass(RC_i, RC_{mn}, MR) \wedge superClass(RC_j, RC_{mn}, MR)) \Rightarrow (RemoveRG(RC_{mn}, RC_j, MR))]]$$

Pour chaque correspondance du mapping d'hyponymie de deux classes C_i et C_j, il faut :

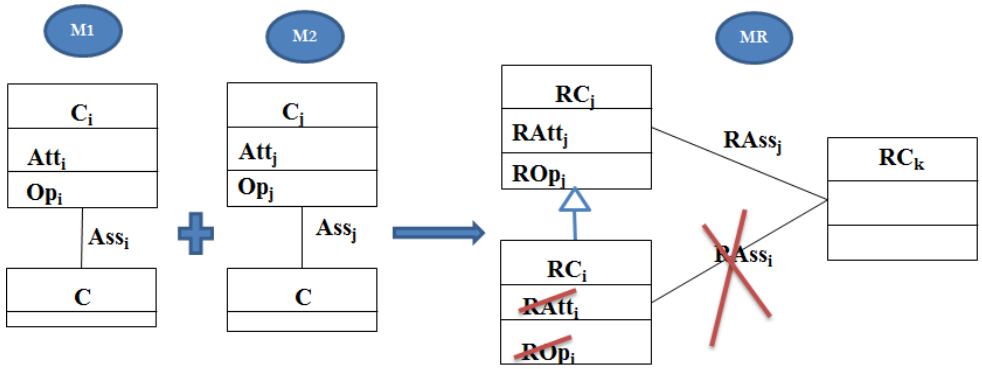


Figure 4-32 Translation de classes hyponymes et élimination des attributs, opérations et associations en double

- créer dans le modèle résultat MR une relation de généralisation (cf. Figure 4-32). Les classes qu'elle relie sont la classe résultante de la super-classe RC_j et la classe résultante de la sous-classe RC_i.
- éliminer les :
  - Attributs en double : s'il existe dans le modèle résultat MR un attribut $RAtt_i$ de RC_i équivalent à un attribut $RAtt_j$ de RC_j, alors il faut éliminer l'attribut résultat $RAtt_i$ appartenant à la classe résultat RC_i.

- Opérations en double : s'il existe dans le modèle résultat MR une opération ROp_i de RC_i équivalente à une opération ROp_j de RC_j alors il faut éliminer l'opération résultat ROp_i appartenant à la classe résultat RC_i.
- Associations en double : s'il existe dans le modèle résultat MR une association $RAss_i$ qui relie RC_i à une classe RC_k, équivalente à une association $RAss_j$ qui relie RC_j à la même classe RC_k, alors il faut éliminer l'association résultat $RAss_i$ reliant la classe résultat RC_i à RC_k.
- Relations de généralisation en double (cf. Figure 4-33) :
  - Cas 1 : s'il existe dans le modèle résultat MR une super-classe RC_{kl} des classes RC_i et RC_j, alors il faut éliminer la relation de généralisation qui relie RC_{kl} à RC_i.
  - Cas 2 : s'il existe dans le modèle résultat MR une sous-classe RC_{mn} des classes RC_i et RC_j, alors il faut éliminer la relation de généralisation qui relie RC_{mn} à RC_j.

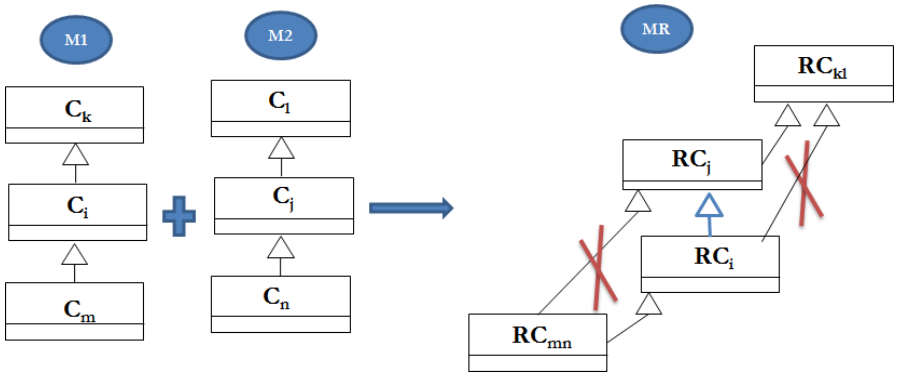


Figure 4-33 Translation de classes hyponymes et élimination des relations de généralisation en double

- **Fusion d'attributs**

Description informelle

Pour chaque correspondance du mapping d'équivalence de deux attributs, nous créons dans le modèle résultat un attribut (cf. Figure 4-34). L'attribut engendré a les propriétés suivantes :

- Nom : prend le nom de l'attribut appartenant au modèle principal, sauf dans le cas de l'inclusion, où elle prend le nom le plus long
- Type : prend le plus grand type. Exemple (si les attributs à fusionner sont un de type float et l'autre de type int alors le type de l'attribut résultat sera de type float)
- Multiplicité maximale (par défaut égales à 1) : garde la plus grande valeur.
- Multiplicité minimale (par défaut égales à 1) : garde la plus petite valeur.

Exemple : les deux attributs « numéro » de « Compte » de M1 et « code » de « CompteBancaire » de M2 sont fusionnés en l'attribut résultat « numéro » de la classe résultante « CompteBancaire ».

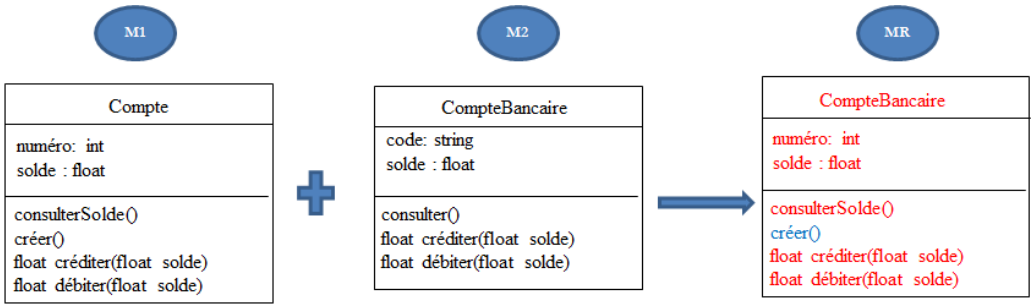


Figure 4-34 Fusion d'attributs et d'opérations et translation d'opération

Représentation formelle

$$\begin{aligned}
&\textbf{Fusion_Attributes}(Att_i, Att_j, RAtt_{ij}, C_i, C_j, RC_{ij}, M1, M2, MR)\\
&\quad \Leftrightarrow [(\textbf{Equivalence_Attributes}(Att_i, Att_j, C_i, C_i, M1, M2)\\
&\quad \wedge \textbf{OwnedAttribute}(Att_i, C_i, M1, T1, L1, U1)\\
&\quad \wedge \textbf{OwnedAttribute}(Att_j, C_j, M2, T2, L2, U2) \wedge \textbf{FusionResultElt}(C_i, C_j, RC_{ij})) \Rightarrow\\
&//\text{Nom}\\
&\quad ((((\textbf{MainModel}(M1) \wedge \textbf{BelongingToMainModel}(Att_i, M1)\\
&\quad \wedge \neg\textbf{InclusionSyn}(Att_i, Att_j)) \Rightarrow \textbf{EqualString}(RAtt_{ij}, Att_i))\\
&\quad \wedge ((\textbf{MainModel}(M2) \wedge \textbf{BelongingToMainModel}(Att_j, M2)\\
&\quad \wedge \neg\textbf{InclusionSyn}(Att_j, Att_i)) \Rightarrow \textbf{EqualString}(RAtt_{ij}, Att_j))\\
&\quad \wedge (\textbf{InclusionSyn}(Att_i, Att_j) \Rightarrow \textbf{EqualString}(RAtt_{ij}, Att_j))\\
&\quad \wedge (\textbf{InclusionSyn}(Att_j, Att_i) \Rightarrow \textbf{EqualString}(RAtt_{ij}, Att_i)))\\
&//\text{Type}\\
&\quad \wedge ((\textbf{EqualType}(T1, T2) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, T1, -, -))\\
&\quad \wedge (\textbf{UpperType}(T1, T2) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, T1, -, -))\\
&\quad \wedge (\textbf{LowerType}(T1, T2) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, T2, -, -)))\\
&//\text{Multiplicité maximale}\\
&\quad \wedge ((\textbf{EqualNb}(U1, U2) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, -, U1))\\
&\quad \wedge (\textbf{Greater}(U1, U2) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, -, U1))\\
&\quad \wedge (\textbf{Greater}(U2, U1) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, -, U2)))\\
&//\text{Multiplicité minimale}\\
&\quad \wedge ((\textbf{EqualNb}(L1, L2) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, L1, -))\\
&\quad \wedge (\textbf{Greater}(L1, L2) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, L2, -))\\
&\quad \wedge (\textbf{Greater}(L2, L1) \Rightarrow \textbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, L1, -))))]
\end{aligned}$$

La fusion de deux attributs Att_i et Att_j équivalents appartenant respectivement à C_i et C_j et aux modèles M1 et M2, et ayant respectivement les types T1 et T2, les multiplicités maximales U1 et U2 et les multiplicités minimales L1 et L2 crée dans le modèle résultat MR un attribut $RAtt_{ij}$ appartenant à la classe résultante de C_i et C_j nommée RC_{ij} (cf. Figure 4-35). L'attribut engendré a les propriétés suivantes :

- Nom : prend le nom de l'attribut appartenant au modèle principal, sauf dans le cas de l'inclusion, où il prend le nom le plus long

- Type : prend le plus grand type de T1 et T2
- Multiplicité maximale : garde la plus grande valeur de U1 et U2
- Multiplicité minimale : garde la plus petite valeur de L1 et L2

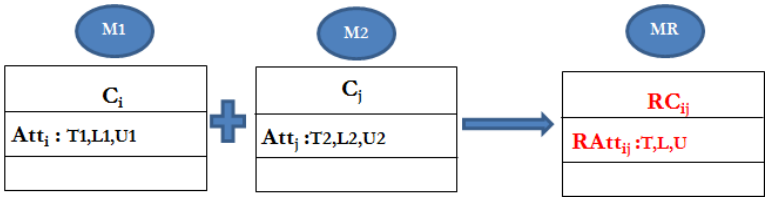


Figure 4-35 Fusion d'attributs

- **Translation d'attributs**

Description informelle

Pour chaque attribut isolé, nous créons dans le modèle résultat un attribut. L'attribut engendré a les propriétés suivantes :

- Nom : garde son nom
- Type : garde son type
- Multiplicité maximale (par défaut 1) : garde sa valeur.
- Multiplicité minimale (par défaut 1) : garde sa valeur.

Exemple : L'attribut « numéro » de la classe « Chéquier » de M1 est translaté en l'élément résultat « numéro » de la classe résultante « Chéquier » (cf. Figure 4-34).

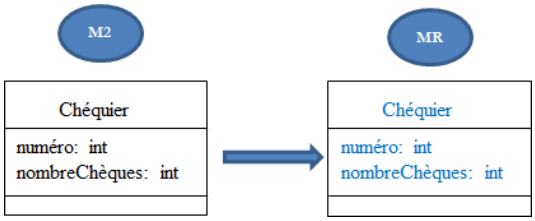


Figure 4-36 Translation d'attribut

Représentation formelle

$$\mathbf{Translation\_Attribute}(\mathbf{Att_i}, \mathbf{RAtt_i}, \mathbf{C_i}, \mathbf{RC_i}, \mathbf{M}, \mathbf{MR}) \Leftrightarrow [(\mathbf{IsolatedAtt}(\mathbf{Att_i}, \mathbf{C_i}, \mathbf{M}) \wedge \mathbf{OwnedAttribute}(\mathbf{Att_i}, \mathbf{C_i}, \mathbf{M}, \mathbf{T}, \mathbf{L}, \mathbf{U}) \wedge \mathbf{ResultElt}(\mathbf{C_i}, \mathbf{RC_i})) \Rightarrow (\mathbf{EqualString}(\mathbf{RAtt_i}, \mathbf{Att_i}) \wedge \mathbf{OwnedAttribute}(\mathbf{RAtt_i}, \mathbf{RC_i}, \mathbf{MR}, \mathbf{T}, \mathbf{L}, \mathbf{U}))]$$

Pour chaque attribut isolé Att_i, appartenant à la classe Ci et au modèle M, ayant pour type T, pour multiplicité maximale U et multiplicité minimale L, un attribut $RAtt_i$ est créé dans le modèle résultat MR appartenant à la classe RC_i (la classe résultante de C_i).

L'attribut engendré a les propriétés suivantes (cf. Figure 4-37) :

- Nom : garde son nom
- Type : garde son type T
- Multiplicité maximale : garde sa valeur U.
- Multiplicité minimale garde sa valeur L.

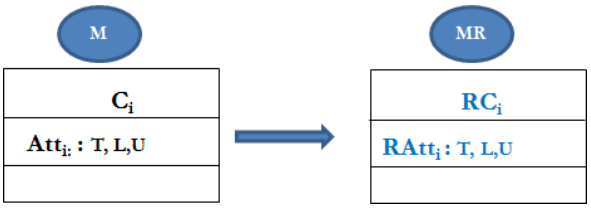


Figure 4-37 Translation d'attributs

- **Fusion d'opérations**

Description informelle

Pour chaque correspondance du mapping d'équivalence de deux opérations, nous créons dans le modèle résultat une opération. L'opération engendrée a les propriétés suivantes :

- Nom : prend le nom de la classe appartenant au modèle principal, sauf dans le cas de l'inclusion, où elle prend le nom le plus long
- Type de retour : prend le plus grand type
- Paramètres : fusion des paramètres équivalents

Exemple : les deux opérations « débiter() » de « Compte » de M1 et « débiter() » de « CompteBancaire » de M2 sont fusionnées en l'opération résultat « débiter » de la classe résultante « Compte » (cf. Figure 4-34).

Représentation formelle

$$\begin{aligned}&\mathbf{Fusion\_Operations}(Op_i, Op_j, ROp_{ij}, C_i, C_j, RC_{ij}, M1, M2, MR)\\&\quad\Leftrightarrow [(\mathbf{Equivalence\_Operations}\ (Op_i, Op_j, C_i, C_i, M1, M2)\\&\quad\wedge \mathbf{OwnedOperation}(Op_i, C_i, M1, T1) \wedge \mathbf{OwnedOperation}(Op_j, C_j, M2, T2)\\&\quad\wedge \mathbf{FusionResultElt}(C_i, C_j, RC_{ij})) \Rightarrow\end{aligned}$$

//Nom

$$\begin{aligned}&(((\mathbf{MainModel}(M1) \wedge \mathbf{BelongingToMainModel}(Op_i, M1)\\&\quad\wedge \neg\mathbf{InclusionSyn}(Op_i, Op_j)) \Rightarrow \mathbf{EqualString}(ROp_{ij}, Op_i))\\&\wedge ((\mathbf{MainModel}(M2) \wedge \mathbf{BelongingToMainModel}(Op_j, M2)\\&\quad\wedge \neg\mathbf{InclusionSyn}(Op_j, Op_i)) \Rightarrow \mathbf{EqualString}(ROp_{ij}, Op_j))\\&\quad\wedge (\mathbf{InclusionSyn}(Op_i, Op_j) \Rightarrow \mathbf{EqualString}(ROp_{ij}, Op_j))\\&\quad\wedge (\mathbf{InclusionSyn}(Op_j, Op_i) \Rightarrow \mathbf{EqualString}(ROp_{ij}, Op_i)))\end{aligned}$$

$$
\begin{aligned}
&//\textbf{Type}\\
&\quad \wedge ((\mathbf{EqualType(T1, T2)} \Rightarrow \mathbf{OwnedOperation}(\mathbf{ROp_{ij}}, \boldsymbol{RC_{ij}}, \boldsymbol{MR}, \mathbf{T1}))\\
&\qquad \wedge (\mathbf{UpperType(T1, T2)} \Rightarrow \mathbf{OwnedOperation}(\mathbf{ROp_{ij}}, \boldsymbol{RC_{ij}}, \boldsymbol{MR}, \mathbf{T1}))\\
&\qquad \wedge (\mathbf{LowerType(T1, T2)} \Rightarrow \mathbf{OwnedOperation}(\mathbf{ROp_{ij}}, \boldsymbol{RC_{ij}}, \boldsymbol{MR}, \mathbf{T2})))\\
&//\textbf{Paramètres}\\
&\wedge ((\forall\, \mathbf{Pm_i\ OwnedParameter}(\mathbf{Pm_i}, \mathbf{Op_i}, \mathbf{C_i}, \mathbf{M1}, -) \wedge \forall\, \mathbf{Pm_j\ OwnedParameter}(\mathbf{Pm_j}, \mathbf{Op_j}, \mathbf{C_j}, \mathbf{M2}, -)\\
&\qquad \wedge \mathbf{Equivalence\_Parameters}(\mathbf{Pm_i}, \mathbf{Pm_j}, \mathbf{OP_i}, \mathbf{OP_j}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{M1}, \mathbf{M2})) \Rightarrow\\
&\quad (\mathbf{Fusion\_Parameters}(\mathbf{Pm_i}, \mathbf{Pm_j}, \mathbf{RPm_{ij}}, \mathbf{Op_i}, \mathbf{Op_j}, \mathbf{ROp_{ij}}, \mathbf{C_i}, \mathbf{C_j}, \mathbf{RC_{ij}}, \mathbf{M1}, \mathbf{M2}, \mathbf{MR})\\
&\qquad \wedge \mathbf{OwnedParameter}(\mathbf{RPm_{ij}}, \mathbf{ROp_{ij}}, \mathbf{RC_{ij}}, \mathbf{MR}, -))))]
\end{aligned}
$$

La fusion de deux opérations équivalentes Op_i et Op_j appartenant respectivement à C_i et C_j et aux modèles M1 et M2, et ayant respectivement les types T1 et T2, crée dans le modèle résultat MR une opération ROp_{ij} appartenant à la classe résultante de C_i et C_j nommée RC_{ij} (cf. Figure 4-38). L'opération engendrée a les propriétés suivantes :

- Nom : prend le nom de la classe appartenant au modèle principal, sauf dans le cas de l'inclusion, où elle prend le nom le plus long
- Type de retour : prend le plus grand type
- Paramètres : fusion des paramètres équivalents

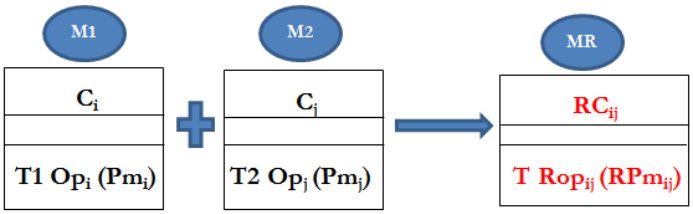


Figure 4-38 Fusion d'opérations

- **Translation d'opération**

Description informelle

Pour chaque opération isolée, nous créons dans le modèle résultat une opération. L'opération engendrée a les propriétés suivantes :

- Nom : garde son nom
- Type de retour : garde son type
- Paramètres : garde ses paramètres

Exemple : L'opération « créer() » de la classe « Compte » de M1 est translatée en l'élément résultat « créer()» de la classe résultante « Compte » (cf. Figure 4-34).

Représentation formelle

$$\begin{aligned}&\mathbf{Translation\_Operation}(Op_i, ROp_i, C_i, RC_i, M, MR)\\&\Leftrightarrow [(\mathbf{IsolatedOp}(Op_i, C_i, M) \wedge \mathbf{OwnedOperation}(Op_i, C_i, M, T)\\&\wedge \mathbf{ResultElt}(C_i, RC_i)) \Rightarrow (\mathbf{EqualString}(ROp_i, Op_i) \wedge \mathbf{OwnedOperation}(ROp_i, RC_i, MR, T)\\&\wedge (\forall\, Pm_i\ \mathbf{OwnedParameter}(Pm_i, Op_i, C_i, M, -) \Rightarrow (\mathbf{Translation\_Parameters}(Pm_i, RPm_i, OP_i, ROP_i, C_i, RC_i, M, MR)\\&\wedge \mathbf{OwnedParameter}(RPm_i, ROp_i, RC_i, MR, -)))))]\end{aligned}$$

Pour chaque opération isolée Op_i appartenant à la classe C_i, au modèle M et ayant pour type T, une opération ROp_i est créée dans le modèle résultat MR appartenant à la classe RC_i (la classe résultante de C_i). L'opération engendrée a les propriétés suivantes (cf. Figure 4-39) :

- Nom : garde son nom
- Type de retour : garde don type T
- Paramètres : garde ses paramètres

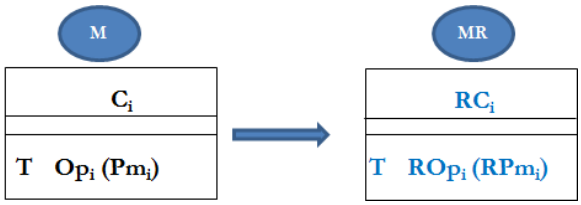


Figure 4-39 Translation d'opération

- **Fusion de paramètres**

Description informelle

Pour chaque correspondance du mapping d'équivalence de deux paramètres, nous créons, dans le modèle résultat, un paramètre. Le paramètre engendré a les propriétés suivantes :

- Nom : prend le nom du paramètre appartenant au modèle principal, sauf dans le cas de l'inclusion, où il prend le nom le plus long
- Type : prend le plus grand type

Exemple : « solde » de l'opération « débiter() » de la classe « Compte » de M1 et le paramètre l'opération « solde » de l'opération « débiter() » de la classe « CompteBancaire » de M2 sont fusionnés en paramètre résultat « solde » de l'opération « débiter() » de la classe résultante « Compte » (cf. Figure 4-34).

Représentation formelle

$$\begin{aligned}&\mathbf{Fusion\_Parameters}(Pm_i, Pm_j, RPm_{ij}, Op_i, Op_j, ROp_{ij}, C_i, C_j, RC_{ij}, M1, M2, MR)\\&\Leftrightarrow [(\mathbf{Equivalence\_Parameters}(Pm_i, Pm_j, OP_i, OP_j, C_i, C_j, M1, M2)\\&\wedge \mathbf{OwnedParameter}(Pm_i, OP_i, C_i, M1, T1)\\&\wedge \mathbf{OwnedParameter}(Pm_j, OP_j, C_j, M2, T2) \wedge \mathbf{FusionResultElt}(C_i, C_j, RC_{ij})\\&\wedge \mathbf{FusionResultElt}(Op_i, Op_j, ROp_{ij})) \Rightarrow\end{aligned}$$

//Nom

$$\begin{aligned}&((((\mathbf{MainModel}(M1) \wedge \mathbf{BelongingToMainModel}(\mathbf{Pm_i}, M1)\\&\quad \wedge \neg\mathbf{InclusionSyn}(\mathbf{Pm_i}, \mathbf{Pm_j})) \Rightarrow \mathbf{EqualString}(\mathbf{RPm_{ij}}, \mathbf{Pm_i}))\\&\wedge ((\mathbf{MainModel}(M2) \wedge \mathbf{BelongingToMainModel}(\mathbf{Pm_j}, M2)\\&\quad \wedge \neg\mathbf{InclusionSyn}(\mathbf{Pm_j}, \mathbf{Pm_i})) \Rightarrow \mathbf{EqualString}(\mathbf{RPm_{ij}}, \mathbf{Pm_j}))\\&\quad \wedge (\mathbf{InclusionSyn}(\mathbf{Pm_i}, \mathbf{Pm_j}) \Rightarrow \mathbf{EqualString}(\mathbf{RPm_{ij}}, \mathbf{Pm_j}))\\&\quad \wedge (\mathbf{InclusionSyn}(\mathbf{Pm_j}, \mathbf{Pm_i}) \Rightarrow \mathbf{EqualString}(\mathbf{RPm_{ij}}, \mathbf{Pm_i})))\end{aligned}$$

//Type

$$\begin{aligned}&\wedge ((\mathbf{EqualType}(\mathbf{T1}, \mathbf{T2}) \Rightarrow \mathbf{OwnedParameter}(\mathbf{RPm_{ij}}, \mathbf{ROp_{ij}}, \mathbf{RC_{ij}}, \mathbf{MR}, \mathbf{T1}))\\&\quad \wedge (\mathbf{UpperType}(\mathbf{T1}, \mathbf{T2}) \Rightarrow \mathbf{OwnedParameter}(\mathbf{RPm_{ij}}, \mathbf{ROp_{ij}}, \mathbf{RC_{ij}}, \mathbf{MR}, \mathbf{T1}))\\&\quad \wedge (\mathbf{LowerType}(\mathbf{T1}, \mathbf{T2}) \Rightarrow \mathbf{OwnedParameter}(\mathbf{RPm_{ij}}, \mathbf{ROp_{ij}}, \mathbf{RC_{ij}}, \mathbf{MR}, \mathbf{T2}))))]\end{aligned}$$

La fusion de deux paramètres équivalents Pm_i et Pm_j appartenant respectivement aux opérations Op_i et Op_j, aux classes C_i et C_j et aux modèles M1 et M2, et ayant respectivement les types T1 et T2, crée, dans le modèle résultat MR, un paramètre RPm_{ij} appartenant à l'opération ROp_{ij} (L'opération résultante de Op_i et Op_j) et la classe RC_{ij} (résultante de C_i et C_j). Le paramètre engendré a les propriétés suivantes (cf. Figure 4-40) :

- Nom : prend le nom du paramètre appartenant au modèle principal, sauf dans le cas de l'inclusion, où il prend le nom le plus long
- Type : prend le plus grand type

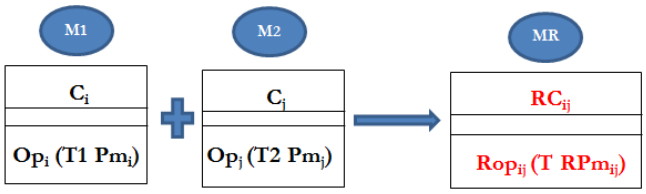


Figure 4-40 Fusion de paramètres

- **Translation de paramètres**

Description informelle

Pour chaque paramètre isolé, nous créons dans le modèle résultat un paramètre. Le paramètre engendré a les propriétés suivantes :

- Nom : garde son nom
- Type : garde son type

Représentation formelle

$$\begin{aligned}&\mathbf{Translation\_Parameter}(\mathbf{Pm_i}, \mathbf{RPm_i}, \mathbf{OP_i}, \mathbf{ROP_i}, \mathbf{C_i}, \mathbf{RC_i}, \mathbf{M}, \mathbf{MR})\\&\quad \Leftrightarrow [(\mathbf{IsolatedPm}(\mathbf{Pm_i}, \mathbf{Op_i}, \mathbf{C_i}, \mathbf{M}) \wedge \mathbf{OwnedParameter}(\mathbf{Pm_i}, \mathbf{Op_i}, \mathbf{C_i}, \mathbf{M}, \mathbf{T})\\&\quad \wedge \mathbf{ResultElt}(\mathbf{C_i}, \mathbf{RC_i}) \wedge \mathbf{ResultElt}(\mathbf{Op_i}, \mathbf{ROp_i})) \Rightarrow (\mathbf{EqualString}(\mathbf{RPm_i}, \mathbf{Pm_i})\\&\quad \wedge \mathbf{OwnedParameter}(\mathbf{RPm_i}, \mathbf{ROp_i}, \mathbf{RC_i}, \mathbf{MR}, \mathbf{T}))]\end{aligned}$$

Pour chaque paramètre isolé Pm_i appartenant à l'opération Op_i, à la classe C_i et au modèle M, et ayant pour type T, un paramètre RPm_i est créé dans le modèle résultat MR appartenant à l'opération ROp_i (résultante de Op_i) et à la classe RC_i (résultante de Ci). Le paramètre engendré a les propriétés suivantes (cf. Figure 4-41) :

- Nom : garde son nom
- Type : garde son type T

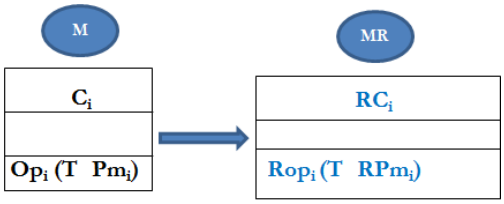


Figure 4-41 Translation de paramètres

- **Fusion d'associations**

Description informelle

Pour chaque correspondance du mapping d'équivalence de deux associations, nous créons, dans le modèle résultat, une association. L'association engendrée a les propriétés suivantes :

- Nom : prend le nom de l'association appartenant au modèle principal, sauf le cas de l'inclusion, où elle prend le nom le plus long
- Composition : sera une composition si les deux associations de départ sont des compositions, sinon elle sera une association
- Navigabilité : sera navigable si les deux associations de départ sont navigables, sinon elle sera non navigables.
- Multiplicités maximales : garde les plus grandes valeurs.
- Multiplicités minimales : garde les plus petites valeurs.
- Noms de rôles : garde les noms de rôles du modèle principal.
- Classes qu'elle relie : voir les deux situations 1 et 2

Situation 1 : (cf. Figure 4-42) « Compte » et « CompteBancaire » sont deux classes équivalentes, « CarteCrédit » et « Carte » sont deux classes équivalentes, l'association « Joint » (M1) reliant « Compte » à « CarteCrédit » est équivalente à l'association «Joint » (M2) reliant « CompteBancaire » à « Carte ». Par conséquent, l'association résultat des deux associations relie la classe résultat « CompteBancaire » à la classe résultat « CarteCrédit ».

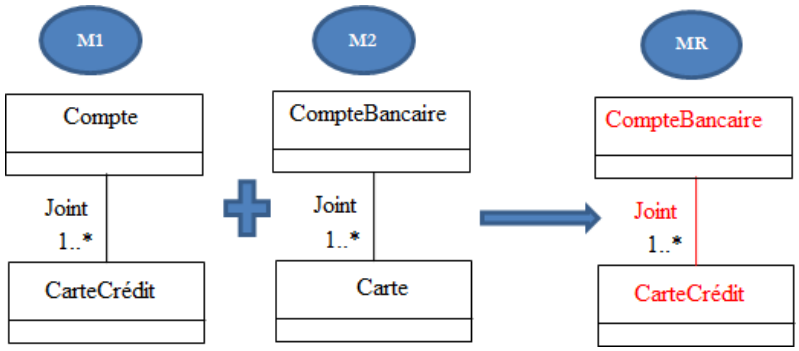


Figure 4-42 Fusion d'association : situation 1

Situation 2 : (cf. Figure 4-43) « Compte » de M1 et « CompteBancaire » de M2 sont deux classes équivalentes, « Client » de M1 et « Client » de M2 sont deux classes équivalentes, « Client » est une sous-classe de « Personne », l'association « Posséder » de M1 reliant « Compte » et « Client » est équivalente à l'association « Avoir » de M2 reliant « Personne » et « CompteBancaire ». Dans ce cas, l'association résultat des deux associations relie la classe résultat « CompteBancaire » à la classe super-classe résultat « Personne ».

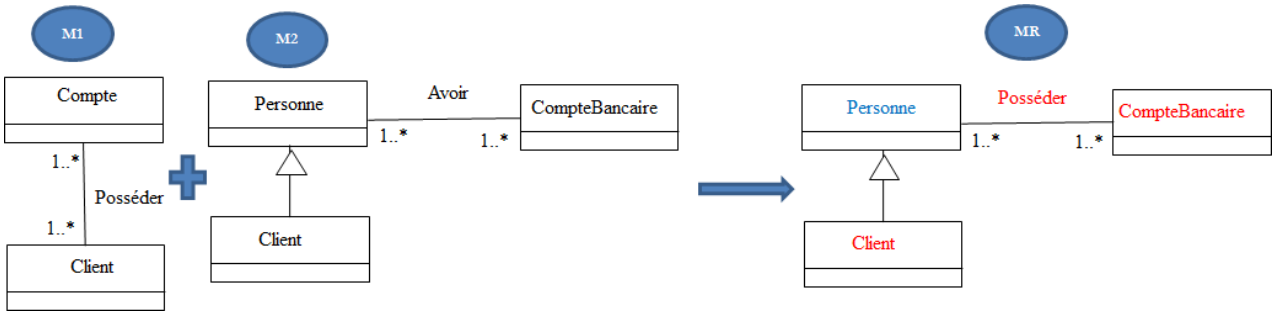


Figure 4-43 Fusion d'associations : situation 2

Représentation formelle

$$\mathbf{Fusion\_Associations}(Ass_i, Ass_j, RAss_{ij}, M1, M2, MR) \Leftrightarrow [\mathbf{Equivalence\_Associations}(Ass_i, Ass_j) \Rightarrow$$

//Nom

$$((((\mathbf{MainModel}(M1) \wedge \mathbf{BelongingToMainModel}(Ass_i, M1) \wedge \neg \mathbf{InclusionSyn}(Ass_i, Ass_j)) \Rightarrow \mathbf{EqualString}(RAss_{ij}, Ass_i)) \wedge ((\mathbf{MainModel}(M2) \wedge \mathbf{BelongingToMainModel}(Ass_j, M2) \wedge \neg \mathbf{InclusionSyn}(Ass_j, Ass_i)) \Rightarrow \mathbf{EqualString}(RAss_{ij}, Ass_j)) \wedge (\mathbf{InclusionSyn}(Ass_i, Ass_j) \Rightarrow \mathbf{EqualString}(RAss_{ij}, Ass_j)) \wedge (\mathbf{InclusionSyn}(Ass_j, Ass_i) \Rightarrow \mathbf{EqualString}(RAss_{ij}, Ass_i))) \wedge$$

//Situation 1

$$[(\exists C_i, C_j, C_k, C_l, P_i, P_j, P_k, P_l\ (\mathbf{Class}(C_i, M1) \wedge \mathbf{Class}(C_j, M2) \wedge \mathbf{Class}(C_k, M1) \wedge \mathbf{Class}(C_l, M2) \wedge \mathbf{OwnedAttribute}(P_i, C_k, M1, C_i, L1, U1) \wedge \mathbf{OwnedAttribute}(P_k, C_i, M1, C_k, L2, U2) \wedge \mathbf{memberEnd}(Ass_i, P_i, -, -) \wedge \mathbf{memberEnd}(Ass_i, P_k, -, -) \wedge$$

$$\mathbf{OwnedAttribute}(P_j, C_l, M2, C_j, L3, U3) \wedge \mathbf{OwnedAttribute}(P_l, C_j, M2, C_l, L4, U4) \wedge$$
$$\mathbf{memberEnd}(Ass_j, P_j, -, -) \wedge \mathbf{memberEnd}(Ass_j, P_l, -, -)) \wedge$$
$$\mathbf{Equivalence\_Classes}(C_i, C_j, M1, M2, -) \wedge \mathbf{Equivalence\_Classes}(C_k, C_l, M1, M2, -) \wedge$$
$$\exists RC_{ij}, RC_{kl}, RP_{ij}, RP_{kl}(\mathbf{FusionResultElt}(C_i, C_j, RC_{ij}) \wedge \mathbf{FusionResultElt}(C_k, C_l, RC_{kl}) \wedge$$
$$\mathbf{FusionResultElt}(P_i, P_j, RP_{ij}) \wedge \mathbf{FusionResultElt}(P_k, P_l, RP_{kl}))) \Rightarrow$$

//Les classes qu'elle relie
$$((\mathbf{OwnedAttribute}(RP_{ij}, RC_{kl}, MR, RC_{ij}, -, -) \wedge \mathbf{OwnedAttribute}(RP_{kl}, RC_{ij}, MR, RC_{kl}, -, -) \wedge$$
$$\mathbf{memberEnd}(RAss_{ij}, RP_{ij}, -, -) \wedge \mathbf{memberEnd}(RAss_{ij}, RP_{kl}, -, -)) \wedge$$
//Multiplicités

//Multiplicités maximale
$$\wedge ((\mathbf{EqualNb}(U1, U3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_{kl}, MR, RC_{ij}, -, U1))$$
$$\wedge (\mathbf{Greater}(U1, U3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_{kl}, MR, RC_{ij}, -, U1))$$
$$\wedge (\mathbf{Greater}(U3, U1) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_{kl}, MR, RC_{ij}, -, U3)))$$

$$\wedge ((\mathbf{EqualNb}(U2, U4) \Rightarrow \mathbf{OwnedAttribute}(RP_{kl}, RC_{ij}, MR, RC_{kl}, -, U2))$$
$$\wedge (\mathbf{Greater}(U2, U4) \Rightarrow \mathbf{OwnedAttribute}(RP_{kl}, RC_{ij}, MR, RC_{kl}, -, U2))$$
$$\wedge (\mathbf{Greater}(U4, U2) \Rightarrow \mathbf{OwnedAttribute}(RP_{kl}, RC_{ij}, MR, RC_{kl}, -, U4)))$$

//Multiplicités minimale

$$\wedge ((\mathbf{EqualNb}(L1, L3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_{kl}, MR, RC_{ij}, L1, -))$$
$$\wedge (\mathbf{Greater}(L1, L3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_{kl}, MR, RC_{ij}, L3, -))$$
$$\wedge (\mathbf{Greater}(L3, L1) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_{kl}, MR, RC_{ij}, L1, -)))$$

$$\wedge ((\mathbf{EqualNb}(L2, L4) \Rightarrow \mathbf{OwnedAttribute}(RP_{kl}, RC_{ij}, MR, RC_{kl}, L2, -))$$
$$\wedge (\mathbf{Greater}(L2, L4) \Rightarrow \mathbf{OwnedAttribute}(RP_{kl}, RC_{ij}, MR, RC_{kl}, L4, -))$$
$$\wedge (\mathbf{Greater}(L4, L2) \Rightarrow \mathbf{OwnedAttribute}(RP_{kl}, RC_{ij}, MR, RC_{kl}, L2, -)))$$

//Navigabilité
$$\wedge ((\mathbf{memberEnd}(Ass_i, P_i, -, true) \wedge \mathbf{memberEnd}(Ass_j, P_j, -, true)) \Rightarrow \mathbf{memberEnd}(RAss_{ij}, RP_{ij}, -, \boldsymbol{true}))$$
$$\wedge (\mathbf{memberEnd}(Ass_i, P_k, -, true)$$
$$\wedge \mathbf{memberEnd}(Ass_j, P_l, -, true)) \Rightarrow \mathbf{memberEnd}(RAss_{ij}, RP_{kl}, -, \boldsymbol{true})))$$

//Composition

$$\wedge (((\mathbf{memberEnd}(Ass_i, P_i, true, -) \wedge \mathbf{memberEnd}(Ass_j, P_j, true, -)) \Rightarrow \mathbf{memberEnd}(RAss_{ij}, RP_{ij}, true, -))$$
$$\wedge (\mathbf{memberEnd}(Ass_i, P_k, true, -)$$
$$\wedge \mathbf{memberEnd}(Ass_j, P_l, true, -)) \Rightarrow \mathbf{memberEnd}(RAss_{ij}, RP_{kl}, true, -)))$$
//Noms des extrémités d'association

$$((\mathbf{MainModel}(M1) \wedge \mathbf{BelongingToMainModel}(P_i, M1) \wedge \neg \mathbf{InclusionSyn}(P_i, P_j)) \Rightarrow \mathbf{EqualString}(RP_{ij}, P_i))$$
$$\wedge ((\mathbf{MainModel}(M2) \wedge \mathbf{BelongingToMainModel}(P_j, M2) \wedge \neg \mathbf{InclusionSyn}(P_j, P_i)) \Rightarrow \mathbf{EqualString}(RP_{ij}, P_j))$$
$$\wedge (\mathbf{InclusionSyn}(P_i, P_j) \Rightarrow \mathbf{EqualString}(RP_{ij}, P_j))$$

$$\wedge\ (\mathbf{InclusionSyn}(P_j, P_i) \Rightarrow \mathbf{EqualString}(RP_{ij}, P_i))$$

$$\wedge\ ((\mathbf{MainModel}(M1) \wedge \mathbf{BelongingToMainModel}(P_k, M1) \wedge \neg\mathbf{InclusionSyn}(P_k, P_l)) \Rightarrow \mathbf{EqualString}(RP_{kl}, P_k))$$

$$\wedge\ ((\mathbf{MainModel}(M2) \wedge \mathbf{BelongingToMainModel}(P_l, M2) \wedge \neg\mathbf{InclusionSyn}(P_l, P_k)) \Rightarrow \mathbf{EqualString}(RP_{kl}, P_l))$$

$$\wedge\ (\mathbf{InclusionSyn}(P_k, P_l) \Rightarrow \mathbf{EqualString}(RP_{kl}, P_l))$$

$$\wedge\ (\mathbf{InclusionSyn}(P_l, P_k) \Rightarrow \mathbf{EqualString}(RP_{kl}, P_k))\big] \wedge$$

//Situation 2

$$\big[(\exists C_i, C_j, C_k, C_m, C_l, P_i, P_j, P_k, P_m\ (\mathbf{Class}(C_i, M1) \wedge \mathbf{Class}(C_j, M2) \wedge \mathbf{Class}(C_k, M1) \wedge \mathbf{Class}(C_l, M2) \wedge \mathbf{Class}(C_m, M2) \wedge \mathbf{superClass}(C_m, C_l, M2) \wedge \mathbf{OwnedAttribute}(P_i, C_k, M1, C_i, L1, U1) \wedge \mathbf{OwnedAttribute}(P_k, C_i, M1, C_k, L2, U2) \wedge \mathbf{memberEnd}(Ass_i, P_i, -, -) \wedge \mathbf{memberEnd}(Ass_i, P_k, -, -) \wedge \mathbf{OwnedAttribute}(P_j, C_m, M2, C_j, L3, U3) \wedge \mathbf{OwnedAttribute}(P_m, C_j, M2, C_m, L4, U4) \wedge \mathbf{memberEnd}(Ass_j, P_j, -, -) \wedge \mathbf{memberEnd}(Ass_j, P_m, -, -)) \wedge \mathbf{Equivalence\_Classes}(C_i, C_j, M1, M2, -) \wedge \mathbf{Equivalence\_Classes}(C_k, C_l, M1, M2, -) \wedge \exists RC_{ij}, RC_{kl}, RC_m, RP_{ij}, RP_m(\mathbf{FusionResultElt}(C_i, C_j, RC_{ij}) \wedge \mathbf{FusionResultElt}(C_k, C_l, RC_{kl}) \wedge \mathbf{ResultElt}(C_m, RC_m) \wedge \mathbf{FusionResultElt}(P_i, P_j, RP_{ij}) \wedge \mathbf{ResultElt}(P_m, RP_m))) \Rightarrow$$

//Les classes qu'elle relie

$$((\mathbf{OwnedAttribute}(RP_{ij}, RC_m, MR, RC_{ij}, -, -) \wedge \mathbf{OwnedAttribute}(RP_m, RC_{ij}, MR, RC_m, -, -) \wedge \mathbf{memberEnd}(RAss_{ij}, RP_{ij}, -, -) \wedge \mathbf{memberEnd}(RAss_{ij}, RP_m, -, -)) \wedge$$

//Multiplicité

//Multiplicité maximale

$$\wedge\ ((\mathbf{EqualNb}(U1, U3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_m, MR, RC_{ij}, -, U1)) \wedge (\mathbf{Greater}(U1, U3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_m, MR, RC_{ij}, -, U1)) \wedge (\mathbf{Greater}(U3, U1) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_m, MR, RC_{ij}, -, U3)))$$

$$\wedge\ ((\mathbf{EqualNb}(U2, U4) \Rightarrow \mathbf{OwnedAttribute}(RP_m, RC_{ij}, MR, RC_m, -, U2)) \wedge (\mathbf{Greater}(U2, U4) \Rightarrow \mathbf{OwnedAttribute}(RP_m, RC_{ij}, MR, RC_m, -, U2)) \wedge (\mathbf{Greater}(U4, U2) \Rightarrow \mathbf{OwnedAttribute}(RP_m, RC_{ij}, MR, RC_m, -, U4)))$$

//Multiplicité minimale

$$\wedge\ ((\mathbf{EqualNb}(L1, L3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_m, MR, RC_{ij}, L1, -)) \wedge (\mathbf{Greater}(L1, L3) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_m, MR, RC_{ij}, L3, -)) \wedge (\mathbf{Greater}(L3, L1) \Rightarrow \mathbf{OwnedAttribute}(RP_{ij}, RC_m, MR, RC_{ij}, L1, -)))$$

$$\wedge\ ((\mathbf{EqualNb}(L2, L4) \Rightarrow \mathbf{OwnedAttribute}(RP_m, RC_{ij}, MR, RC_m, L2, -)) \wedge (\mathbf{Greater}(L2, L4) \Rightarrow \mathbf{OwnedAttribute}(RP_m, RC_{ij}, MR, RC_m, L4, -)) \wedge (\mathbf{Greater}(L4, L2) \Rightarrow \mathbf{OwnedAttribute}(RP_m, RC_{ij}, MR, RC_m, L2, -)))$$

//Navigabilité

$$\wedge\ (((\mathbf{memberEnd}(Ass_i, P_i, -, \mathrm{true}) \wedge \mathbf{memberEnd}(Ass_j, P_j, -, \mathrm{true})) \Rightarrow \mathbf{memberEnd}(RAss_{ij}, RP_{ij}, -, \boldsymbol{true}) \wedge ((\mathbf{memberEnd}(Ass_i, P_k, -, \mathrm{true}) \wedge \mathbf{memberEnd}(Ass_j, P_m, -, \mathrm{true})) \Rightarrow \mathbf{memberEnd}(RAss_{ij}, RP_m, -, \boldsymbol{true})))$$

//Composition

$$\wedge\ (((\mathbf{memberEnd}(\mathbf{Ass_i}, \mathbf{P_i}, \mathbf{true}, -) \wedge \mathbf{memberEnd}(\mathbf{Ass_j}, \mathbf{P_j}, \mathbf{true}, -)) \Rightarrow \mathbf{memberEnd}(\mathbf{RAss_{ij}}, \mathbf{RP_{ij}}, \mathbf{true}, -))$$
$$\wedge\ ((\mathbf{memberEnd}(\mathbf{Ass_i}, \mathbf{P_k}, \mathbf{true}, -)$$
$$\wedge\ \mathbf{memberEnd}(\mathbf{Ass_j}, \mathbf{P_m}, \mathbf{true}, -)) \Rightarrow \mathbf{memberEnd}(\mathbf{RAss_{ij}}, \mathbf{RP_m}, \mathbf{true}, -)))$$

// Noms des extrémités d'association

$$(((\mathbf{MainModel(M1)} \wedge \mathbf{BelongingToMainModel}(\mathbf{P_i}, \mathbf{M1}) \wedge \neg\mathbf{InclusionSyn}(\mathbf{P_i}, \mathbf{P_j})) \Rightarrow \mathbf{EqualString}(\mathbf{RP_{ij}}, \mathbf{P_i}))$$
$$\wedge\ ((\mathbf{MainModel(M2)} \wedge \mathbf{BelongingToMainModel}(\mathbf{P_j}, \mathbf{M2}) \wedge \neg\mathbf{InclusionSyn}(\mathbf{P_j}, \mathbf{P_i})) \Rightarrow \mathbf{EqualString}(\mathbf{RP_{ij}}, \mathbf{P_j}))$$
$$\wedge\ (\mathbf{InclusionSyn}(\mathbf{P_i}, \mathbf{P_j}) \Rightarrow \mathbf{EqualString}(\mathbf{RP_{ij}}, \mathbf{P_j}))$$
$$\wedge\ (\mathbf{InclusionSyn}(\mathbf{P_j}, \mathbf{P_i}) \Rightarrow \mathbf{EqualString}(\mathbf{RP_{ij}}, \mathbf{P_i}))$$
$$\wedge\ \mathbf{EqualString}(\mathbf{RP_m}, \mathbf{P_m}))]]$$

La fusion de deux associations Ass_i et Ass_j, ayant respectivement les multiplicités minimales L1, L2 et L3, L4 et les multiplicités maximales U1,U2 et U3,U4, crée une association résultat $RAss_{ij}$, dans le modèle résultat MR. L'association engendrée a les propriétés suivantes :

- Nom : prend le nom de l'association appartenant au modèle principal, sauf dans le cas de l'inclusion, où elle prend le nom le plus long
- Composition : sera une composition si les deux associations de départ sont des compositions, sinon elle sera une association
- Navigabilité : sera navigable si les deux associations de départ sont navigables, sinon elle sera non navigable
- Multiplicités maximales : garde les plus grandes valeurs
- Multiplicités minimales : garde les plus petites valeurs
- **Situation 1** (cf. Figure 4-44) : Ass_i relie les classes C_i (du côté de l'extrémité d'association P_i) et C_k (du côté de l'extrémité d'association P_k) et Ass_j relie C_j (du côté de l'extrémité d'association P_j) et C_l(du côté de l'extrémité d'association P_l)
  - Les classes qu'elle relie : RC_{ij} (résultante de C_i et C_j) et RC_{kl} (résultante de C_k et C_l)
  - Noms des extrémités d'association : les noms du modèle principal, sauf dans le cas de l'inclusion, où elle prend le nom le plus long
- **Situation 2** (cf. Figure 4-45) : Ass_i relie les classes C_i (du côté de l'extrémité d'association P_i) et C_k (du côté de l'extrémité d'association P_k) et Ass_j relie C_j (du côté de l'extrémité d'association P_j) et C_m (du côté de l'extrémité d'association P_m), une super-classe de C_l, avec C_k et C_l des classes équivalentes et C_i et C_j des classes équivalentes.
  - Les classes qu'elle relie : RC_{ij} (résultante de C_i et C_j) et RC_m (résultante de C_m)
  - Noms des extrémités d'association :
    - Du côté du RC_{ij} : le nom du modèle principal, sauf dans le cas de l'inclusion, où elle prend le nom le plus long
    - Du côté du RC_m : le nom de RP_m

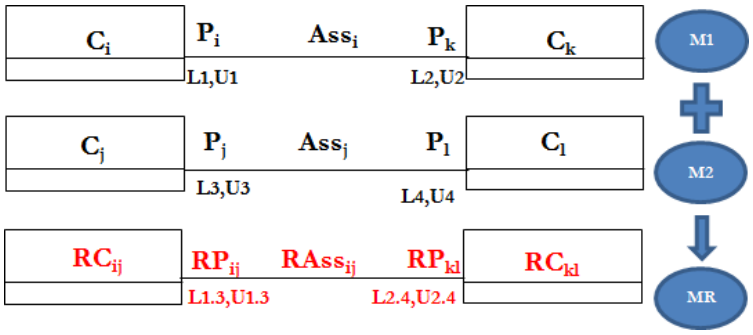


Figure 4-44 Fusion d'association : Situation 1

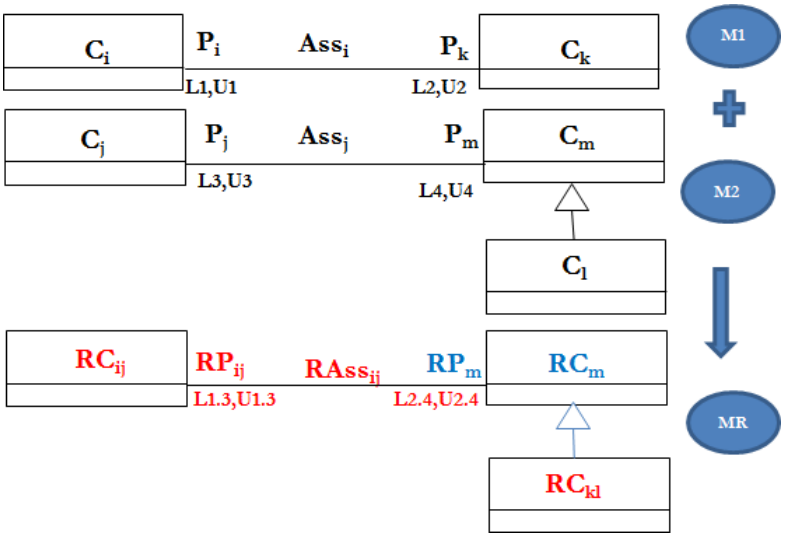


Figure 4-45 Fusion d'association : Situation 2

- **Translation d'association**

Description informelle

Pour chaque association isolée, nous créons dans le modèle résultat une association (cf. Figure 4-46). L'association engendrée a les propriétés suivantes :

- Nom : garde son nom
- Composition : garde sa nature (composition ou association)
- Navigabilité : garde sa navigabilité
- Multiplicités : garde ses multiplicités
- Noms de rôles : garde ses noms de rôles
- Classes qu'elles relient : les classes résultantes (suite à une fusion ou translation) des deux classes qu'elle relie.

Exemple : L'association isolée « Joint » de M2 est translatée dans MR reliant les classes « CompteCourant » et « Chéquier »

Figure 4-46 Translation d'association

Représentation formelle

$$\textbf{Translation\_Associations}(\textbf{Ass}_i, \textbf{RAss}_i, \textbf{M}, \textbf{MR}) \Leftrightarrow [[\exists \textbf{C}_i, \textbf{C}_j, \textbf{P}_i, \textbf{P}_j\ (\textbf{Class}(\textbf{C}_i, \textbf{M}) \wedge \textbf{Class}(\textbf{C}_j, \textbf{M}) \wedge \textbf{OwnedAttribute}(\textbf{P}_i, \textbf{C}_j, \textbf{M1}, \textbf{C}_i, \textbf{L1}, \textbf{U1}) \wedge \textbf{OwnedAttribute}(\textbf{P}_j, \textbf{C}_i, \textbf{M1}, \textbf{C}_i, \textbf{L2}, \textbf{U2}) \wedge \textbf{memberEnd}(\textbf{Ass}_i, \textbf{P}_i, \textbf{Cp1}, \textbf{N1}) \wedge \textbf{memberEnd}(\textbf{Ass}_i, \textbf{P}_j, \textbf{Cp2}, \textbf{N2}) \wedge \textbf{IsolatedAss}(\textbf{P}_i, \textbf{Ass}_i, \textbf{C}_i, \textbf{M}) \wedge \textbf{IsolatedAss}(\textbf{P}_j, \textbf{Ass}_i, \textbf{C}_j, \textbf{M})) \wedge \exists \textbf{RC}_i, \textbf{RC}_j, \textbf{RP}_i, \textbf{RP}_j (\textbf{ResultElt}(\textbf{C}_i, \textbf{RC}_i) \wedge \textbf{ResultElt}(\textbf{C}_j, \textbf{RC}_j) \wedge \textbf{ResultElt}(\textbf{P}_i, \textbf{RP}_i) \wedge \textbf{ResultElt}(\textbf{P}_j, \textbf{RP}_j))] \Rightarrow$$

//Nom
$$[\textbf{EqualString}(\textbf{RAss}_i, \textbf{Ass}_i)$$

//Les classes qu'elle relie, Multiplicités, navigabilité, composition
$$(\textbf{OwnedAttribute}(\textbf{RP}_i, \textbf{RC}_j, \textbf{MR}, \textbf{RC}_i, \textbf{L1}, \textbf{U1}) \wedge \textbf{OwnedAttribute}(\textbf{RP}_j, \textbf{RC}_i, \textbf{MR}, \textbf{RC}_j, \textbf{L2}, \textbf{U2}) \wedge \textbf{memberEnd}(\textbf{RAss}_i, \textbf{RP}_i, \textbf{Cp1}, \textbf{N1}) \wedge \textbf{memberEnd}(\textbf{RAss}_i, \textbf{RP}_j, \textbf{Cp2}, \textbf{N2})) \wedge$$

//Noms de rôles
$$\textbf{EqualString}(\textbf{RP}_i, \textbf{P}_i) \wedge \textbf{EqualString}(\textbf{RP}_j, \textbf{P}_j)]]$$

Pour toute association isolée Ass_i appartenant au modèle M reliant les classes C_i et C_j, ayant les extrémités d'association P_i du côté de C_i et P_j du côté de C_j, avec P_i et P_j ayantt respectivement les multiplicités minimales L1 et L2, les multiplicités maximales U1 et U2, les navigabilités N1 et N2 et les compositions Cp1 et Cp2, une association $RAss_i$ est créée dans le modèle résultat MR (cf. Figure 4-47).

L'association engendrée a les propriétés suivantes :

- Nom : garde son nom
- Composition : garde sa nature (composition ou association)
- Navigabilité : garde sa navigabilité
- Multiplicités : garde ses multiplicités
- Noms de rôles : garde ses noms de rôles
- Classes qu'elles relient : les classes RC_i (résultant de C_i) et RC_j (résultant de C_j)

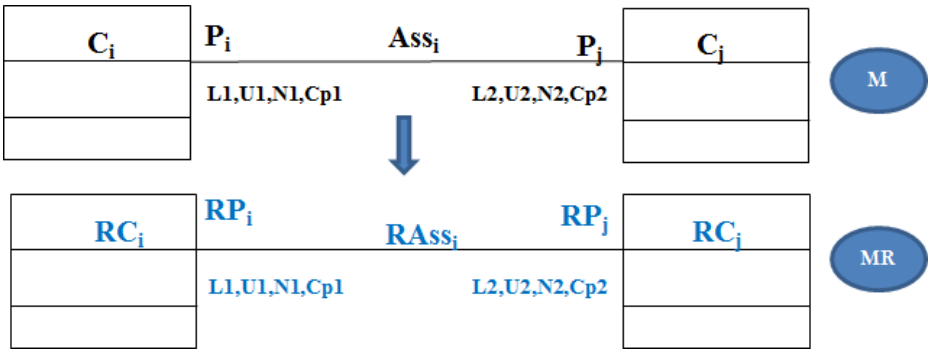


Figure 4-47 Translation d'association

- **Fusion de relations de généralisation**

Description informelle

Pour chaque correspondance du mapping d'équivalence de deux relations de généralisation, nous appliquons les étapes suivantes :

- Créer dans le modèle résultat une relation de généralisation. La relation de généralisation engendrée a les propriétés suivantes :
  - Super-classe : la classe résultante des deux super classes équivalentes
  - Sous-classe : la classe résultante des deux sous classes équivalentes
- Supprimer les doublons (attributs, opération, associations, relation de généralisation) entre la sous-classe et la super-classe résultantes. C'est le même cas de suppression que lors de la translation des classes hyponymes.

Représentation formelle

$$\mathbf{Fusion\_Gen}(C_i, C_j, C_m, C_n, M1, M2, MR)$$
$$\Leftrightarrow [(\exists C_i, C_j, C_m, C_n (\mathbf{Class}(C_i, M1) \wedge \mathbf{Class}(C_j, M2) \wedge \mathbf{Class}(C_m, M1) \wedge \mathbf{Class}(C_n, M2)$$
$$\wedge \mathbf{Equivalence\_Gen}(C_i, C_j, C_m, C_n, M1, M2)) \wedge \exists RC_{mn}, RC_{ij} (\mathbf{Class}(RC_{mn}, MR)$$
$$\wedge \mathbf{Class}(RC_{ij}, MR) \wedge \mathbf{FusionResultElt}(C_i, C_j, RC_{ij})$$
$$\wedge \mathbf{FusionResultElt}(C_m, C_n, RC_{mn}))) \Rightarrow (\mathbf{superClass}(RC_{mn}, RC_{ij}) \wedge$$

//Eliminer les attributs en double

$$[(\exists\, RAtt_{ij}\ \mathbf{OwnedAttribute}(RAtt_{ij}, RC_{ij}, MR, -, -, -)$$
$$\wedge \exists\, RAtt_{mn}\ \mathbf{OwnedAttribute}(RAtt_{mn}, RC_{mn}, MR, -, -, -)$$
$$\wedge \mathbf{Equivalence\_Attributes}(RAtt_{ij}, RAtt_{mn}, RC_{ij}, RC_{mn}, MR, MR)) \Rightarrow \mathbf{RemoveAtt}(RAtt_{ij}, RC_{ij}, MR)] \wedge$$

//Eliminer les opérations en double

$$[(\exists\, ROp_{ij}\ \mathbf{OwnedOperation}(ROp_{ij}, RC_{ij}, MR, -) \wedge \exists\, ROP_{mn}\ \mathbf{OwnedOperation}(ROp_{mn}, RC_{mn}, MR, -)$$
$$\wedge \mathbf{Equivalence\_Operations}(ROp_{ij}, ROp_{mn}, RC_{ij}, RC_{mn}, MR, MR)) \Rightarrow \mathbf{RemoveOp}(ROp_{ij}, RC_{ij}, MR)]]$$

//Eliminer les associations en double

$$[(\exists\, \mathbf{RAss_i}, \mathbf{RC_k}(\mathbf{OwnedAttribute}(\mathbf{RP_{ij}}, \mathbf{RC_k}, \mathbf{MR}, \mathbf{RC_{ij}}, -, -) \wedge \mathbf{memberEnd}(\mathbf{RAss_i}, \mathbf{RP_{ij}}, -, -) \wedge \mathbf{OwnedAttribute}(\mathbf{RP_k}, \mathbf{RC_i}, \mathbf{MR}, \mathbf{RC_k}, -, -) \wedge \mathbf{memberEnd}(\mathbf{RAss_i}, \mathbf{RP_k}, -, -)) \wedge \exists\, \mathbf{RAss_j}\ (\mathbf{OwnedAttribute}(\mathbf{RP_{mn}}, \mathbf{RC_k}, \mathbf{MR}, \mathbf{RC_{mn}}, -, -) \wedge \mathbf{memberEnd}(\mathbf{RAss_j}, \mathbf{RP_{mn}}, -, -) \wedge \mathbf{OwnedAttribute}(\mathbf{RP_k}, \mathbf{RC_{mn}}, \mathbf{MR}, \mathbf{RC_k}, -, -) \wedge \mathbf{memberEnd}(\mathbf{RAss_j}, \mathbf{RP_k}, -, -)) \wedge \mathbf{Equivalence\_Associations}(\mathbf{RAss_i}, \mathbf{RAss_j})) \Rightarrow \mathbf{RemoveAss}(\mathbf{RAss_i}, \mathbf{RC_{ij}}, \mathbf{RC_k}, \mathbf{MR})] \wedge$$

//Eliminer les relations de généralisation en double – cas 1

$$[(\exists\, \mathbf{RC_{kl}}\ (\mathbf{superClass}(\mathbf{RC_{kl}}, \mathbf{RC_{ij}}, \mathbf{MR}) \wedge \mathbf{superClass}(\mathbf{RC_{kl}}, \mathbf{RC_{mn}}, \mathbf{MR}))) \Rightarrow \mathbf{RemoveRG}(\mathbf{RC_{kl}}, \mathbf{RC_{ij}}, \mathbf{MR})] \wedge$$

//Eliminer les relations de généralisation en double – cas 2

$$[(\exists\, \mathbf{RC_{op}}\ (\mathbf{superClass}(\mathbf{RC_{ij}}, \mathbf{RC_{op}}, \mathbf{MR}) \wedge \mathbf{superClass}(\mathbf{RC_{mn}}, \mathbf{RC_{op}}, \mathbf{MR}))) \Rightarrow (\mathbf{RemoveRG}(\mathbf{RC_{mn}}, \mathbf{RC_{op}}, \mathbf{MR}))])]$$

La fusion de deux relations de généralisation équivalentes reliant respectivement la super-classe C_m à sa sous-classe C_i et la super-classe C_n à sa sous-classe C_j :

- Crée dans le modèle résultat MR une relation de généralisation (cf. Figure 4-48). La relation de généralisation engendrée a les propriétés suivantes :
  - Super-classe : la classe RC_{mn} (résultante des deux super-classes C_m et C_n).
  - Sous-classe : la classe RC_{ij} (résultante des deux sous-classes C_i et C_j).
- Supprime les doublons (attributs, opération, associations, relation de généralisation équivalents) existants entre les classes RC_{ij} et RC_{mn}.

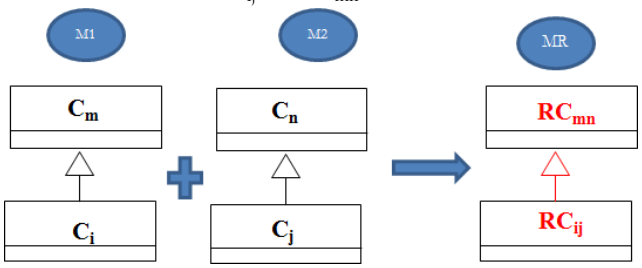


Figure 4-48 Fusion de relations de généralisation

- **Translation de relation de généralisation**

Description informelle

Pour chaque relation de généralisation isolée, nous procédons comme suit :

- Créer dans le modèle résultat une relation de généralisation. La relation de généralisation engendrée a les propriétés suivantes :
  - Super-classe : la classe résultante de la super-classe isolée
  - Sous-classe : la classe résultante de la sous-classe isolée
- Supprimer les doublons (attributs, opération, associations, relation de généralisation) entre la sous-classe et la super-classe résultantes. C'est le même cas de suppression lors de la translation des classes hyponymes.

Représentation formelle

$$\textbf{Translation\_Gen}(C_i, C_j, M, MR)$$
$$\Leftrightarrow [(\exists C_i, C_j\ (\textbf{Class}(C_i, M) \wedge \textbf{Class}(C_j, M) \wedge \textbf{superClass}(C_i, C_j, M))$$
$$\wedge \exists RC_i, RC_j\ (\textbf{Class}(RC_i, MR) \wedge \textbf{Class}(RC_j, MR) \wedge \textbf{ResultElt}(C_i, RC_i)$$
$$\wedge \textbf{ResultElt}(C_j, RC_j))) \Rightarrow (\textbf{superClass}(RC_i, RC_j) \wedge$$

//Eliminer les attributs en double

$$[(\exists\ RAtt_i\ \textbf{OwnedAttribute}(RAtt_i, RC_i, MR, -, -, -) \wedge \exists\ RAtt_j\ \textbf{OwnedAttribute}(RAtt_j, RC_j, MR, -, -, -)$$
$$\wedge \textbf{Equivalence\_Attributes}(RAtt_i, RAtt_j, RC_i, RC_j, MR, MR)) \Rightarrow \textbf{RemoveAtt}(RAtt_j, RC_j, MR)]$$
$$\wedge$$

//Eliminer les opérations en double

$$[(\exists\ ROp_i\ \textbf{OwnedOperation}(ROp_i, RC_i, MR, -) \wedge \exists\ ROP_j\ \textbf{OwnedOperation}(ROp_j, RC_j, MR, -)$$
$$\wedge \textbf{Equivalence\_Operations}(ROp_i, ROp_j, RC_i, RC_j, MR, MR)) \Rightarrow \textbf{RemoveOp}(ROp_j, RC_j, MR)]]$$

//Eliminer les associations en double

$$[(\exists\ RAss_i, RC_k(\textbf{OwnedAttribute}(RP_i, RC_k, MR, RC_i, -, -) \wedge \textbf{memberEnd}(RAss_i, RP_i, -, -)$$
$$\wedge \textbf{OwnedAttribute}(RP_k, RC_i, MR, RC_k, -, -) \wedge \textbf{memberEnd}(RAss_i, RP_k, -, -))$$
$$\wedge \exists\ RAss_j\ (\textbf{OwnedAttribute}(RP_j, RC_k, MR, RC_j, -, -) \wedge \textbf{memberEnd}(RAss_j, RP_j, -, -)$$
$$\wedge \textbf{OwnedAttribute}(RP_k, RC_j, MR, RC_k, -, -) \wedge \textbf{memberEnd}(RAss_j, RP_k, -, -))$$
$$\wedge \textbf{Equivalence\_Associations}(RAss_i, RAss_j)) \Rightarrow \textbf{RemoveAss}(RAss_j, RC_j, RC_k, MR)] \wedge$$

//Eliminer les relations de généralisation en double – cas 1

$$[(\exists\ RC_{kl}\ (\textbf{superClass}(RC_{kl}, RC_i, MR) \wedge \textbf{superClass}(RC_{kl}, RC_j, MR))) \Rightarrow \textbf{RemoveRG}(RC_{kl}, RC_j, MR)] \wedge$$

//Eliminer les relations de généralisation en double – cas 2

$$[(\exists\ RC_{mn}\ (\textbf{superClass}(RC_i, RC_{mn}, MR) \wedge \textbf{superClass}(RC_j, RC_{mn}, MR))) \Rightarrow \textbf{RemoveRG}(RC_i, RC_{mn}, MR)])]$$

Pour chaque relation de généralisation isolée qui relie la super-classe C_i à sa sous-classe C_j :

- Une relation de généralisation est créée dans le modèle résultat MR (cf. Figure 4-49). La relation de généralisation engendrée a les propriétés suivantes :
  - Super-classe : la classe RC_i (résultante de C_i)
  - Sous-classe : la classe RC_j (résultante de C_j)
- Suppression des doublons (attributs, opération, associations, relation de généralisation équivalents), existants entre les classes RC_i et RC_j.

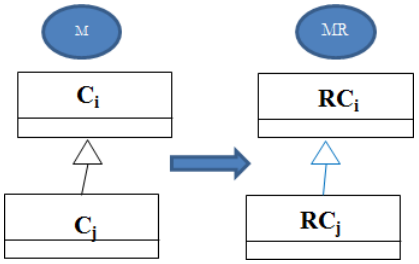


Figure 4-49 Translation de relation de généralisation

4.2.6. Conclusion

Nous avons présenté dans cette partie les règles de composition en logique des prédicats. Les entrées, les sorties et les ressources du système Co-Models ont été représentées en langage formel, en transformant les modèles UML et OWL, en modèles exprimés en logique des prédicats. Les transformations ont été réalisées dans le contexte de l'Ingénierie Dirigée par les Modèles.

Conclusion

Ce chapitre a présenté notre contribution à la composition de modèles. Nous avons répondu à notre problématique en proposant une approche hybride de composition de modèles, qui permet de générer de réelles correspondances et de réelles différences pouvant exister entre les modèles, et obtenir un modèle résultat sans conflits de redondances et de perte d'information. Notre principale contribution est le fait d'exploiter pleinement l'ontologie afin de détecter toutes les correspondances pouvant exister entre les modèles.

Nous avons présenté le système Co-Models, suivi du processus qui décrit les étapes de comparaison et de fusion/translation. Ce système à base de règles a été présenté sous forme de règles informelles puis de règles formelles en logique des prédicats. Nous avons opté pour la logique des prédicats parce qu'elle est la plus proche, de nos besoins, en terme d'expressions (connecteurs logiques, quantificateurs, prédicats, variables, etc.), son pouvoir d'expression est grand, elle permet d'éliminer toute ambiguïté d'expression, et enfin, elle permet un raisonnement rigoureux et une description précise.

Afin d'exprimer les règles en logique des prédicats, les entrées, les sorties et les ressources du système ont été représentées aussi en langage formel, ce qui a nécessité la transformation des modèles UML et OWL, en modèles exprimés en logique des prédicats. Les transformations ont été réalisées dans le contexte de l'Ingénierie Dirigée par les Modèles, où une transformation est définie par des règles de transformation qui permettent de transformer un modèle conforme à son méta-modèle en un autre modèle conforme à son méta-modèle.

Le chapitre qui suit présente deux études de cas de composition de modèles, l'implémentation de l'approche proposée ainsi que son évaluation.

Chapitre 5

Mise en œuvre et validation

5. Mise en œuvre et validation

Introduction

Le chapitre précédent a permis de répondre à un certain nombre de préoccupations portant sur la composition de modèles. A présent, nous allons nous focaliser, dans ce chapitre, sur les aspects liés à l'implémentation de notre prototype, et la validation et l'évaluation de notre approche.

Afin de valider nos propositions, nous avons été amenés à proposer des études de cas. Tout au long de notre travail, nous avons effectué un ensemble de tests de composition de modèles dans différents domaines, à savoir, le domaine du tourisme (agence de voyage), des bibliothèques, de l'enseignement, etc. Notre choix s'est porté finalement sur deux études de cas, la première concerne le domaine de banque et la seconde le domaine du e-gouvernement.

Dans un premier temps, nous avons voulu présenter un exemple de modèles couvrant un grand nombre d'hétérogénéités afin d'illustrer tous les cas possibles de composition de modèles. Pour cela, nous avons choisi le domaine bancaire, un domaine classique utilisé dans les exemples de modélisation. Dans un second temps, nous avons voulu traiter un cas réel de composition de modèles. Nous avons donc choisi le cas du projet de la gestion intégrée des recettes du Ministère de l'Economie et des Finances.

La section 5.1 décrit les études de cas que nous avons traitées. La section 5.2 présente les principaux éléments de l'implémentation de notre prototype. Nous définissons succinctement les outils que nous avons utilisés, à savoir Eclipse, JDOM et JENA. Nous présentons, ensuite, les principales interfaces de Co-Models. La section 5.3 expose une évaluation de notre approche. Nous présentons, ainsi, le cadre général, les objectifs, et l'analyse des résultats des expérimentations effectuées.

5.1. Etudes de cas

Cette section consiste à présenter la composition de modèles dans les domaines bancaire et du e-gov.

5.1.1. Etude de cas 1 : fusion de Banques

Depuis la fin des années 1980, de plus en plus de banques se regroupent. Il s'agit d'une tendance mondiale, à laquelle on peut difficilement échapper. Ceci a pour objectif le développement de la présence à l'international comme relais de croissance aux marchés nationaux et la course à la taille critique pour faire face à l'ouverture des frontières [Jai, 2006]. Un tel regroupement construit un groupe financier puissant, renforce son pouvoir de marché, et donne naissance à une banque puissante capable de faire face à la concurrence [Gagnon, 2011].

Plusieurs banques ont fusionné telles que :

- La banque américaine Bankers Trust et la banque allemande Deutchst Bank[23]
- Dubai Bank (DUB) et Emirates Islamic Bank (EIB), deux filiales d'Emirates NBD (ENBD) dédiées à la finance islamique, ont fusionné afin de former le quatrième groupe bancaire sharia compliant des Emirats Arabes Unis après Dubai Islamic Bank, Abu Dhabi Islamic Bank et Al Hilal Bank.
- COBACI et la banque atlantique en Côte d'Ivoire[24]

[23] https://www.bankerstrust.com/

- La Caisse d'Epargne et la Banque Populaire en France[25]
- Les deux banques marocaines BCM et Wafa qui a donné naissance à AttijariWafa Bank[26]

Cependant, plusieurs difficultés et risques sont rencontrés lors de la fusion de deux banques qui se concrétisent, d'abord, par une perte d'emplois liée à la volonté de baisse des coûts, et ensuite, par des problèmes d'obstacles culturels, causés par les différences de cultures d'entreprise. Se rajoutent des difficultés de gouvernance de la nouvelle structure, et la complexité de la restructuration du nouveau système d'information. La fusion de deux systèmes d'information se montre très complexe, et nécessite beaucoup d'efforts du fait de l'hétérogénéité des processus, des fonctionnalités, des applications et, ainsi, des modèles.

Afin de contribuer à la résolution de la difficulté de restructuration de la banque résultante, nous proposons aux banques concernées, lors de la fusion de leurs systèmes d'information, d'appliquer notre approche de composition de modèles.

Reprenons l'exemple des deux diagrammes de classes, présentés dans le chapitre 3 section 3.1.2, auxquels nous appliquons notre processus de composition. Ainsi, nous présentons dans la suite, l'ontologie de domaine de banque utilisée, la comparaison de ces deux modèles, les mappings d'équivalence et d'hyponymie, les éléments isolés identifiés ainsi que le modèle résultat de la composition de modèles.

5.1.1.1. Ontologie du domaine bancaire

Nous présentons ci-dessous l'ontologie du domaine bancaire, utilisée dans la composition des modèles M1 et M2. L'ontologie exprimée en langage OWL et sa représentation graphique dans PROTÉGÉ en utilisant le pluggin OWLViz, sont présentées en annexe 1. Dans ce cas d'étude, l'ontologie du domaine utilisée est décrite par un ensemble d'informations présentées ci-dessous :

- **Classes**: « Personne », « Client », « Employé », « Personnel », « Compte », «PersonnePhysique » « PersonneMorale », « CarteCrédit », « Chéquier », « Marié », « Célibataire », « CompteCourant », « CompteEpargne », « Versement », « Opération », « Retrait », « SituationFamiliale »
- **Propriétés d'objet** :
  - Un client est géré par un employé
  - Un compte bancaire possède une carte de crédit « CarteCredit » et un chéquier « Chéquier »
  - Une personne possède un compte
  - Un compte comprend une opération
  - Une personne a une situation familiale
- **Relations de subsomption** :
  - Une situation familiale peut être « Marié » ou « Célibataire »
  - Une « Personne » dans une banque peut être un « Client » ou « Employé »
  - Un client peut être personne physique «PersonnePhysique » ou morale « PersonneMorale »

[24] http://www.banqueatlantique.net/
[25] http://www.bpce.fr/
[26] http://www.attijariwafabank.com

  - Un compte peut être un « CompteCourant » ou « CompteEpargne »
  - Une « Opération » peut être un « Versement » ou « Retrait »
- **Propriétés de données** :
  - Une personne est caractérisée par un numéro de téléphone, une adresse, et un email
  - Un employé possède un identifiant et un poste
  - Une personne morale a une raison sociale
  - Une personne physique a un nom, un prénom et une CIN
  - Un compte possède un numéro et un solde
  - Une carte de crédit est décrite par un code, un retrait maximal, un type, et une date d'expiration
  - Un chéquier possède un numéro
  - Une opération est caractérisée par une référence, une date et un montant
- **Relation d'équivalence entre les classes**
  - « Personnel » et « Employé »
- **Relation de disjonction entre les classes**
  - « Marié » et « Célibataire »
- **Relation d'inverse entre les propriétés d'objet**
  - « Gérer » et « EstGérePar »
- **Relation d'équivalence entre les propriétés d'objet**
  - « Posséder » et « Avoir »
- **Relation d'équivalence entre les propriétés de données**
  - « numéro » et « code ».

5.1.1.2. Comparaison des deux modèles de banques

Nous exposons, dans cette partie, le résultat de l'application du processus de comparaison des deux modèles bancaires, en expliquant les correspondances qui existent entre les éléments des modèles M1 et M2 représentés respectivement par les diagrammes de classes des figures 5-1 et 5-2.

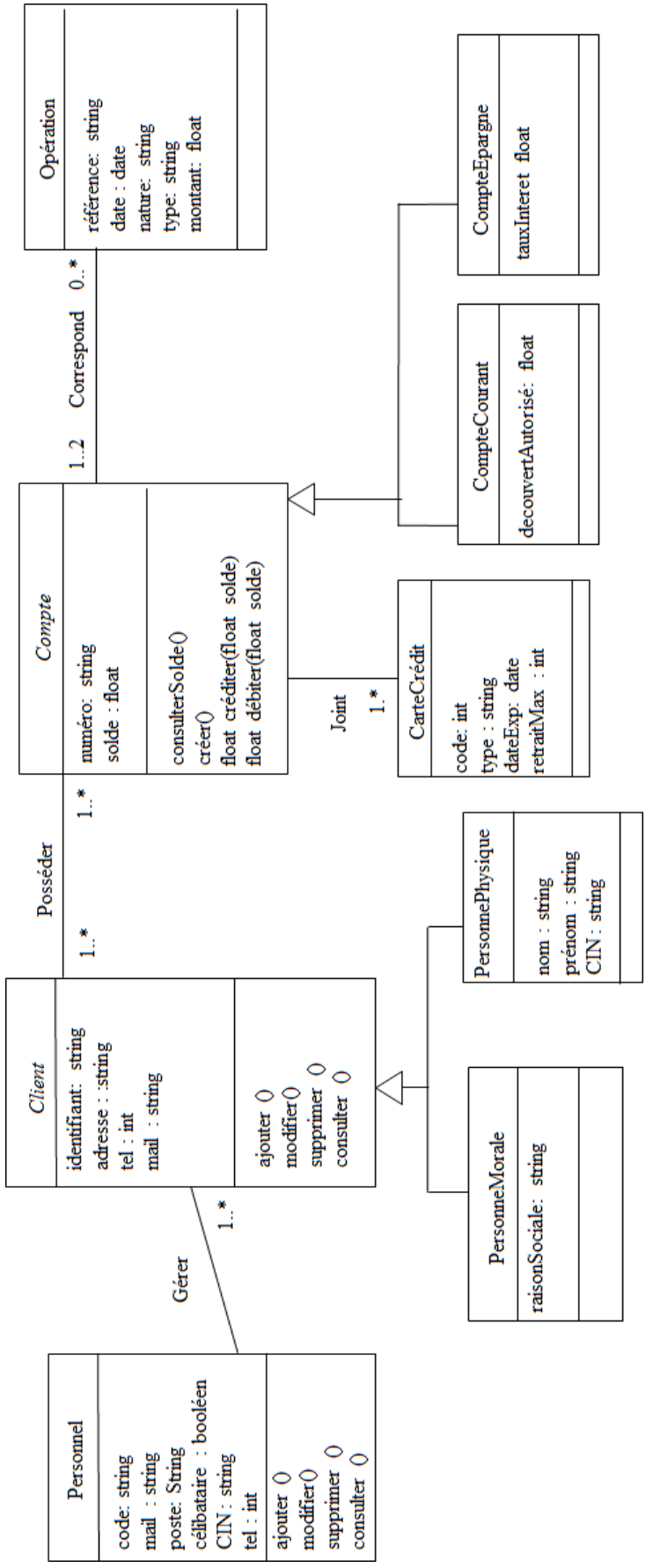


Figure 5-1 Modèle M1 de la Banque1

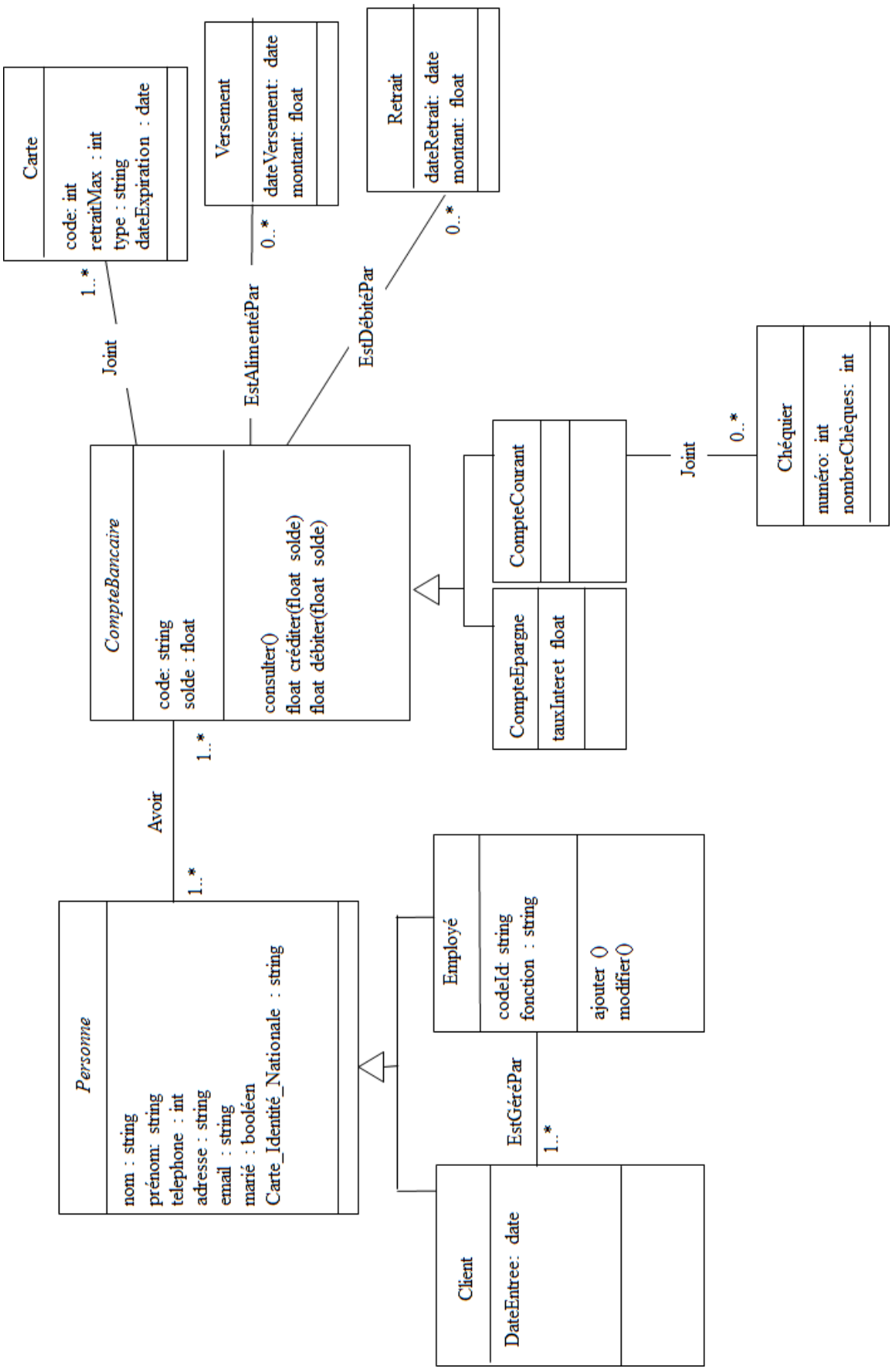


Figure 5-2 Modèle M2 de la Banque2

Les mappings et les éléments isolés sont représentés comme suit :

- Le mapping d'équivalence, entre deux modèles M1 et M2, est représenté sous forme d'un tableau de deux colonnes contenant respectivement les éléments de M1 et M2. Une ligne du tableau contient, un élément de M1 et son équivalent dans M2.
- Le mapping d'hyponymie, est également représenté, sous forme d'un tableau de deux colonnes contenant les éléments de modèles en relation d'hyponymie. Une ligne du mapping contient un élément de M1 et son hyponyme dans M2.
- Les éléments isolés, quant à eux, sont listés dans un tableau de trois colonnes. La première colonne désigne les classes isolées, la deuxième comprend les attributs et opérations isolés et la troisième colonne représente les associations et les relations de généralisation isolées.

Un élément du mapping ou isolé est représenté comme suit. Nous avons choisi cette représentation afin de garantir l'unicité lors de la représentation des éléments.

- Classe : Nom de la classe. Exemple : « Client »
- Attribut : Nom de la classe à laquelle il appartient, suivi du nom de l'attribut, séparés par un point « . ». Exemple : « Client.adresse »
- Opération : Nom de la classe à laquelle elle appartient, suivi du nom de l'opération, séparés par un point « . ». Exemple : « Personnel.ajouter() »
- Paramètre : Nom de la classe à laquelle il appartient, suivi du nom de l'opération à laquelle il appartient, suivi du nom de paramètre, séparés par un point « . ». Exemple « Compte.débiter().solde »
- Association : Nom des deux classes qu'elle relie, suivi du nom de l'association, séparés par un point « . ». Exemple : « Personnel.Client.Gérer »
  Remarque : Si, en voulant comparer la structure locale ou globale de deux classes C1 et C2, et nous constatons que C2 hérite une caractéristique (attribut, opération, association) de sa super-classe C3, alors nous remplaçons le nom de la classe C2, comme représenté dans le cas classique, par le nom de la super-classe C3, suivi du nom de la classe C2, séparés par un point « . ». Exemple : «Personne.Client.nom »
- Relation de généralisation : RG(Nom de la super-classe, Nom de la sous-classe). Exemple : « RG(Personne, Employé) »

Le Tableau 5-1 présente le résultat de l'application du processus de comparaison des éléments de modèles des banques M1 et M2. Les deux premières colonnes présentent respectivement les éléments de M1 et M2. La troisième colonne présente une explication de la correspondance d'équivalence: équivalence syntaxique « Syn », équivalence sémantique « Sem », équivalence de classes en structure locale partielle « LP », équivalence de classes en structure locale totale « LT », équivalence de classes en structure globale partielle « GP », équivalence de classes en structure globale totale « GT », équivalence d'éléments (attributs, opérations, paramètres) en structure locale « L ». Le « S » désigne une relation de subsomption et le « H » une relation d'hyponymie entre les classes.

Nous commençons d'abord par éliminer les ponctuations des éléments des deux modèles. « Carte_Identité_Nationale » devient donc « CarteIdentitéNationale ».

M1	M2	Explication
Classes		
CompteCourant	CompteBancaire	S
CompteEpargne	CompteBancaire	S
Compte	CompteEpargne	S
Compte	CompteCourant	S
PersonneMorale	Client	S
PersonnePhysique	Client	S
PersonneMorale	Personne	S
PersonnePhysique	Personne	S
Personnel	Personne	S
Opération	Versement	S+H
Opération	Retrait	S+H
Client	Personne	S
Client	Client	Syn(Identité)+LP+GP
CompteCourant	CompteCourant	Syn(Identité)+LP+GP
CompteEpargne	CompteEpargne	Syn(Identité)+LP+GP
Compte	CompteBancaire	Syn(Inclusion)+LP+GP
CarteCrédit	Carte	Syn(Inclusion)+LT+GT
Personnel	Employé	Sem(Equiv onto)+ LP+GP
Attributs et opérations		
Client.adresse	Personne.Client.adresse	Syn(Identité)+L
Client.mail	Personne.Client.email	Syn(Inclusion)+ L
Client.tel	Personne.Client.telephone	Syn(Inclusion)+ L
Personnel.code	Employé.codeid	Syn(Inclusion)+ L
Personnel.mail	Personne.Employé.email	Syn(Inclusion)+ L
Personnel.tel	Personne.Employé.telephone	Syn(Inclusion)+ L
Personnel.CIN	Personne.Employé.CarteIdentitéNationale	Syn(Acronyme)+ L
Personnel.poste	Employé.fonction	Sem(Synonymie)+ L
Personnel .célibataire	Personne.Employé.marié	Sem(Disjonction)+ L
Personnel.ajouter()	Employé.ajouter()	Syn(Identité)+ L
Personnel.modifier()	Employé.modifier()	Syn(Identité)+ L
Compte.solde	CompteBancaire.solde	Syn (Identité) + L
Compte.numéro	CompteBancaire.code	Sem(Equiv onto)+ L
Compte.débiter()	CompteBancaire.débiter()	Syn(Identité)+ L
Compte.débiter().solde	CompteBancaire.débiter().solde	Syn(Identité)+ L
Compte.créditer()	CompteBancaire.créditer()	Syn(Identité)+ L
Compte.créditer().solde	CompteBancaire.créditer().solde	Syn(Identité)+ L
Compte.consulterSolde()	CompteBancaire.consulter()	Syn(Inclusion)+ L
Compte.CompteEpargne.solde	CompteBancaire.CompteEpargne.solde	Syn(Identité)+ L
Compte.CompteEpargne.numéro	CompteBancaire.CompteEpargne.code	Sem(Equiv onto)+ L
Compte.CompteEpargne.débiter()	CompteBancaire.CompteEpargne.débiter()	Syn(Identité)+ L
Compte.CompteEpargne.créditer()	CompteBancaire.CompteEpargne.créditer()	Syn(Identité)+ L
Compte.CompteEpargne.consulterSolde()	CompteBancaire.CompteEpargne.consulter()	Syn(Iinclusion)+ L
CompteEpargne.tauxInteret	CompteEpargne.tauxInteret	Syn(Identité)+ L
Compte.CompteCourant.solde	CompteBancaire.CompteCourant.solde	Syn(Identité)+ L
Compte. CompteCourant.numéro	CompteBancaire.CompteCourant.code	Sem(Equiv onto)+ L
Compte.CompteCourant.débiter()	CompteBancaire.CompteEpargne.débiter()	Syn(Identité)+ L
Compte. CompteCourant.créditer()	CompteBancaire. CompteCourant.créditer()	Syn(Identité)+ L
Compte.CompteCourant.consulterSolde()	CompteBancaire.CompteCourant.consulter()	Syn(Inclusion)+ L
CarteCrédit.code	Carte.code	Syn(Identité)+ L
CarteCrédit.type	Carte.type	Syn(Identité)+ L

CarteCrédit.retraitMax	Carte. retraitMax	Syn(Identité)+ L
CarteCrédit.dateExp	Carte.dateExpiration	Syn(Inclusion)+ L
Associations et RG		
Personnel.Client.Gérer	Employé.Client.EstGérePar	Sem(Inverse)
Client.Compte.Posséder	Personne.Client.CompteBancaire.Avoir	Sem(Equiv onto)
Compte.CarteCrédit.Joint	CompteBancaire.Carte.Joint	Syn(Identité)
RG(Compte, CompteEpargne)	RG(CompteBancaire, CompteEpargne)	
RG(Compte, CompteCourant)	RG(CompteBancaire, CompteCourant)	

Tableau 5-1 Composition des modèles bancaires : Comparaison

5.1.1.3. Mapping d'équivalence

Le Tableau 5-2 présente le mapping d'équivalence, obtenu après le paramétrage de l'utilisateur, autrement dit, après le choix de l'option d'équivalence de classes.

Nous constatons que quel que soit le choix fait par l'utilisateur, à savoir LP, GP ou LP+GP, cela donnent le même mapping. Les autres choix s'avèrent peu intéressants, du fait que le modèle résultat comporterait plusieurs classes dupliquées comme (« Compte » et « CompteBancaire », « Personnel » et « Employé », etc.). En effet, pour les choix GT, GT+LP, LT+GP, LT+GT, LT, et GT+LT : seules les classes « CarteCrédit » de M1 et « Carte » de M2 seront équivalentes.

M1	M2
Client	Client
CompteCourant	CompteCourant
CompteEpargne	CompteEpargne
Compte	CompteBancaire
CarteCrédit	Carte
Personnel	Employé
Client.adresse	Personne.Client.adresse
Client.mail	Personne.Client.email
Client.tel	Personne.Client.telephone
Personnel.code	Employé.codeid
Personnel.mail	Personne.Employé.email
Personnel.tel	Personne.Employé.telephone
Personnel.CIN	Personne.Employé.CarteIdentitéNationale
Personnel.poste	Employé.fonction
Personnel .célibataire	Personne.Employé.marié
Personnel.ajouter()	Employé.ajouter()
Personnel.modifier()	Employé.modifier()
Compte.solde	CompteBancaire.solde
Compte.numéro	CompteBancaire.code
Compte.débiter()	CompteBancaire.débiter()
Compte.débiter().solde	CompteBancaire.débiter().solde
Compte.créditer()	CompteBancaire.créditer()
Compte.créditer().solde	CompteBancaire.créditer().solde
Compte.consulterSolde()	CompteBancaire.consulter()
Compte.CompteEpargne.solde	CompteBancaire.CompteEpargne.solde
Compte.CompteEpargne.numéro	CompteBancaire.CompteEpargne.code
Compte.CompteEpargne.débiter()	CompteBancaire.CompteEpargne.débiter()
Compte.CompteEpargne.créditer()	CompteBancaire.CompteEpargne.créditer()

Compte.CompteEpargne.consulterSolde()	CompteBancaire.CompteEpargne.consulter()
CompteEpargne.tauxInteret	CompteEpargne.tauxInteret
Compte.CompteCourant.solde	CompteBancaire.CompteCourant.solde
Compte. CompteCourant.numéro	CompteBancaire.CompteCourant.code
Compte.CompteCourant.débiter()	CompteBancaire.CompteEpargne.débiter()
Compte. CompteCourant.créditer()	CompteBancaire. CompteCourant.créditer()
Compte.CompteCourant.consulterSolde()	CompteBancaire.CompteCourant.consulter()
CarteCrédit.code	Carte.code
CarteCrédit.type	Carte.type
CarteCrédit.retraitMax	Carte. retraitMax
CarteCrédit.dateExp	Carte.dateExpiration
Personnel.Client.Gérer	Employé.Client.EstGérérPar
Client.Compte.Posséder	Personne.Client.CompteBancaire.Avoir
Compte.CarteCrédit.Joint	CompteBancaire.Carte.Joint
RG(Compte, CompteEpargne)	RG(CompteBancaire, CompteEpargne)
RG(Compte, CompteCourant)	RG(CompteBancaire, CompteCourant)

Tableau 5-2 Composition des modèles bancaires : Mapping d'équivalence

5.1.1.4. Eléments isolés

Les éléments isolés sont représentés dans le Tableau 5-3.

Classes	Attributs et Opérations	Associations et Relations de généralisation
PersonnePhysique	Client.ajouter()	RG(Client,PersonnePhysique)
PersonneMorale	Client.modifier()	RG(Client,PersonneMorale)
Personne	Client.supprimer()	RG(Personne,Employé)
Chéquier	Client.consulter()	RG(Personne,Client)
Opération	Personnel.supprimer()	CompteBancaire.Versement.EstAlimentéPar
Versement	Personnel.consulter()	Compte.Opération.Correspond
Retrait	Personne.nom	Compte.Chéquier.Joint
	Personne.prénom	CompteBancaire.Retrait.EstDébitéPar
	Personne.telephone	
	Personne.mail	
	Personne.adresse	
	Personne.marié	
	Personne.CarteIdentitéNationale	
	Client.identifiant	
	Client.DateEntrée	
	PersonneMorale. raisonSociale	
	CompteCourant. decouvertAutorisé	
	Chéquier.numéro	
	Chéquier.nombreChèques	
	Compte.créer()	
	PersonnePhysique.nom	
	PersonnePhysique.prénom	
	PersonnePhysique.CIN	
	Opération.référence	
	Opération.date	
	Opération.nature	
	Opération.type	
	Opération.montant	

	Versement.dateVersement	
	Versement.motant	
	Retrait.dateRetrait	
	Retrait.montant	

Tableau 5-3 Composition des modèles bancaires : Eléments isolés

5.1.1.5. Mapping d'hyponymie

Les classes hyponymes sont représentées dans le Tableau 5-4. Les deux colonnes représentent les classes de M1 et de M2.

M1	M2
Opération	Versement
Opération	Retrait

Tableau 5-4 Composition des modèles bancaires : Mapping d'hyponymie

5.1.1.6. Modèle résultat : Cas bancaire

Le modèle résultat des deux modèles de banque M1 et M2 est représenté dans la Figure 5-3 ci-dessous. Il représente le cas où l'utilisateur du système de composition de modèles a choisi comme modèle principal M1. Les éléments fusionnés sont en rouge et ceux translatés sont en bleu.

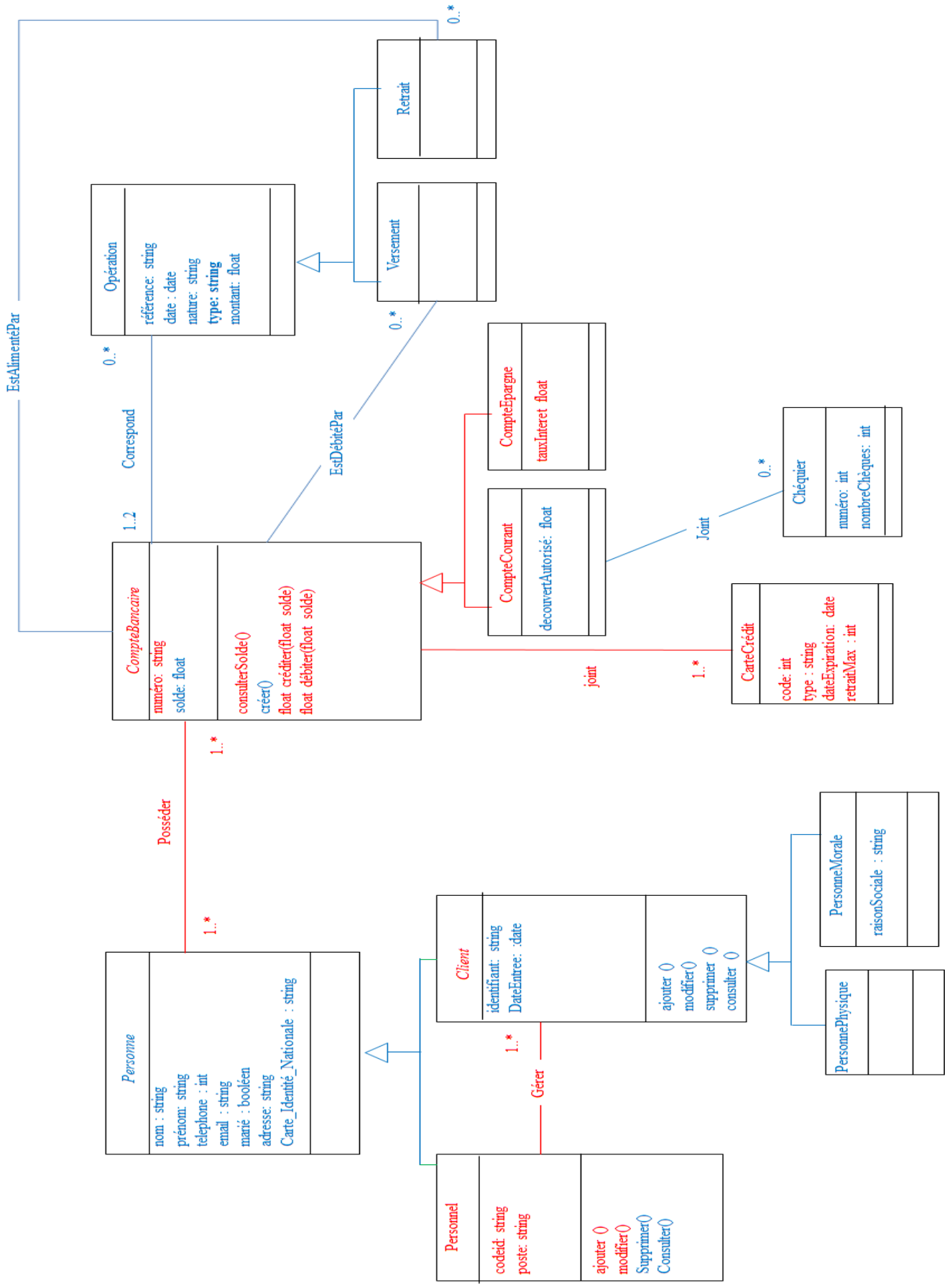


Figure 5-3 Modèle résultat de la composition de modèles bancaires M1 et M2

Les classes « Personnel », « CompteBancaire », « CompteCourant », « CompteEpargne » et « CarteCrédit » ont été créées suite à la fusion de classes équivalentes. Les classes restantes « Personne », « PersonnePhysique », « PersonneMorale », « Chéquier », « Versement », « Retrait » et « Opération » ont été translatées dans le modèle résultat du fait qu'elles sont isolées.

Dans les classes fusionnées, on retrouve les attributs/opérations fusionnés et ceux translatés. Les classes isolées ont été translatées contenant leurs attributs/opérations isolés.

La classe résultante « Client » est abstraite car elle résulte de deux classes, l'une étant abstraite, et l'autre concrète. Les classes « Personne » et « CompteBancaire » sont abstraites car elles résultent de deux classes abstraites.

Comme est expliqué dans la règle de fusion de classes, les attributs présents dans M2 de la classe « Client » (hérités de la classe « Personne ») étant équivalents aux attributs de la classe « Client » du modèle M1, n'ont pas été fusionnés, même s'ils apparaissent dans le mapping d'équivalence. Les attributs « adresse », « email » et « telephone » de la classe « Personne » de M2 ont été translatés donc lors de la translation de « Personne ».

Une relation de généralisation a été créée, reliant les classes hyponymes « Opération » et « Versement », et « Opération » et « Retrait ».

Les attributs présents dans le modèle M1 « dateVersement » et « montant » de la classe « Versement », et « dateRetrait » et « montant » de la classe « Retrait » ont été supprimés, car leurs équivalents respectifs existent : « date » et « montant » dans la super-classe « Opération ».

De la même manière, les attributs présents dans le modèle M1 « nom », « prénom » et « CIN » de la classe « PersonnePhysique » ont été supprimés en raison de présence de leurs équivalents dans la super-classe « Personne » dans le modèle résultat.

Les associations « Posséder », « Gérer » et « Joint » (reliant « CompteBancaire » et « CarteCrédit ») ont été créées suite à une fusion. Les autres associations « EstAlimentéPar », « EstDébitéPar », « Joint » (reliant « CompteCourant » et « Chéquier »), et « Comprend » ont été rajoutées dans le modèle résultat.

Les relations de généralisation reliant « CompteBancaire » et « CompteCourant », « CompteBancaire » et « CompteEpargne » ont été fusionnées et créées dans le modèle résultat. Les autres relations de généralisation ont été translatées car elles appartiennent aux éléments isolés.

Nous avons présenté dans cette section l'étude de cas bancaire. Nous présentons dans ce qui suit notre deuxième étude de cas concernant la gestion intégrée des recettes dans un contexte du e-gouvernement.

5.1.2. Etude de cas 2 : E-gouvernment – Cas du système de Gestion Intégrée de la Recette

Nous présentons, dans cette section, l'application de notre approche sur un cas réel, issu du domaine du e-gouvernement. Il s'agit du système de la Gestion Intégrée des Recettes (GIR) du Ministère de l'Economie et des Finances. Nous commençons par définir brièvement le e-gouvernement et mettre en avant le besoin d'interopérabilité et d'intégration pour le e-gouvernement. Ensuite, nous présentons le cadre général de la gestion de la recette, à savoir l'existant, les problèmes rencontrés, la

démarche de la mise en œuvre de GIR, ainsi que notre proposition. Enfin, nous mettons en avant l'application de notre approche de composition sur deux modèles représentant deux applications « Recette Budgétaire » et « Perception ».

5.1.2.1. Définition du e-gov

Le e-gouvernement (e-gov) ou gouvernement électronique est une manière fondamentalement nouvelle, permettant de fournir des services publics en utilisant, de façon optimale, les Technologies de l'Information et de la Communication (TIC). En d'autres termes, il désigne l'utilisation des TIC dans les administrations publiques, associée à des changements de l'organisation de celles-ci et au développement de nouvelles aptitudes professionnelles, afin d'améliorer la qualité et la fourniture des services publics, les processus démocratiques et de renforcer le soutien des politiques publiques.

Parmi les définitions, il y a celle de [Wimmer et al., 2001] : « Le gouvernement électronique (appelé aussi administration électronique) reflète les visions finales pour que les administrations publiques et les gouvernements subissent une modernisation et une réorganisation énormes ».

Cependant, le e-gov ne se limite pas à la simple mise en ligne d'informations sur les sites web des administrations publiques. Il implique une profonde refonte de la structure et du fonctionnement des administrations, ce que l'on regroupe sous le vocable "back office". Les procédures administratives telles que la collecte, le traitement et l'échange électronique des données au sein ou entre administrations, doivent être adaptées à la fourniture de services publics électroniques qui répondent aux besoins des citoyens et des entreprises [Ouchettou et al., 2007].

L'administration électronique peut se développer dans tout type d'administration ou de service public, en contact avec le public, c'est-à-dire front-office ou non back-office. Elle se caractérise par l'emploi des TIC visant à améliorer (1) les processus, la communication entre usagers et administrations ou entre administrations, et (2) l'efficacité de l'administration, que ce soit en termes de délais, de qualité, ou de productivité des agents publics.

Les supports de l'administration électronique sont nombreux. On pense souvent d'abord à Internet (services web sur ordinateur ou téléphone mobile). Mais un projet d'administration électronique peut aussi s'appuyer sur toute forme de télématique : communication en champ proche, bluetooth, les projets de carte à puce, éventuellement combinés à la biométrie (carte d'identité électronique, passeport biométrique, etc.), ainsi que des procédures de vote électronique, ou encore la vidéo surveillance, la constitution de bases de données et des procédés biométriques de reconnaissance faciale.

5.1.2.2. Besoin d'intégration et d'interopérabilité dans le e-gouvernement

Depuis les années 90, la plupart des pays ont publié leurs stratégies e-gouvernement et ont défini des approches diverses entraînant des progrès significatifs à tous les niveaux de l'administration publique. Malgré ces progrès, le niveau d'ambition étant plus élevé, ce qui a conduit à des déceptions. Une des causes de la faiblesse des progrès est que de nombreux efforts du e-gouvernement sont confrontés à un manque d'interopérabilité des systèmes [Janssen et al., 2011]. Créer l'interopérabilité et l'intégration (au sens des définitions proposées par Vernadat présentées dans le chapitre 3) est une tâche complexe [Scholl et al., 2007], [Weerakkody et al., 2007]. Selon [UNDESA, 2008], les

organismes publics sont rarement interconnectés, intégrés et coordonnés entre eux afin de fournir un guichet unique, orientée services proactifs vers les besoins des citoyens.

De même, au Maroc, selon [Debbagh et al., 2007], le e-gouvernement connaît des limites qui se concrétisent par le développement local des services. En effet, le progrès du e-gouvernement est accompli au niveau monodépartemental. La situation actuelle du e-gov se limite à des applications verticales « en silos » avec des services administratifs des différents départements ministériels.

Ces limites engendrent de nombreuses difficultés, à savoir le suivi de l'état d'un service à plusieurs intervenants, la recherche du service adéquat, et la nécessité de consulter plusieurs portails pour un service unique. La problématique actuelle du e-gov consiste donc en l'interopérabilité et l'intégration de ses services.

D'après [Janssen et al., 2011], les organisations et les ministères doivent collaborer, créer des chaînes d'activités, fonctionner dans les réseaux et agir comme un tout intégré vers le public. Cela nécessite une refonte du secteur public.

Pour le Maroc, devant les limites de l'administration électronique, il convient désormais, selon [Debbagh et al., 2007], de s'attacher davantage au développement de la dimension multidépartementale du e-gov, permettant aux administrations de travailler de manière plus synergique et mieux intégrée. La mise en place d'une plateforme d'interopérabilité entre systèmes d'information, devra assurer la gestion de l'exécution des processus transversaux inter-administrations, tout en prenant en charge toutes les fonctionnalités relatives aux échanges sécurisée et transparents des informations soit entre administrations, soit entre l'usager et l'administration.

5.1.2.3. Cas du Ministère de l'Economie et des Finances (MEF) au Maroc

Le Ministère de l'Economie et des Finances, face à ces limites, propose une intégration et une interopérabilité de ces entités métier. Le MEF se charge de plusieurs missions qui se sont concrétisées en la création d'entités métier (cf. Figure 5-4), à savoir la gestion des impôts, la gestion de la douane, la réglementation des finances, le contrôle financier, la gestion de la dette, la gestion des recettes, la gestion des dépenses, la gestion des biens fonciers et les prévisions économiques [MEF-Docs, 2012].

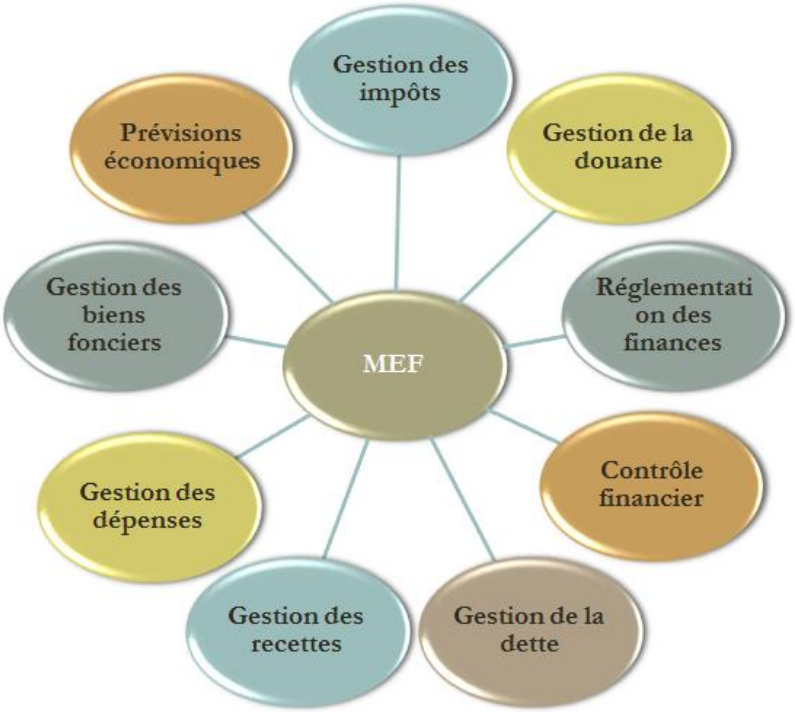


Figure 5-4 Entités métier du MEF [MEF-Docs, 2012]

Antérieurement à l'insertion dans le projet e-gov, comme le montre la Figure 5-5, les entités métier ne sont pas connectées, et pour chacune d'entre elles, les acteurs participants disposent chacun d'une application et d'une base de données [MEF-Docs, 2012].

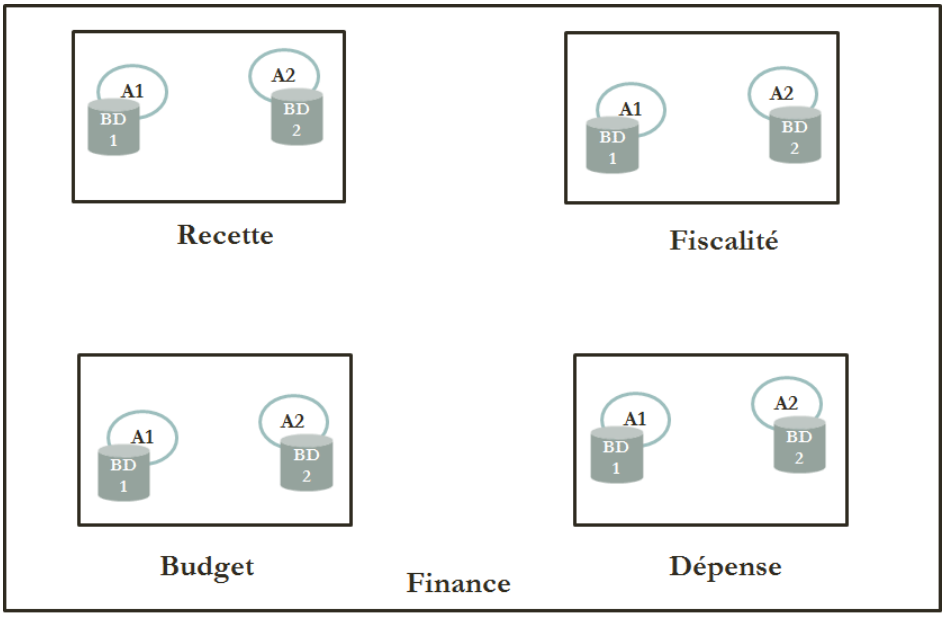


Figure 5-5 Etat du MEF antérieur aux initiatives du e-gov

L'objectif du MEF est de créer des systèmes intégrés pour chaque entité métier telles que la Gestion Intégré des Dépenses (GID), GIR, la Gestion Intégrée de la Fiscalité, la Gestion Intégrée du Budget (GIB), et de mettre en œuvre une intégration et une interopérabilité entre ces entités (cf. Figure 5-6).

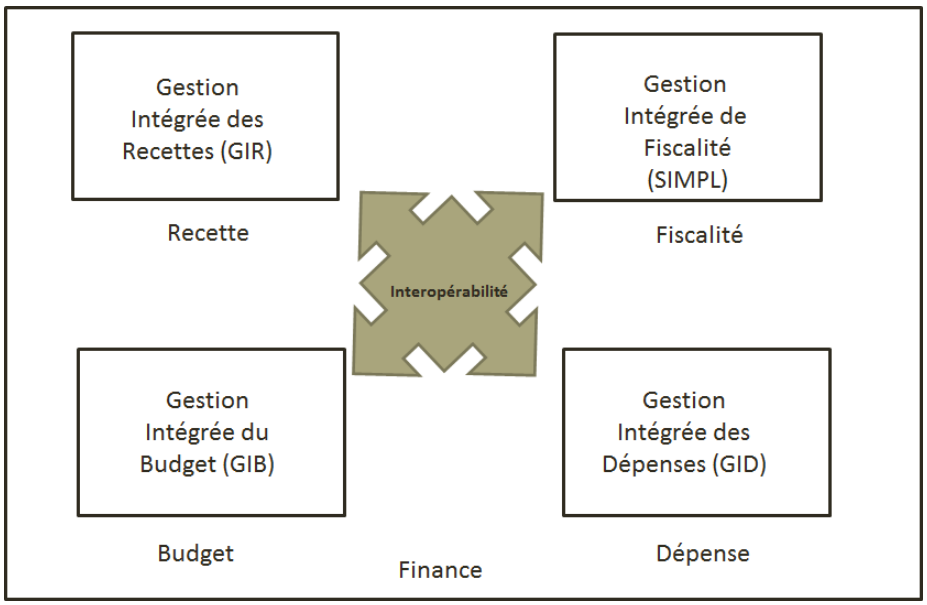


Figure 5-6 Objectif du MEF

Nous nous intéressons, pour notre étude de cas, au système GIR. Nous décrivons d'abord la gestion de la recette dans la trésorerie générale, les problèmes rencontrés, la solution GIR en cours de réalisation, et leur démarche de mise en œuvre. Nous exposerons ensuite notre proposition. Ces informations sont issues de documents officiels, fournis par le ministère ou à travers les entretiens réalisés avec des ingénieurs travaillant sur les systèmes d'information du MEF.

5.1.2.4. Cas du système de Gestion Intégrée des Recettes (GIR)

Actuellement, avant la mise en place du projet GIR, la gestion (encaissement) de la recette dans la Trésorerie Générale du Royaume se passe de la manière présentée dans la Figure 5-7. L'ordonnateur crée et émet la créance au contribuable qui doit payer la créance au niveau du comptable encaisseur. Celui-ci, après encaissement, remet le justificatif au contribuable, qui reçoit une quittance en cas de paiement en liquide ou une déclaration de versement en cas de paiement par chèque ou par virement.

Ensuite, le comptable encaisseur envoie les bordereaux de versement au comptable assignataire qui traite, à la fin de la journée, les chèques en émettant leurs bordereaux à BAM (Bank Al Maghrib).

Le comptable assignataire traite les bordereaux de versement et envoie, après validation, les certificats de recettes à l'ordonnateur. Ce dernier, après validation, envoie au comptable assignataire l'ordre de régularisation[27].

[27] Présentation : système d'information GIR, Rabat, le 18 mars 2010

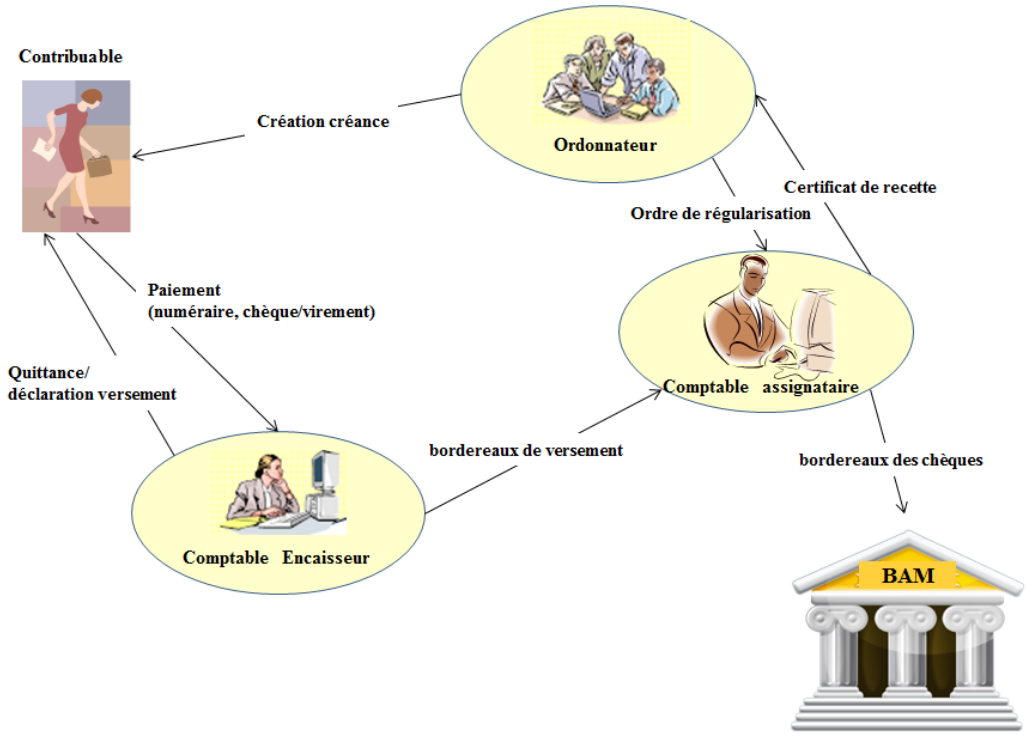


Figure 5-7 Gestion des recettes : avant la mise en place du projet GIR [MEF-Docs, 2012]

Cependant, de nombreux problèmes surgissent :

- Au niveau des systèmes d'information : les disparités importantes dans l'utilisation des TIC par les intervenants dans la chaîne de la recette, le peu d'intégration des systèmes d'information, et l'absence de coordination.
- Au niveau métier : la redondance et l'inefficacité des contrôles, l'absence de système d'évaluation de l'efficacité des services (indicateurs de performance et allocation des ressources), et le retard dans la mise à disposition auprès des organes constitutionnels de contrôles, des outils nécessaires à l'exercice de leurs missions.

Cette situation a eu pour conséquences l'allongement des coûts et des délais des encaissements de recettes, la mobilisation de ressources importantes pour la gestion de la recette au détriment des vrais métiers de chaque administration, et la multiplication des coûts d'acquisition et de maintenance des équipements informatiques.

Dans le cadre de la modernisation de la Trésorerie Générale du Royaume, visant notamment la rationalisation et l'optimisation de la gestion des finances publiques, il a été décidé de développer, un système de Gestion Intégrée des Recettes (cf. Figure 5-8), contenant une seule application et une seule base de données.

La Gestion Intégrée des Recettes est un système centralisé et unifié qui doit couvrir :

- toutes les natures de recettes;
- tous les intervenants (Ordonnateurs de l'Etat, Collectivités locales et comptables) ;

- tout le processus de gestion des recettes, depuis sa constatation chez l'ordonnateur jusqu'à son encaissement et son imputation budgétaire par le comptable assignataire ;
- toutes les composantes budgétaires (budget général, comptes spéciaux du trésor, services de l'état gérés de manière autonome).

La mise en œuvre de ce nouveau système permettra, notamment, de réduire les délais de traitement d'actes de la recette, d'optimiser les coûts de traitement des actes, de disposer en temps réel de l'information, d'améliorer la qualité de service rendu aux citoyens, de renforcer l'efficacité et l'efficience du recouvrement, de simplifier et optimiser les procédures sous-tendant la gestion des recettes et de répondre aux attentes des intervenants. Pour les ordonnateurs, le système GIR offrira une efficacité du recouvrement, une visibilité sur les recettes et une célérité d'imputation. Pour les comptables, il offrira l'efficacité, l'efficience, l'intégration et l'ergonomie du SI. Enfin, pour le redevable (contribuable), il permettra une banalisation de l'encaissement et une consolidation des créances.

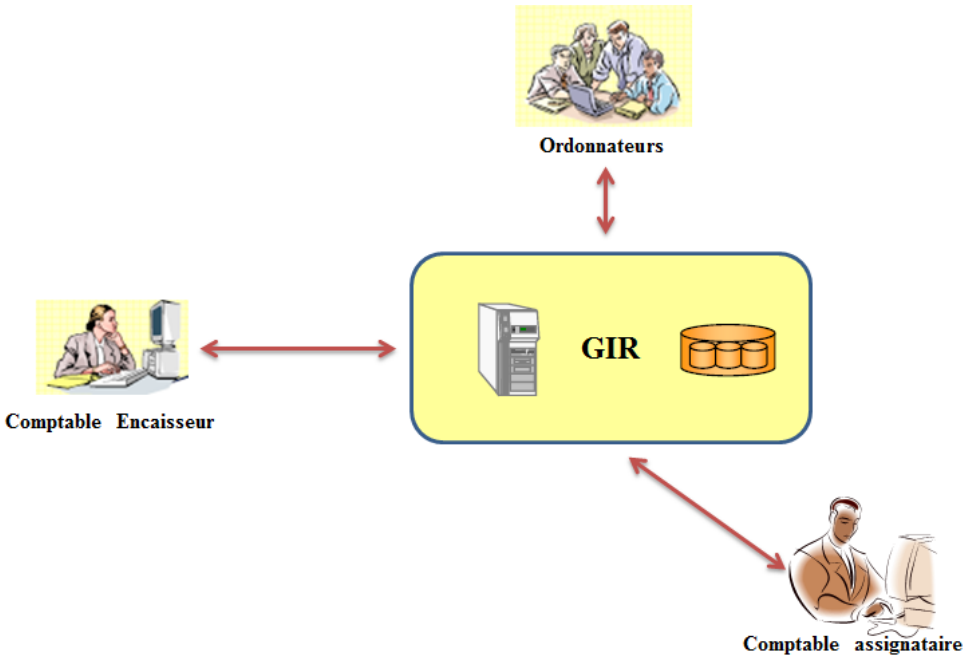


Figure 5-8 Gestion Intégrée de la Recette [MEF-Docs, 2012]

Les bénéfices escomptés du GIR seront la réduction significative des erreurs induites par la saisie multiple des mêmes informations, la communication rapide des informations entre les partenaires (sans investissement important de leur part) et une administration centralisée des échanges de données : routage, traçabilité des flux, sécurité, etc.

Conçu pour être fortement paramétrable afin de gérer des recettes d'une multitude de nature, le système se développe à l'instar du système de Gestion Intégrée de la Dépense (GID)[28]. Le projet de réalisation a procédé par réutilisation d'un ensemble de modules de l'application GID auxquels vont se rajouter des fonctionnalités propres à GIR. Ces dernières vont être implémentées en s'inspirant des applications existantes.

[28] Trésorerie Générale infos, Lettre d'information trimestrielle Octobre-décembre 2010.

La solution GIR est en cours de réalisation. Pour le développement de l'application résultante, nous proposons de réutiliser les applications plutôt que de les développer à nouveau. En effet, la composition des applications existantes permettra de produire l'application résultante. Nous proposons une composition non pas au niveau des applications vu leur complexité (différentes plate-forme, codes complexes à lire, etc.), mais au niveau des modèles, indépendamment des détails techniques des plates-formes d'exécution. La génération automatique du code permettra de produire l'application résultante. Notons qu'il existe, actuellement, des outils de génération de code à partir de modèles tel que l'outil Acceleo[29].

Le coût de cette solution est de modéliser les applications existantes. Néanmoins, l'acquisition des modèles permet de s'inscrire dans une approche MDA et, ainsi, de bénéficier de tous ces avantages. En d'autres termes, dans le cas où des fonctionnalités doivent être rajoutées, modifiées ou mises à jour au niveau des applications, il faudrait intervenir au niveau des modèles et non pas au niveau des applications.

5.1.2.4.1. Modèles sources : Cas du GIR

Nous avons créé les modèles en utilisant le reverse engineering (rétro-ingénierie) à partir des applications « Perception » contenant une base de données de 550 tables (Modèle M1), et « Recette Budgétaire » contenant une base de données de 57 tables (Modèle M2).

Le modèle M1 est un extrait du modèle de l'application « Perception ». Un Chef ordonnateur est caractérisé par un login, son code et l'intitulé du poste. Son rôle est de créer des créances. Une créance a un numéro, un montant, un type de paiement (numéraire, virement, chèque), une date de constatation et une date d'exigibilité. Elle peut être liée à un chèque possédant un numéro de chèque, un état (payé, non encore payé, rejeté), date d'opération, le code de la banque à laquelle il appartient, le code de la ville de la banque et le montant du chèque.

Une créance est payée par un contribuable, caractérisé par son nom, prénom, CIN, adresse, et numéro de téléphone. Un collègue (terme utilisé pour exprimer le comptable encaisseur) a pour rôle d'encaisser la créance. Il est caractérisé par son code et l'intitulé du poste. Plusieurs collègues peuvent disposer d'un compte caractérisé par un code et un titre. Un compte est lié à une loi de finance qui possède son code économique, son code fonctionnel, son titre et l'année de la loi de finance (*exerc*).

Le modèle M2 est un extrait du modèle de l'application « Recette Budgétaire ». Un comptable est caractérisé par son code et l'intitulé du poste. Un comptable peut être un encaisseur ou un assignataire. Plusieurs encaisseurs disposent d'un compte, caractérisé par un numéro et un intitulé.

Un compte est associé à une rubrique budgétaire, qui possède un code, un intitulé et l'année de la loi de finance (*exercice)*. Le rôle du comptable encaisseur est d'encaisser la recette payée par le redevable. Celui-ci est caractérisé par son nom, prénom, CIN, adresse, numéro de téléphone, et son email.

Une recette a un numéro, un montant et un type de paiement (numéraire, virement, chèque). Elle peut être liée à un chèque possédant un numéro de chèque, un état (payé, non encore payé, rejeté), la date du chèque, la banque à laquelle il appartient, le code de la ville de la banque, le montant du chèque, le compte de l'émetteur, le compte du récepteur, et la date d'envoi pour encaissement.

[29] http://www.acceleo.org/pages/accueil/fr

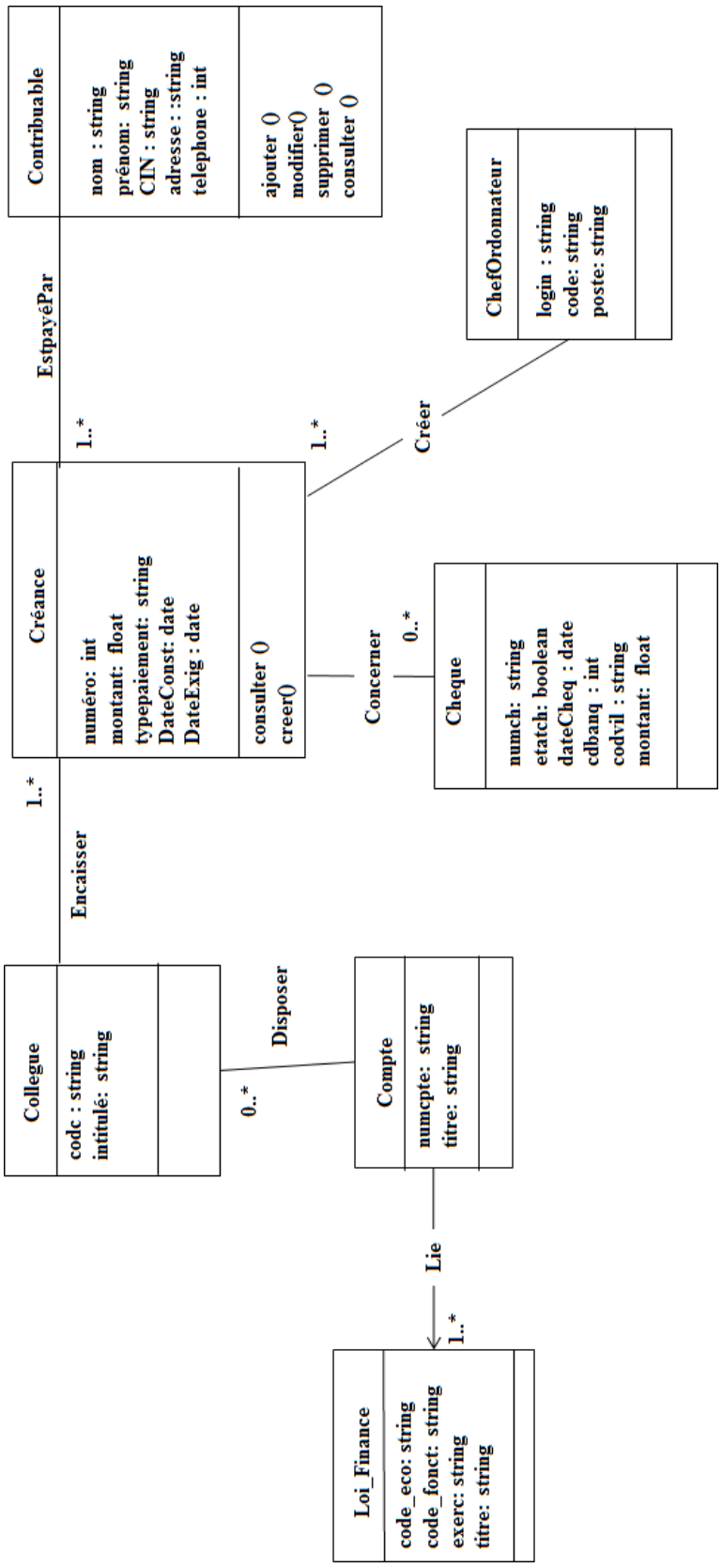


Figure 5-9 Cas du GIR : Modèle M1

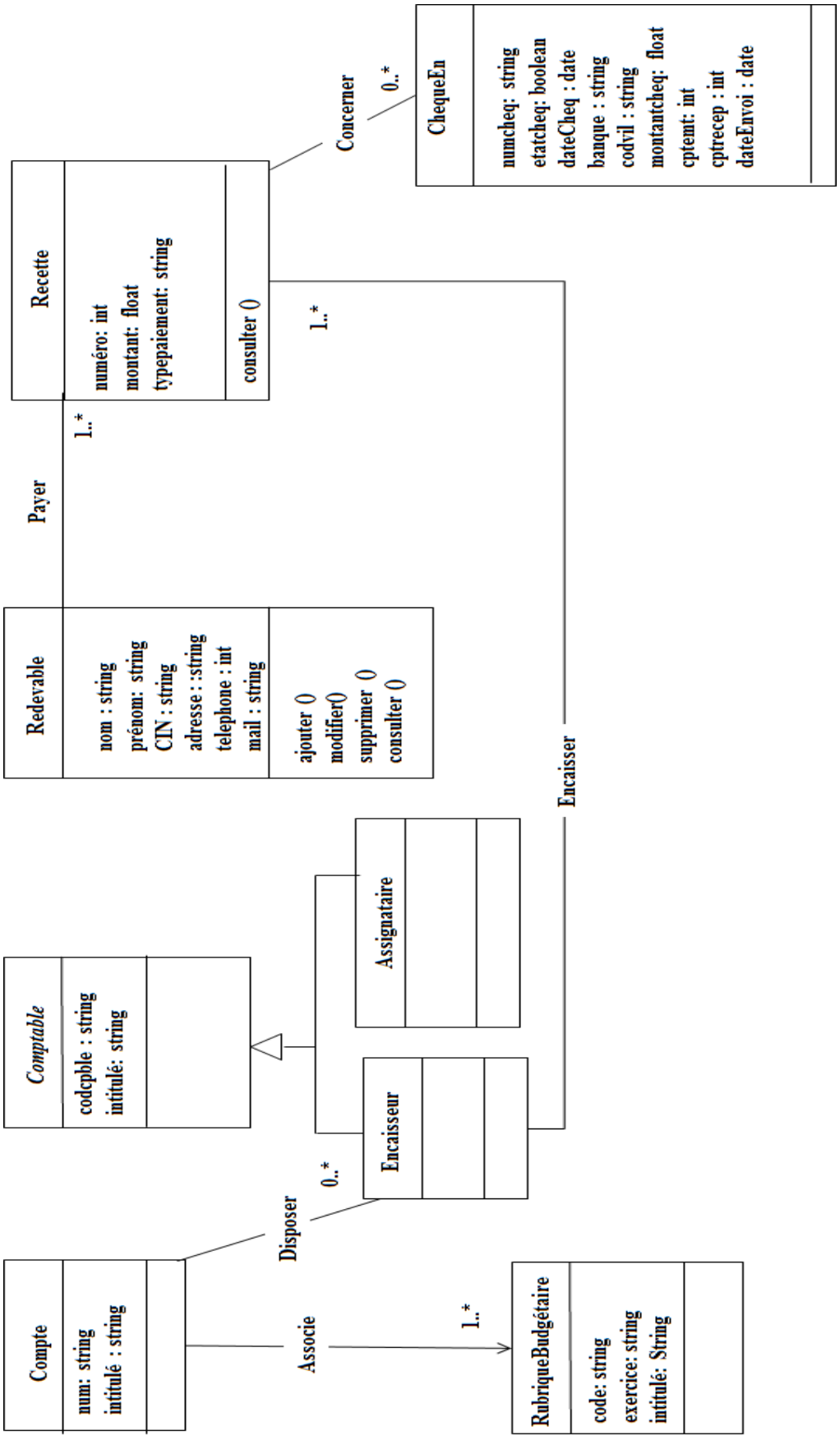


Figure 5-10 Cas du GIR : Modèle M2

5.1.2.4.2. Ontologie de domaine

L'ontologie du domaine utilisée, exprimée en OWL et sa représentation graphique sont présentées en annexe 2. L'ontologie dans ce cas d'étude se compose de :

- **Classes**: « Comptable », « Encaisseur », « Assignataire », « Chèque », « Créance », « Redevable», « RubriqueBudgetaire », « Ordonnateur », « Compte », « Collegue », « Contribuable », « LoiFinance », « Recette ».
- **Propriétés d'objet** :
  - Le rôle du comptable encaisseur est d'encaisser la recette.
  - Un comptable encaisseur dispose d'un compte.
  - Un compte est associé à une loi de finance
  - Le redevable paye la recette
  - Une recette est liée à un chèque
  - Le rôle de l'ordonnateur est de créer des créances.
- **Relation de subsomption** : Un comptable « Comptable » peut être un encaisseur « Encaisseur » ou un assignataire « Assignataire ».
- **Propriétés de données** :
  - Un comptable est caractérisé par son code et l'intitulé du poste.
  - Un compte est caractérisé par un code et un libellé.
  - Une rubrique budgétaire possède un code économique, un code fonctionnel, un intitulé et l'année de la loi de finance.
  - Un redevable est caractérisé par son nom, prénom, CIN, adresse, numéro de téléphone, et son email.
  - Une recette a un numéro, un montant, un type de payement, une date d'exigibilité et une date d'émission.
  - Un chèque possède un numéro, un état, la date d'émission du chèque, la banque à laquelle il appartient, le code de la ville de la banque, le montant du chèque, le compte de l'émetteur, le compte du récepteur, et la date d'envoi pour encaissement.
  - Un ordonnateur est caractérisé par un login, son code et l'intitulé du poste.
- **Relation d'équivalence entre les classes**
  - « Encaisseur » et « Collegue »,
  - « RubriqueBudgetaire » et « LoiFinance »,
  - « Recette » et « Créance »,
  - « Redevable » et « Contribuable »
- **Relation d'inverse entre les propriétés d'objet**
  - « Payer » et « EstPayéPar »
- **Relation d'équivalence entre les propriétés d'objet**
  - « lie » et « associe ».

5.1.2.4.3. Equivalences trouvées après la comparaison

Comme nous l'avons présenté pour la première étude de cas, le Tableau 5-5 présente le résultat de l'application du processus de comparaison des deux modèles M1 et M2 du système GIR.

M1	M2	Explication
Classes		
Collegue	Comptable	S
Compte	Compte	Syn(Identité)+LT+GT
Cheque	ChequeEn	Syn(Inclusion)+LP+GT
Collegue	Encaisseur	Sem(Equiv Onto)+ LT +GP
Loi_Finance	RubriqueBudgetaire	Sem(Equiv Onto)+LP+GT
Créance	Recette	Sem(Equiv Onto)+LP+GP
Contribuable	Redevable	Sem(Equiv Onto)+LP+ GT
Attributs et opérations		
Compte.numcpte	Compte.num	Syn(Inclusion)+ L
Compte.titre	Compte.intitulé	Sem(Synonyme)+ L
Collegue.codc	Comptable.Encaisseur.codcpble	Syn(Inclusion)+ L
Collegue.intitulé	Comptable.Encaisseur.intitulé	Syn(Identité)+ L
Loi_Finance.code_eco	RubriqueBudgétaire.code	Syn(Inclusion)+ L
Loi_Finance.exerc	RubriqueBudgétaire.exercice	Syn(Inclusion)+ L
Loi_Finance.titre	RubriqueBudgétaire.intitulé	Sem(synonyme)+local
Créance.numéro	Recette.numéro	Syn(Identité)+ L
Créance.montant	Recette.montant	Syn(Identité)+ L
Créance.typepaiement	Recette.typepaiement	Syn(Identité)+ L
Créance.consulter()	Recette.consulter()	Syn (Identité)+ L
Cheque.numch	ChequeEn.numcheq	Syn+(Inclusion)+ L
Cheque.etatch	ChequeEn.etatcheq	Syn(Inclusion)+ L
Cheque.dateCheq	ChequeEn.dateCheq	Syn(Identité)+ L
Cheque.codvil	ChequeEn.codvil	Syn(Identité)+ L
Cheque.montant	Cheqenc.montantcheq	Syn(Inclusion)+ L
Contribuable.nom	Redevable.nom	Syn(Identité)+ L
Contribuable.prénom	Redevable.prénom	Syn(Identité)+ L
Contribuable.CIN	Redevable.CIN	Syn(Identité)+ L
Contribuable.adresse	Redevable.adresse	Syn(Identité)+ L
Contribuable.telephone	Redevable.telephone	Syn(Identité)+ L
Contribuable.ajouter ()	Redevable.ajouter ()	Syn(Identité)+ L
Contribuable.modifier()	Redevable.modifier()	Syn(Identité)+ L
Contribuable.supprimer ()	Redevable.supprimer ()	Syn(Identité)+ L
Contribuable.consulter ()	Redevable.consulter ()	Syn(Identité)+ L
Associations et Relations de généralisation		
Compte.Loi_Finance.Lie	Compte. RubriqueBudgétaire.Associe	Sem(Equiv Onto)
Compte.Collegue.Disposer	Compte.Encaisseur.Disposer	Syn(Identité)
Collegue.Créance.Encaisser	Encaissseur.Recette.Encaisser	Syn(Identité)
Créance.Contribuable.EstPayéPar	Recette.Redevable.Payer	Sem(Inverse)
Créance.Cheque.Concerner	Recette.ChequeEn.Concerner	Syn(Identité)

Tableau 5-5 Composition des modèles pour GIR : Comparaison

5.1.2.4.4. Mapping d'équivalence

Le Tableau 5-6 présente le mapping d'équivalence après le paramétrage de l'utilisateur. Il s'agit des choix des options LP, GP ou LP+GP qui donnent le même résultat pour ce cas de modèles. Les

autres choix s'avèrent peu intéressants, du fait que le modèle résultat comporterait plusieurs classes dupliquées. En effet, pour les choix :

- LT+GT : seules les classes « Compte » de M1 et « Compte » de M2 seront considérées équivalentes.
- LT ou LT+GP : seules les classes (« Compte » de M1 et « Compte » de M2) et (« Encaisseur » de M1 et « Collegue » de M2) seront considérées équivalentes.
- GT ou LP+GT : les classes « Encaisseur » de M1, « Collegue » de M2, « Créance » de M1 et « Recette » de M2 seront considérées isolées.

M1	**M2**
Compte	Compte
Cheque	ChequeEn
Collegue	Encaisseur
Loi_Finance	RubriqueBudgetaire
Créance	Recette
Contribuable	Redevable
Compte.numcpte	Compte.num
Compte.titre	Compte.intitulé
Collegue.codc	Comptable.Encaisseur.codcpble
Collegue.intitulé	Comptable.Encaisseur.intitulé
Loi_Finance.code_eco	RubriqueBudgétaire.code
Loi_Finance.exerc	RubriqueBudgétaire.exercice
Loi_Finance.titre	RubriqueBudgétaire.intitulé
Créance.numéro	Recette.numéro
Créance.montant	Recette.montant
Créance.typepaiement	Recette.typepaiement
Créance.consulter()	Recette.consulter()
Cheque.numch	ChequeEn.numcheq
Cheque.etatch	ChequeEn.etatcheq
Cheque.dateCheq	ChequeEn.dateCheq
Cheque.codvil	ChequeEn.codvil
Cheque.montant	Cheqenc.montantcheq
Contribuable.nom	Redevable.nom
Contribuable.prénom	Redevable.prénom
Contribuable.CIN	Redevable.CIN
Contribuable.adresse	Redevable.adresse
Contribuable.telephone	Redevable.telephone
Contribuable.ajouter ()	Redevable.ajouter ()
Contribuable.modifier()	Redevable.modifier()
Contribuable.supprimer ()	Redevable.supprimer ()
Contribuable.consulter ()	Redevable.consulter ()
Compte.Loi_Finance.Lie	Compte. RubriqueBudgétaire.Associe
Compte.Collegue.Disposer	Compte.Encaisseur.Disposer
Collegue.Créance.Encaisser	Encaissseur.Recette.Encaisser
Créance.Contribuable.EstPayéPar	Recette.Redevable.Payer
Créance.Cheque.Concerner	Recette.ChequeEn.Concerner

Tableau 5-6 Composition des modèles pour GIR : Mapping d'équivalence

5.1.2.4.5. Eléments isolés

Les éléments isolés sont représentés dans le Tableau 5-7 ci-dessous.

Classes	Attributs et Opération	Association et Relation de généralisation
Comptable	Loi_Finance.code_fonct	Créance.Ordonnateur.Créer()
Assignataire	Créance.DateExig	RG(Comptable,Encaisseur)
ChefOrdonnateur	Créance.creer()	RG(Comptable,Assignataire)
	Cheque.cdbanq	
	ChefOrdonnateur.login	
	ChefOrdonnateur.code	
	ChefOrdonnateur.poste	
	ChequeEn.banque	
	ChequeEn.cptemt	
	ChequeEn.cptrecep	
	ChequeEn.DateEnvoi	
	Créance.dateConst	

Tableau 5-7 Composition des modèles pour GIR : Eléments isolés

5.1.2.4.6. Mapping d'hyponyme

Il n'existe pas de classes hyponymes dans ces deux modèles.

5.1.2.4.7. Modèle résultat : cas du GIR

Le modèle résultat des deux modèles M1 et M2 est représenté dans la Figure 5-11. Il représente le cas où l'utilisateur du système de composition de modèles a choisi comme modèle principal le modèle par défaut (M2 car il comporte le plus grand nombre de classes). Rappelons que les éléments fusionnés sont en rouge et ceux translatés sont en bleu.

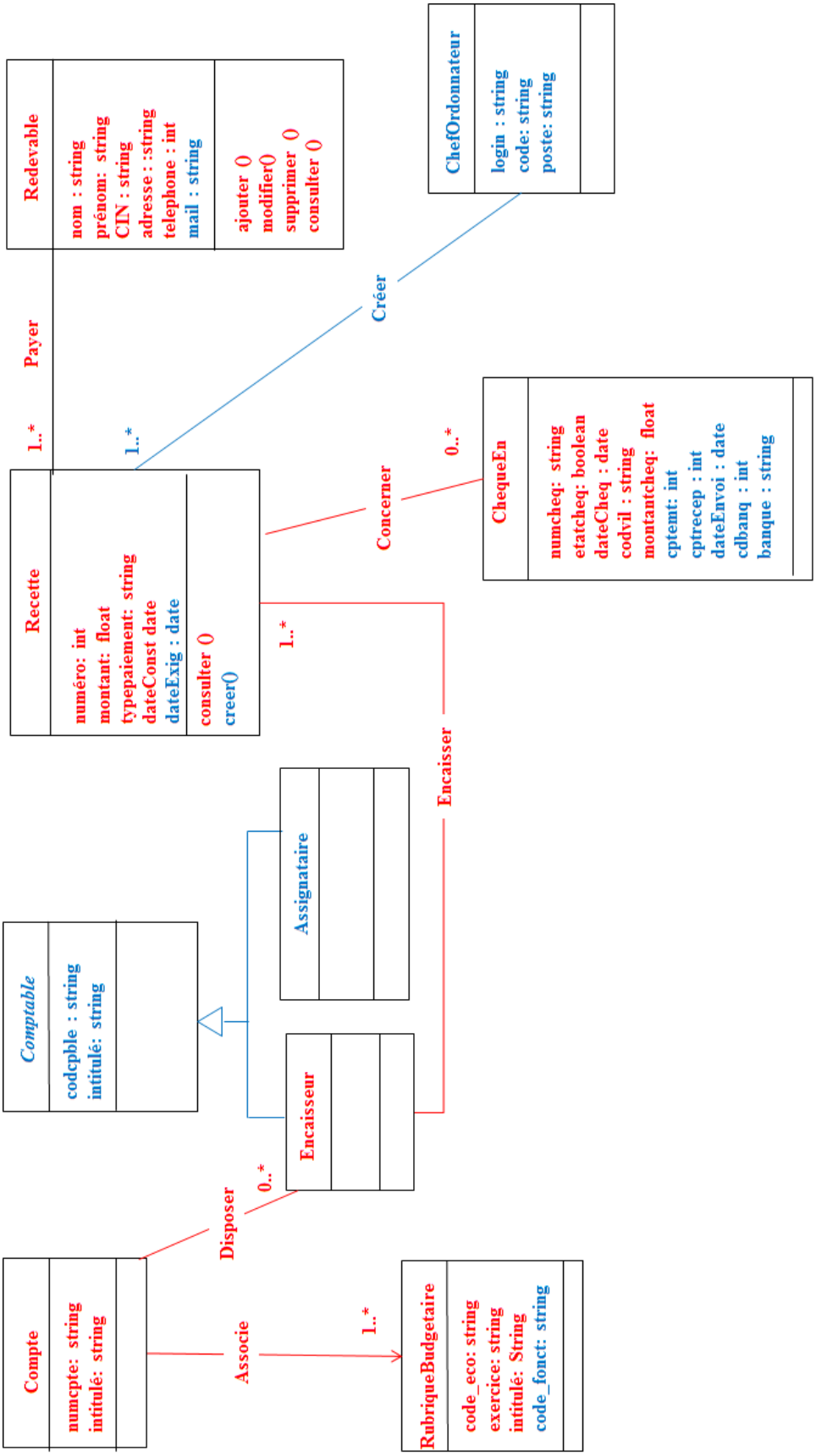


Figure 5-11 Composition des modèles pour GIR : Modèle résultat

Les classes « Compte », « RubriqueBudgetaire », « Recette », « Redevable », « ChequeEn » et « Encaisseur » ont été créées suite à la fusion de classes équivalentes. Les classes restantes « Assignataire », « Comptable » et « ChefOrdonnateur » ont été translatées dans le modèle résultat du fait qu'elles sont isolées.

Dans les classes fusionnées, on retrouve les attributs/opérations fusionnés et ceux translatés. Les classes isolées ont été translatées contenant leurs attributs/opérations isolés.

Comme est expliqué dans la règle de fusion de classes, les attributs présents dans M2 de la classe « Encaisseur » (hérités de la classe « Compatble ») étant équivalents aux attributs de la classe « Collegue » du modèle M1, n'ont pas été fusionnés, même s'ils apparaissent dans le mapping d'équivalence. Les attributs « codcpble », et « intitulé » de la classe « Comptable » de M2 ont été translatés donc lors de la translation de « Comptable ».

L'association « Créer », reliant « Recette » et « ChefOrdonnateur » a été créée suite à une translation. Les autres associations « Associe », « Disposer », « Encaisser », « Concerner », et « Payer » ont été créées suite à une fusion.

Les relations de généralisation reliant « Comptable » et « Encaisseur », et « Comptable » et « Assignataire » ont été rajoutées dans le modèle résultat, car elles sont isolées.

Nous avons présenté les deux études de cas : bancaire et GIR. Nous présentons dans la suite du chapitre l'implémentation de notre approche ainsi que son évaluation.

5.2. Implémentation

Cette section présente les différents éléments qui caractérisent le développement de notre prototype.

5.2.1. Architecture du prototype

L'architecture générale du prototype Co-Models comprend deux principaux modules (Figure 5-12) : un module de composition de modèles et un module comportant un convertisseur du format UML vers XMI et du format XMI vers UML.

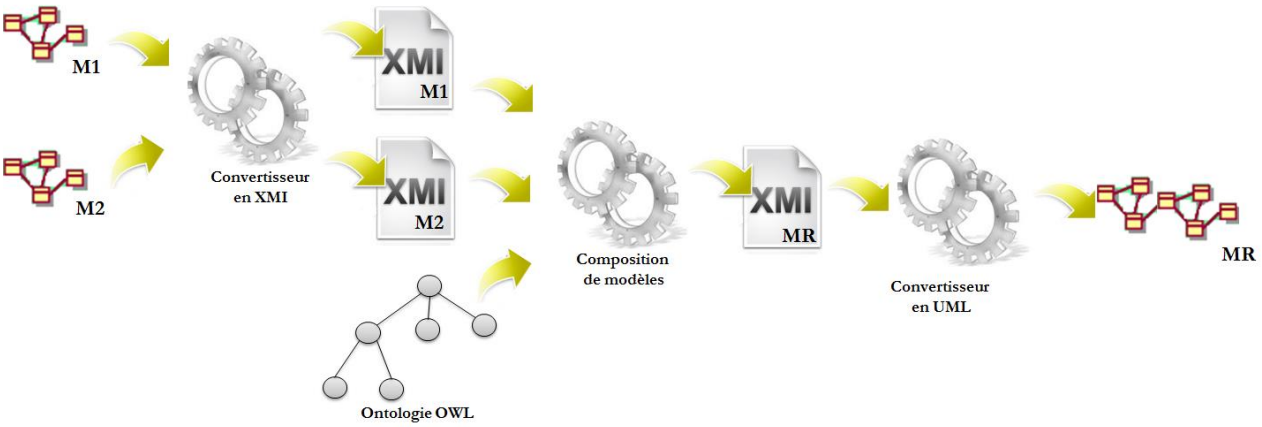


Figure 5-12 Architecture du prototype

Co-Models gère des modèles (d'entrée et de sortie) exprimés en XMI [XMI, 2001]. Pour ce faire, le convertisseur transforme les deux modèles UML en entrée vers des modèles en XMI. Le module de composition de modèles prend en entrée les deux modèles en format XMI et l'ontologie OWL et produit en sortie le modèle résultat en format XMI. Le convertisseur transforme ce dernier vers un modèle UML.

5.2.2. Outils utilisés

Co-Models a été développé en Java, car ce langage permet l'utilisation de plusieurs APIs pour manipuler les ontologies OWL tel que JENA. La conversion entre les types « modèle orienté objet (moo) » et « xmi » se fait par l'outil PowerAMC.

La représentation et la manipulation des documents XMI sont assurées par l'API JDOM. Nous avons adopté Eclipse, pour le développement de Co-Models. Nous décrivons, dans ce qui suit, Eclipse ainsi que les deux APIs JENA et JDOM.

5.2.2.1. Eclipse

Eclipse[30] est un environnement de développement intégré libre (le terme Eclipse désigne également le projet correspondant, lancé par IBM) extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement, mettant en œuvre n'importe quel langage de programmation. Eclipse est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions. La spécification d'Eclipse vient de son architecture, totalement développée autour de la notion de plug-in.

Plusieurs logiciels commerciaux sont basés sur ce logiciel libre, comme IBM Lotus Notes 8, IBM Symphony ou Websphere Studio Application Developer.

5.2.2.2. API JDOM

JDOM[31] est une API open source Java dont le but est de représenter et manipuler un document XML de manière intuitive pour un développeur Java sans requérir une connaissance pointue de XML. Le but de JDOM n'est pas de définir un nouveau type de parseur mais de faciliter la manipulation au sens large de document XML, telles que la lecture d'un document, la représentation sous forme d'arborescence, la manipulation de cet arbre, la définition d'un nouveau document, l'exportation vers plusieurs formats cibles, etc.

JDOM est donc un modèle de documents objets open source, dédié à Java, pour encapsuler un document XML. JDOM propose, aussi, une intégration de SAX, DOM, XSLT et XPath.

5.2.2.3. API JENA

JENA [32] est un framework écrit en Java, dont l'objectif est de fournir un environnement facilitant le développement d'applications dédiées au web sémantique. JENA permet de manipuler des documents RDF, RDFS et OWL, et fournit, en plus, un moteur d'inférences permettant des raisonnements sur les ontologies.

[30] http://www.eclipse.org/
[31] http://www.jdom.org/
[32] http://jena.apache.org/

5.2.3. Co-Models

Nous présentons, dans cette section, notre prototype. L'interface comporte les boutons « Parcourir » pour déterminer les chemins des fichiers XMI des deux modèles à composer, et choisir l'ontologie de domaine (cf. Figure 5-13). Un espace de paramétrage est dédié à l'utilisateur pour qu'il choisisse le modèle principal ainsi que l'option d'équivalence (Figure 5-14).

Le modèle principal choisi par défaut est celui qui contient le plus grand nombre de classes, et l'option d'équivalence par défaut est l'équivalence en structure locale partielle et en structure globale partielle.

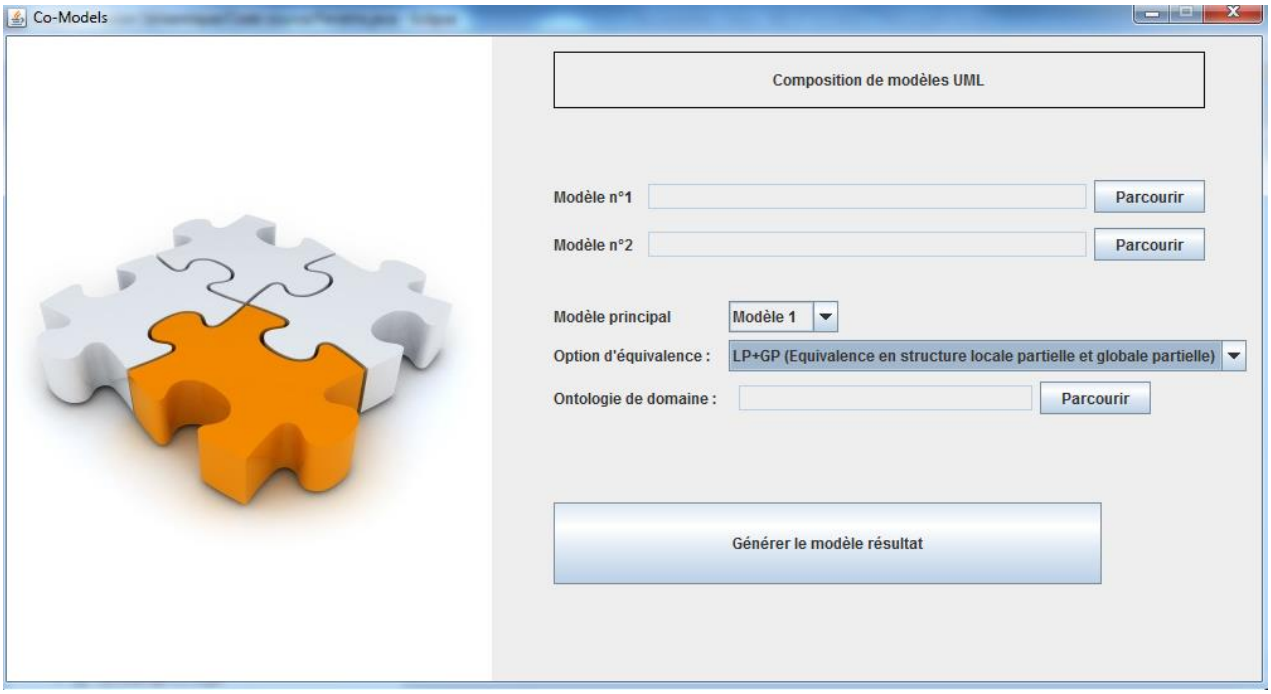


Figure 5-13 Interface de Co-Models

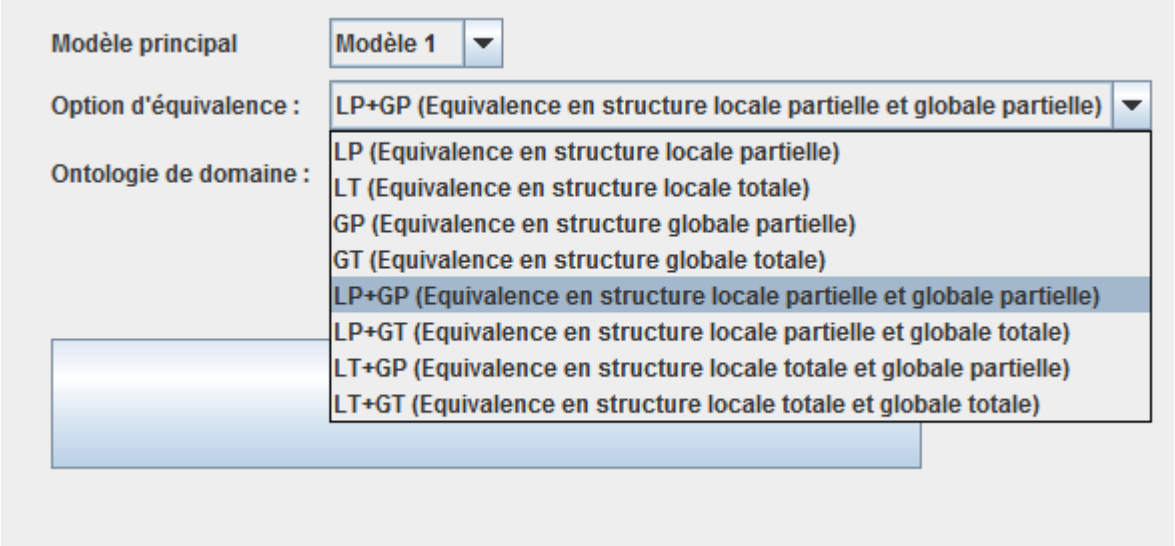


Figure 5-14 Co-Models : Choix de l'option d'équivalence

Le déroulement de la composition des deux modèles de l'étude de cas du GIR, par le prototype Co-Models, est présenté dans l'annexe 3.

5.3. Evaluation de Co-Models

5.3.1. Cadre et objectifs des expérimentations

Dans le but d'évaluer nos propositions en matière de composition de modèles, nous avons effectué des expérimentations (cf. Annexe 4).

Ces expérimentations visent deux principaux objectifs. Le premier est d'évaluer dans quelle mesure notre approche se rapproche du raisonnement humain. Le second objectif est de connaitre les avis des utilisateurs sur Co-Models.

Pour répondre au premier objectif, nous avons demandé aux sujets, dans une première partie de l'expérimentation, de réaliser la composition de modèles manuellement, afin de comparer le modèle résultat avec celui produit automatiquement par Co-Models. En ce qui concerne le second objectif, la deuxième partie de l'expérimentation a été de fournir un questionnaire d'évaluation auquel l'utilisateur répondra, après avoir utilisé Co-Models.

- **Profil des participants**

Les sujets participants à ces expérimentations sont décrits dans le Tableau 5-8. Le groupe est constitué de 10 personnes dont 9 ingénieurs logiciels et un enseignant en informatique. Toutes les personnes ont déjà utilisé des modèles UML et ont au moins des notions en ontologie. Cinq d'entre elles font partie des équipes qui ont déjà réalisé une composition de modèles. Cette dernière a été réalisée manuellement.

Profil	9 Ingénieurs en Informatique (logiciel) 1 Enseignant en Informatique
Sexe	6H 4F
Moyenne d'âge	31 ans
Utilisation des modèles UML	10/10
Pratique de la composition de modèles	5/10
Connaissance des ontologies	Notions

Tableau 5-8 Profil des sujets

Nous avons établi un ensemble de critères, concernant les deux principaux objectifs de cette expérimentation :

- **Critères de la 1ère partie de l'expérimentation:**
  - **Type de comparaison syntaxique et sémantique:** ce critère permet de déterminer les types de comparaison syntaxique et sémantique utilisés pour comparer les modèles sources. Par exemple l'inclusion et l'acronymie pour le type de comparaison syntaxique, synonymie, polysémie, inverse et disjonction pour le type de comparaison sémantique.
  - **Type de comparaison de nature :** ce critère permet de déterminer si la comparaison est effectuée même pour les éléments de type différents.

- **Type de comparaison structurelle locale et globale** : ce critère permet de déterminer les types de comparaison structurelle (LP, GP, LT, GT, LT+GT, LP+GP, LT+GP, GT+LP) sur lesquels la comparaison s'est basée pour considérer que deux classes sont équivalentes.
- **Modèle principal** : ce critère permet de préciser si un modèle principal a été choisi dans le but d'y prendre les noms du modèle résultat.
- **Temps :** ce critère permet de mesurer le temps de la composition de modèles.

- **Critères de la 2ème partie de l'expérimentation:**
  - **Utilité :** ce critère permet de mesurer l'utilité du système, en demandant si Co-Models est pas du tout utile, moyennement utile ou très utile.
  - **Simplicité d'utilisation :** ce critère permet de mesurer le niveau de satisfaction de la simplicité d'utilisation du système (simple, moyennement simple ou pas simple)
  - **Volonté d'utilisation :** ce critère permet de préciser la volonté du participant à utiliser Co-Models au lieu de composer manuellement.
  - **Temps :** ce critère permet de mesurer à quel point l'utilisateur estime gagner du temps.

5.3.2. Evaluation du rapprochement de Co-Models avec le raisonnement humain

5.3.2.1. Résultats obtenus

Les données produites à l'issue des expérimentations réalisées, concernent une représentation sur papier du modèle résultat, et un questionnaire d'évaluation. Nous présentons notre analyse en deux temps, tout d'abord les résultats obtenus sur la première partie de l'expérimentation, puis sur la deuxième partie.

- **Composition automatique par Co-Models**

Comme est expliqué dans le chapitre 4, Co-Models utilise différents types de comparaison. Pour la comparaison syntaxique, il prend en compte l'identité, l'inclusion totale, l'abréviation et l'acronymie. Et pour ce qui est de la comparaison sémantique, il détecte la synonymie, la disjonction, l'inverse, la polysémie et l'hyponymie.

Co-Models laisse la possibilité à l'utilisateur, de choisir dans quel cas considérer que deux classes de modèles peuvent être mises en correspondance. Ce paramétrage comporte huit options : LP, GP, LT, GT, LT+GT, LP+GP, LT+GP, GT+LP.

Co-Models utilise un modèle principal afin de s'y référer pour prendre les noms des éléments résultat, en cas de différents entre les éléments sources. Le temps de composition de Co-Models est estimé à une minute, pour une composition de deux modèles comportant respectivement 9 et 10 classes.

Co-Models ne prend pas en compte l'hétérogénéité de nature. Par exemple « Versement » est représenté sous forme de classe dans M2 et sous forme d'attribut dans M1 : « type » de la classe « Opération ». Dans le modèle résultat, l'incohérence provient du fait que la classe « Opération » contient l'attribut « type » et la sous-classe « Versement ». De plus, Co-Models ne permet pas de prendre en compte l'inclusion partielle.

- **Composition manuelle réalisée par les participants**

Tous les participants (10/10) ont eu recours aux types de comparaison syntaxiques et sémantiques afin de trouver les correspondances. Ils ont utilisé l'inclusion, l'abréviation, l'acronymie, la synonymie, la disjonction et l'inverse. Un d'entre eux n'a pas détecté une relation d'hyponymie, et trois d'entre eux n'ont pas détecté la polysémie entre deux classes. 4 participants sur 10 ont pris en compte le type de comparaison de nature. Tous les participants (10/10) ont jugé que des classes sont équivalentes si elles sont équivalentes en structure locale partielle et en structure globale partielle.2 participants sur 10 ont choisi de garder les noms d'un seul modèle, contre 8 qui ont combiné les noms des éléments des deux modèles.

Les modèles produits par les participants contiennent exactement les mêmes classes que le modèle produit par Co-Models. Cependant, 4 participants sur 10 ont produit une composition (agrégation) en résultat d'une composition et une association ; 9 participants sur 10 ont produit une navigabilité en résultat d'une navigabilité et une non navigabilité ; et 2 participants sur 10 ont produit de fausses multiplicités.Le temps moyen de composition manuelle de deux modèles est de 40 minutes pour des modèles contenant respectivement 9 et 10 classes. Ce temps augmentera évidemment avec l'augmentation du nombre de classes.

5.3.2.2. Analyse des résultats

Nous présentons dans le Tableau 5-9 une synthèse des résultats obtenus, à partir de la première partie de l'expérimentation. La première colonne représente les critères de comparaison. La deuxième concerne les résultats obtenus pour Co-Models. Et la troisième représente le nombre de participants ayant satisfait le critère. Nous présentons dans une dernière ligne du tableau les limites de la composition par Co-Models et celle par les participants.

Critères			**Co-Models**	**Raisonnement humain (participants)**
Type de comparaison syntaxique et sémantique	Identité		Oui	10/10
	Inclusion	Totale	Oui	10/10
		Partielle	Non	10/10
	Abréviation		Oui	10/10
	Acronymie		Oui	10/10
	Synonymie		Oui	10/10
	Disjonction		Oui	10/10
	Inverse		Oui	10/10
	Polysémie		Oui	7/10
	Hyponymie		Oui	9/10
Type de comparaison de nature			Non	4/10
Type de comparaison structurelle locale et globale			LP+GP (par défaut)	10/10 LP+GP
Modèle principal			Oui	2/10
Temps			1 minute/(9classes en M1 et 10 classes en M2)	40 minutes/(9 classes en M1 et 10 classes en M2)
Limites			Non prise en compte de : - Hétérogénéité de nature - Inclusion partielle	Incohérences : - Composition (4/10) - Navigabilité (9/10) - Multiplicités (2/10)

Tableau 5-9 Résultats de la partie 1 de l'expérimentation

La détection des équivalences et des non-équivalences par Co-Models, suite à une comparaison, est identique à celle réalisée par les participants. Cependant, Co-Models ne prend pas en compte l'hétérogénéité de nature et l'inclusion partielle.

En effet, nous n'avons pas pu automatiser l'inclusion partielle, du fait que cela peut générer des erreurs d'un autre type. Prenons le cas de l'exemple suivant : dans un premier modèle, « matériel » et dans un deuxième « matière » ; dans le cas de la prise en compte de l'inclusion partielle, les deux attributs seront jugés équivalents, ce qui n'est pas le cas. Autrement dit, deux éléments avec des syllabes en communs, seront jugés équivalents.

Pour ce qui est du modèle produit manuellement, celui-ci contient des incohérences de navigabilité, de composition, et de multiplicité. Cela est dû probablement du fait que tous les participants n'ont pas une expertise sur la modélisation UML.

Co-Models permet, d'une part, un gain de temps considérable, et permet d'assurer la fusion des éléments présentés différemment dans les modèles cibles, avec par exemple des différences en multiplicité, composition et navigabilité. Choses qui font de Co-Models un outil intéressant sur lequel les utilisateurs pourraient s'appuyer pour la composition de modèles.

Le résultat de ces expérimentations nous a permis de déterminer la logique de comparaison adoptée par la majorité des participants et de choisir ainsi l'option par défaut du paramétrage de notre système : équivalence en structure locale partielle et structure globale partielle (LP+GP).

Nous précisons que si l'utilisateur choisit l'option d'équivalence locale totale et globale totale, les mapping et les éléments isolés auront une très grande probabilité d'être sûrs. Cependant, le système peut ne pas détecter des éléments pouvant être équivalents. Par exemple, deux classes qui ont le même nom, équivalentes en structure globale et possédant trois attributs sur quatre équivalents, seront considérées, dans ce cas de paramétrage, non équivalentes (isolées) et donc apparaîtront toutes les deux dans le modèle résultat.

En effet, selon nos expérimentations, dans une grande majorité des cas, les choix de paramétrage qui comportent « Equivalence Globale Totale » et/ou « Equivalence Locale Totale » font apparaitre un modèle résultat erroné, c'est-à-dire comportant plusieurs classes en double.

Nous déconseillons donc ce choix. Néanmoins, nous choisissons d'offrir la possibilité de ce choix uniquement dans le cas où (1) l'utilisateur du système sait au préalable que les classes sont équivalentes en structure locale totale et/ou globale ; (2) L'utilisateur veut considérer que deux classes qui n'ont pas exactement la même structure locale et/ou structure globale, soient différentes, c'est-à-dire qu'il veut composer uniquement les éléments qui sont exactement les mêmes.

5.3.3. Evaluation de Co-Models par les participants

L'analyse des données issues de la deuxième partie de l'expérimentation a donné lieu à des résultats illustrant la répartition des opinions selon les différentes rubriques à tester. Nous synthétisons dans le Tableau 5-10 ci-dessous ces opinions.

Critères	Points forts	Points faibles
Volonté d'utilisation	10/10	-
Utilité	Très utile (10/10)	-
Simplicité d'utilisation	Simple (8/10)	Moyennement simple (2/10)
Temps	Gain de temps significatif (10/10)	-

Tableau 5-10 Résultats de la partie 2 de l'expérimentation

Tous les participants ont exprimé leur volonté d'utiliser Co-Models par rapport à une composition manuelle. Les 10 participants ont jugé la composition de modèles avec Co-Models très utile. 8 participants ont jugé que Co-Models était simple à utiliser, contre 2 d'entre eux qui l'ont jugé moyennement simple. Les 10 participants estiment gagner un temps considérable en utilisant Co-Models.

L'analyse des résultats a révélé que Co-Models peut être utilisé pour une éventuelle composition de modèles, du fait qu'il a été jugé utile, facile (ou moyennement facile) à utiliser, et permet un gain significatif en terme de temps.

5.3.4. Conclusion

A l'issue de nos expérimentations, nous avons pu vérifier, selon un ensemble de critères, que Co-Models répond effectivement aux objectifs de notre thèse : une comparaison hybride incluant la sémantique pour se rapprocher du raisonnement humain, une fusion et translation correctes des éléments de modèles sources. Ainsi, Co-Models a permis d'obtenir un modèle résultat, avec un niminum de conflits.

Par conséquent, nous considérons que Co-Models est un outil intéressant, sur lequel, même un non-expert peut se baser, pour composer les modèles.

Conclusion

L'objectif de ce chapitre était de mettre en œuvre, de valider et d'évaluer l'ensemble des propositions présentées dans cette thèse. Pour ce faire, nous avons présenté deux études de cas sur lesquelles nous avons appliqué notre approche. Il s'agit dans le premier cas, de fusion de deux banques, et dans le deuxième cas, de la création du système de la Gestion Intégré des Recettes au Ministère de l'Economie et des Finances. Nous avons inclus, également, l'implémentation de notre approche qui s'est concrétisée par le prototype Co-Models. Enfin, nous avons présenté les résultats des expérimentations réalisées dans le but d'évaluer nos propositions.

Chapitre **6**

Conclusion générale

6. Conclusion générale

6.1. Rappel du cadre des travaux

Le contexte général de nos travaux porte sur la composition de modèles basée sur les ontologies. Nous nous sommes d'abord intéressés, d'une part, à l'Ingénierie Dirigée par les Modèles qui constitue le cadre général des modèles, méta-modèles et de transformation de modèles, et, d'autre part, à l'ontologie. Ensuite, nous avons porté l'essentiel de notre effort sur la composition de modèles. Nous avons commencé par présenter les différentes définitions de la composition de modèles, puis nous nous sommes ensuite penchés sur les différentes hétérogénéités pouvant exister entre deux modèles à composer. Ceci a été illustré par les deux modèles du domaine bancaire.

Nous nous sommes intéressés, par la suite, à l'étude des approches de composition de modèles existantes. Nous les avons classées selon des critères afin de les évaluer et de déterminer leurs limites. Nous avons constaté qu'aucune approche ne traite tous les types de comparaison, ce qui peut engendrer des conflits dans le modèle résultat.

La problématique abordée était donc : Comment réaliser la composition de modèles, afin d'avoir un modèle résultat en empêchant l'apparition de conflits ?

Afin de répondre à cette problématique, nous avons proposé une approche de composition qui a la particularité de combiner et de couvrir tous les types de comparaison afin de trouver de réelles correspondances et de réels éléments isolés. Cette approche a permis aussi de déterminer, comment les éléments présentés différemment dans les modèles sources, apparaîtront dans le modèle résultat.

6.2. Synthèse sur les apports de ces travaux

- **Approche hybride de composition de modèles**

L'apport principal de nos travaux réside dans la proposition d'une approche hybride de composition de modèles, comparant sémantiquement les modèles et couvrant tous les types de comparaison. Ceci avait pour objectif d'augmenter la probabilité de trouver de réelles correspondances et de réels éléments isolés. L'idée principale est d'utiliser une ontologie de domaine qui a pour objectif de fournir des informations relatives aux éléments des modèles sources, comme l'équivalence ou l'inverse de deux concepts.

- **Système Co-Models**

Nous avons concrétisé notre approche par un système de composition contenant deux sous-systèmes. Le premier est celui de la comparaison. Il s'agit d'un système à base de règles qui prend, en entrée, les deux modèles sources, et produit, en sortie, les mappings d'équivalence et d'hyponymie et les éléments isolés. Ce système se base sur une ontologie de domaine ainsi que d'autres ressources, tels que les dictionnaires, afin de couvrir tous les types de comparaison.

Le deuxième système est celui de la fusion/translation. Il prend en entrée les mappings, les éléments isolés, et les modèles sources et produit, en sortie, le modèle résultat. Ce système contient des règles de fusion et de translation. Ces directives de composition ont pour objectif de préciser comment les éléments sources vont apparaître dans le modèle résultat.

Une comparaison tenant compte de la sémantique des modèles, combinée aux directives de composition a permis d'empêcher l'apparition de conflits dans le modèle résultat. Il s'agit, plus précisément, d'avoir un modèle sans redondance et sans perte d'information.

Un processus de composition a été proposé afin de décrire le déroulement de la comparaison, la fusion et la translation des éléments de modèles sources.

Nous avons offert la possibilité à l'utilisateur de Co-Models, de choisir dans quel cas considérer que deux classes de modèles peuvent être mises en correspondance. Ceci permet d'avoir un outil modulable en fonction du besoin de l'utilisateur.

- **Règles de composition**

Nous avons présenté les règles de comparaison, de fusion et de translation de manière informelle, accompagnées d'exemples pour faciliter la compréhension. Pour éviter l'ambiguïté, et pour présenter d'une manière plus précise ces règles, nous avons proposé un cadre formel de notre approche, proposant sa formalisation en logique des prédicats. Notre démarche de formalisation a nécessité un ensemble de transformations qui se concrétisent en :

- Transformation d'un modèle UML (conforme au méta-modèle UML) vers un modèle logique (conforme au méta-modèle de la logique des prédicats)
- Transformation d'une ontologie (conforme au méta-modèle OWL) vers un modèle logique (conforme au méta-modèle de la logique des prédicats)

- **Validation de notre approche : Etudes de cas**

Nous avons appliqué notre approche sur deux études de cas, portant sur les domaines bancaire et du e-gouvernement. Le premier cas d'étude s'inscrit dans la tendance mondiale de la fusion de deux banques. Des difficultés de restructuration du nouveau système d'information en résultent. En effet, la fusion de deux systèmes d'information est très complexe et nécessite beaucoup d'efforts du fait de l'hétérogénéité des processus, des fonctionnalités, des applications et ainsi des modèles. Nous avons donc proposé d'utiliser notre approche afin de faciliter la composition de modèles.

La deuxième étude de cas concerne la création du système de Gestion Intégrée des Recettes (GIR), un projet en cours de réalisation par le Ministère de l'Economie et des Finances au Maroc. D'après les ingénieurs logiciels travaillant directement sur ce projet, la démarche actuelle de la réalisation du système GIR est de réutiliser un ensemble de modules du projet existant GID (Gestion Intégrée de la Dépense) et d'y rajouter de nouvelles fonctionnalités propres à la gestion de la recette. Le développement des nouveaux modules se fait de zéro, en s'inspirant des applications existantes.

Notre proposition s'inscrit dans une approche MDA, c'est-à-dire ne plus agir au niveau des applications mais au niveau des modèles. En effet, la composition des applications existantes est une tâche complexe, du fait de la complexité : codes complexes à comprendre, plusieurs lignes, etc. Nous préconisons de composer les modèles des applications existantes, indépendamment des détails techniques des plates-formes d'exécution, afin de permettre la génération automatique du code des applications et d'obtenir un gain significatif en termes de temps.

- **Le prototype Co-Models**

La mise en œuvre de notre proposition s'est concrétisée par l'implémentation d'un prototype de composition de modèles prenant en charge toutes nos propositions de contributions. Co-Models permet, à l'utilisateur, de choisir les modèles sources, d'importer l'ontologie de son domaine, de le paramétrer, en choisissant le modèle principal et les options d'équivalence, et enfin de générer le modèle cible.

- **Evaluation de notre approche : expérimentations**

Notre travail s'est achevé par des expérimentations d'évaluation de notre approche. Cette évaluation avait pour objectifs (1) de comparer le modèle résultat produit manuellement à celui produit automatiquement, et (2) d'évaluer l'outil Co-Models.

Bien que Co-Models ne prenne pas en compte l'hétérogénéité de nature et de l'inclusion partielle, il permet un gain considérable de temps, et permet d'assurer une fusion des éléments présentés différemment dans les modèles cibles, meilleure que celle proposée par les participants. En effet, les participants produisent un modèle résultat contenant des incohérences de navigabilité, de composition (agrégation), et de multiplicité.

Selon les participants, Co-Models est intéressant à utiliser pour une composition de modèles, du fait qu'il a été jugé utile, facile (ou moyennement facile) à utiliser, et permet un gain significatif en terme de temps. Tous ces critères font de Co-Models un outil intéressant sur lequel les utilisateurs peuvent s'appuyer pour la composition de modèles.

6.3. Limites de nos travaux

Nos travaux ne prétendent pas apporter une solution parfaite et indiscutable à une problématique complexe qui est celle de la composition de modèles. Nous avons essentiellement essayé, dans ces travaux, de mettre en valeur l'utilisation des ontologies ainsi que d'autres ressources, afin de proposer une approche hybride, dans l'objectif d'augmenter la chance d'avoir un modèle résultat sans conflit. De ce fait, les limites de nos travaux sont :

- **Dépendance vis-à-vis d'une ontologie de domaine**

L'utilisation de notre outil de composition de modèles requiert l'exploitation d'une ontologie de domaine. Nous avons ainsi fait hypothèse que nous disposions de cette ontologie.

Dans la pratique, l'ingénierie des connaissances ne couvre pas encore tous les domaines. Pour les besoins de nos études de cas, nous avons eu besoin par conséquent, de construire des extraits d'ontologie dans le domaine bancaire, et celui de la finance (en particulier pour ce qui concerne la recette publique).

Cependant, d'un côté, il existe aujourd'hui un nombre d'ontologies de plus en plus grandissant, tels que des ontologies du e-gov[33], des ontologies biomédicales[34] (au nombre de 328), une centaine d'ontologies OWL[35] dans les domaines du tourisme, voyage, biochimie, famille, géographie, etc., des

[33] http://oegov.org/
[34] http://bioportal.bioontology.org/
[35] http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library#OWL_ontologies

librairies d'ontologies[36], ainsi que des moteurs de recherche, tel que Swoogle[37], possédant jusqu'à 10.000 ontologies.

D'un autre côté, nous sommes confiants que dans l'avenir, les ontologies couvriront tous les domaines. En effet, les ontologies sont depuis quelques années un sujet de recherche en pleine expansion, du fait qu'elles sont utilisées dans différents domaines d'application à savoir l'intelligence artificielle, le web sémantique, le génie logiciel, l'informatique biomédicale ou encore l'architecture de l'information.

- **Hétérogénéité de nature**

Lors de la comparaison, nous nous sommes restreints à la comparaison de deux éléments de même nature (deux classes, deux attributs, etc.). Cependant, il se peut que dans un premier modèle, un concept soit modélisé par un certain type d'élément et dans un deuxième modèle par un autre. L'exemple classique rencontré en modélisation est celui d'un concept modélisé en tant que classe dans un modèle, et modélisé en tant que simple attribut dans un autre modèle. Ainsi, le concept « Auteur » peut être représenté sous forme de classe dans un modèle et sous forme d'attribut dans un autre, tout simplement parce que le concepteur n'accorde pas le même niveau d'importance (au vu de ses besoins). Dans ce cas, dans notre système, il y a un risque de se retrouver avec les deux informations sous des formes différentes (attribut et classe), provoquant alors une duplication dans le modèle résultat.

- **Hétérogénéité d'inclusion partielle**

Nous avons fait le choix de ne pas traiter l'inclusion partielle de deux éléments. En effet, nous considérons que juger équivalents deux éléments avec seulement des syllabes en commun, peut générer de fausses équivalences. Exemple d'un premier modèle, contenant l'élément « matériel », et un second modèle avec l'élément « matière », pour lesquels aucune équivalence syntaxique ne peut être attribuée.

Afin de pallier les hétérogénéités de nature et d'inclusion partielle, nous donnons la possibilité à l'utilisateur, dans ces cas, de modifier le modèle résultat.

Prenons l'exemple de l'étude de cas du domaine bancaire, le modèle M2 comprend « Versement » et « Retrait » qui sont représentés sous forme de **classes** reliées respectivement par les associations « EstAlimentéPar » et « EstDébitéPar » à la classe « CompteBancaire »; et représentés dans le modèle M1, sous forme de l'**attribut** « type » de la classe « Opération » reliée à la classe « Compte » par l'association « Comprend ».

L'utilisateur, dans ce cas, doit intervenir pour modifier le modèle cible, afin d'éliminer le conflit de redondance dû à l'hétérogénéité de nature. De ce fait, il doit supprimer l'attribut « type » et les deux associations « EstAlimentéPar » et « EstDébitéPar ». Le modèle résultat modifié par l'utilisateur est présenté en Annexe 5.

[36] http://www.loa.istc.cnr.it/medicine/#theory-list
[37] www.waterstones.com

6.4. Perspectives

- **Nombre de modèles sources**

Le processus de composition, décrit dans nos travaux, reçoit en entrée uniquement deux modèles. Pour l'étendre à un nombre quelconque de modèles, nous adoptons la stratégie à échelle, étudiée entre autres dans [Batini et al., 1986]. Elle consiste à composer d'abord deux modèles. Le résultat de cette première composition est à son tour composé avec un troisième modèle et ainsi de suite, jusqu'à ce que tous les modèles soient composés.

- **Hétérogénéité de nature**

Nous pensons étendre notre phase de composition afin qu'elle prenne en compte l'hétérogénéité de nature, en comparant syntaxiquement et sémantiquement tous les éléments de nature différentes.

- **Hétérogénéité d'inclusion partielle**

Afin de prendre en compte l'inclusion partielle, nous pensons définir une règle de comparaison qui permet de vérifier l'équivalence sémantique même si les éléments sont en inclusion partielle. Dans ce cas, pour l'exemple cité plus haut, les deux éléments « matériel » et « matière » seront jugés différents car ils ne sont pas sémantiquement équivalents.

- **Vérification du modèle résultat**

Une autre perspective concerne la vérification automatique des modèles cibles après la composition de modèles. Nous pensons à une étape de post-composition qui permettra une vérification et validation du modèle résultat.

- **Composition de modèles UML plus complets**

Nous pensons à la composition de modèles plus complets, c'est-à-dire supportant par exemple des classes-associations, ou des contraintes OCL. Cette ambition se base sur notre conviction que l'ontologie de domaine possède une richesse sémantique qui peut assurer la composition de contraintes OCL, par exemple l'intersection, l'union, etc.

Références Bibliographiques

A

- [Alanen et al., 2003] M. Alanen et I. Porres, « Difference and Union of Models », UML 2003, LNCS 28663, Perdita Stevens, Jon Whittle, Grady Booch (Eds.), pp. 2-17, 2003.
- [Algergawy et al., 2009] A. Algergawy, E. Schallehn, et G. Saake, « Improving XML schema matching performance using Prüfer sequences », *Data & Knowledge Engineering*, Vol. 68 No. 8, pp. 728-747, Août 2009.
- [Anwar et al., 2006] A. Anwar, M. Nassar, B. Coulette, S. Ebersold, et A. Kriouile, « Vers la fusion de modèles par points de vues avec le profil VUML », *Actes du Workshop OCM-SI'2006*, Hammamet, Tunisie, 31 Mai 2006.
- [Anwar et al., 2007a] A. Anwar, S. Ebersold, B. Coulette, M. Nassar, et A. Kriouile, « Une approche MDA pour produire un modèle VUML par intégration de modèles par points de vue », *Actes des 3ème journées sur l'ingénierie dirigée par les modèles (IDM'2007)*, Hermès Science, pp. 41-58, Toulouse, France, 2007.
- [Anwar et al., 2007b] A. Anwar, S. Ebersold, B. Coulette, M. Nassar, et A. Kriouile, « Vers une approche à base de règles pour la composition de modèles. Application au profil VUML », *L'Objet, Hermès Science Publications, Numéro spécial Ingénierie Dirigée par les Modèles*, Vol. 13, No. 4/2007, pp. 73-103, Décembre 2007.
- [Anwar et al., 2008a] A. Anwar, M. Nassar, S. Ebersold, B. Coulette, et A. Kriouile, « A QVTBased Approach For Model Composition: Application to the VUML Profile », *International Conference on Entreprise Information Systems (ICEIS 2008)*, Barcelona, INSTICC Press, pp. 360-367, Juin 2008.
- [Anwar et al., 2008b] A. Anwar, S. Ebersold, M. Nassar, B. Coulette, et A. Kriouile, «Towards a generic approach for model composition », *International Conference on Software Engineering Advances (ICSEA), IEEE Computer Society*, pp. 84-90, Malte, Octobre 27-30, 2008.
- [Anwar et al., 2010] A. Anwar, S. Ebersold, M. Nassar, B. Coulette, et A. Kriouile, «A Rule-Driven Approach for composition of Viewpoint-oriented Models », *Journal of Object Technology (JOT)*, Mars 2010.
- [ATL, 2006] ATLAS group LINA & INRIA, ATL: Atlas Transformation Language - User Manual, 2006

B

- [Bachimont, 2000] B. Bachimont, « Engagement sémantique et engagement ontologique : conception et réalisation d'ontologies en ingénierie des connaissances », *Ingénierie des connaissances, évolutions récentes et nouveaux défis*, Vol. 1, pp. 1-16, 2000.
- [Bartelt, 2008] C. Bartelt, « Consistence Preserving Model Merge in Collaborative Development Processes », *International Workshop on Comparison and versioning of software models*, pp. 13-18, 2008

- [Batini et al., 1986] C. Batini, M. Lenzerini, et S.B. Navathe, « A comparative analysis of methodologies for database schema integration », *Journal ACM Computing Surveys (CSUR)*, Vol. 18, No. 4, pp. 323–364, Décembre 1986.
- [Bechhofer et al., 2001] S. Bechhoferb, I. Horrocks, C. Goble et R. Stevens, « OilEd a reason-able ontology editor for the semantic web », *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, number 2174 in Lecture Notes in Computer Science*, pp. 396–408, Vienna : Springer-Verlag 2001, Disponible sur : http://potato.cs.man.ac.uk/papers/ki2001.pdf.
- [Bellahsene et al., 2011] Z. Bellahsene, A. Bonifati, et E. Rahm, « Schema Matching and Mapping », Bellahsene, Zohra, Bonifati, Angela, Rahm, Erhard (Eds.), ISBN 978-3-642-16517-7, Springer, 2011.
- [Benabdellah et al., 2009] S. C. Benabdellah, M. Fredj, S. Mouline, « Vers une fusion des modèles métier en utilisant les ontologies », *1ères Journées Doctorales en Technologies de l'Information et de la Communication (JDTIC'09)*, ENSIAS, Rabat, 16-18 Juillet 2009.
- [Benabdellah et al., 2010a] S. C. Benabdellah, M. Fredj, S.Mouline, « Un système à base de règles d'aide à la décision pour la comparaison sémantique des modèles », *Conférence internationale SITA'10 (Systèmes Intelligents : Théorie et Applications)*, 4-5 Mai 2010, ENSIAS, Rabat, Maroc.
- [Benabdellah et al., 2010b] S. C. Benabdellah, M. Fredj, « Vers une contribution à l'intégration sémantique des modèles UML », *ERATSI (Évolution, la Réutilisation, l'Adaptation et la Traçabilité des Systèmes d'Information) organisé conjointement avec INFORSID'10*, Marseille, France, 25 Mai 2010.
- [Benabdellah et al., 2010c] S. C. Benabdellah, M. Fredj et S. Mouline, « Intégration des modèles métier avec les ontologies », *2èmes Journées Doctorales en Technologies de l'Information et de la Communication (JDTIC'10)*, ENSIAS, Rabat, Juillet 2010.
- [Benabdellah et al., 2010d] S. C. Benabdellah, M. Fredj et S. Mouline, « An Hybryd approach for Models Comparison », *IADIS (International Association for the Development of Information Society) Applied Computing 2010 (AC 2010)*, Timisoara, Roumanie, 14-16 Octobre 2010.
- [Benabdellah et al., 2010e] S. C. Benabdellah, M. Fredj et S. Mouline, « Towards a Complete Models Comparison for information Systems Integration Context », *International Conference on Models of Information and Communications Systems (MICS'10)*, 2-4 Novembre 2010, ENSIAS, Rabat, Maroc.
- [Benabdellah et al., 2010f] S. C. Benabdellah, M. Fredj et S. Mouline, « An Hybryd approach for Models Comparison », *IADIS international Journal on Computer Science and Information Systems*, Volume 6, No. 1, ISSN: 1646-3692, 2010
- [Benabdellah et al., 2011a] S. C. Benabdellah, M. Fredj et S. Mouline, « MDA based-approach for UML Models Complete Comparison », *International Journal of Computer Science Issues*, Volume 8, No. 2, ISSN: 1694-0814, Mars 2011
- [Benabdellah et al., 2011b] S. C. Benabdellah, M. Fredj et S. Mouline, « Metamodels for models complete Integration », *IEEE International Conference on Information Reuse and Integration (IRI 2011)*, 3-5 August 2011, Las Vegas, Nevada, USA

- [Bernstein, 2003] P. A. Bernstein, « Applying Model Management to Classical Meta Data Problems », *1st Biennial Conference on Innovative Data Systems Research (CIDR)*, pp. 209-220, 2003.
- [Bernstein et al., 2000] P. A. Bernstein, A. Y. Halevy, et R. A. Pottinger. « A vision of management of complex models », *ACM Special Interest Group on Management of Data,* Vol. 29 No. 4, pp. 55-63, 2000.
- [Bernstein et al., 2011] P. A. Bernstein, J. Madhavan et E. Rahm, « Generic Schema Matching, Ten Years Later », *Very Large Database VLDB*, Vol. 4 No. 1, pp. 695-701, 2011.
- [Bézivin et al., 2001] J. Bézivin et O. Gerbé, « Towards a Precise Definition of the OMG/MDA Framework », *16th IEEE international conference on Automated software engineering ASE'01*, Novembre 2001.
- [Bézivin et al., 2002] J. Bézivin et X.Blanc, « MDA : Vers un important changement de paradigme en génie logiciel», *Journal Développeur Référence*, pp. 7-11, Juillet 2002.
- [Bézivin, 2004] J. Bézivin, « Sur les principes de base de l'ingénierie des modèles », *L'objet*, ISSN 1262-1137, Vol. 10, No 4, pp. 145-156, 2004
- [Bézivin et al., 2006] J. Bézivin, S. Bouzitouna, M. Didonet Del Fabro, M. P. Gervais, F. Jouault, D. Kolovos, I. Kurtev, et R.C. Paige, « A canonical scheme for model composition » , *Proceedings European Conference in Model Driven Architecture (EC-MDA) 2006*, Bilbao, Espagne, Juillet 2006.
- [Booch, 1994] G. Booch. « Object-Oriented Analysis and Design with Applications», 2nd Edition, The Benjamin/Cummings Series in Object-Oriented Software Engineering, 1994
- [Boronat et al., 2005] A. Boronat, J. Carsí, et I. Ramos, « MOMENT: a formal MOdel manageMENT tool », *Summer School on Generative and Transformational Techniques in Software Engineering*, 2005.
- [Boronat et al., 2007] A. Boronat, J. Carsí, I. Ramos, et P. Letelier , « Formal Model Merging Applied to Class Diagram Integration », *Electronic Notes in Theoretical Computer Science (ENTCS)*, Vol. 116, pp. 5-26, Janvier 2007.
- [Borst, 1997] W. N. Borst, « Construction of Engineering Ontologies for Knowledge Sharing and Reuse », PhD thesis, Tweente University, Enschede, NL- Centre for Telematica and Information Technology, 1997.
- [Bouzitouna et al., 2004] S. Bouzitouna et M.P. Gervais : « Composition Rules for PIM Reuse», *2nd European Workshop on MDA with Emphasis on Methodologies and Transformations*, David H. Akehurst (Ed.), Canterbury, UK, pp. 36-43, 2004
- [Breitman et al., 2005] K. Breitman, C. Felicissimo, et M. Casanova, « CATO - A Lightweight Ontology Alignment Tool », *CAiSE Short Paper Proceedings*, 2005.
- [Brunet et al., 2006] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, et M. Sabetzadeh, « A Manifesto for Model Merging », *Proceedings of the 2006 international workshop on Global integrated model management*, pp. 5 – 12, Shanghai, Chine, 2006.

C

- [Charlet, 2002] J. Charlet, « L'Ingénierie des connaissances : développements, résultats et perspectives pour la gestion des connaissances médicales », Rapport interne, Habilitation à diriger des recherches, Université Paris 6, Disponible sur : http://archivesic.ccsd.cnrs.fr/sic_00001064, Soutenu le 10 Décembre 2002.
- [Coleman et al. 1994] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, et P. Jeremes. «Object-Oriented Development. The Fusion Method », Prentice Hall, 1994
- [Combemale, 2008] B. Combemale, « Ingénierie Dirigée par les Modèles (IDM) - État de l'art », Rapport de recherche, Institut de Recherche en Informatique de Toulouse (UMR CNRS 5505), 2008, Disponible sur : http://hal.inria.fr/docs/00/37/15/65/PDF/mde-stateoftheart.pdf
- [Cook et al., 1994] S. Cook et J. Daniels. « Designing Object Systems. Object-Oriented Modelling with Syntropy », Prentice-Hall, 1994
- [Corcho et al., 2000] O. Corcho et A. Gomez-Pérez, « Evaluating knowledge representation and reasoning capabilities of ontology specification languages», *Workshop on Applications of Ontologies and Problem Solving Methods (ECAI)*, Berlin, Allemagne, 2000, Disponible sur: http://www.cs.man.ac.uk/~ocorcho/documents/ECAI00WS_CorchoGomezPerez.pdf

D

- [Dameron, 2003] O. Dameron, « Modélisation, représentation et partage de connaissances anatomiques sur le cortex cérébral », Thèse de doctorat, Faculté de médecine de Rennes, Disponible sur : http://sim3.univ-rennes1.fr/~odameron/thesis/dameronThesis.pdf
- [Dean et al., 2004] M. Dean, G. Schreiber, S. Bechhofer, F. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider et L. A. Stein , « OWL Web Ontology Language – Reference », 2004.
- [Debbagh et al., 2007] T. Debbagh, I. Alaoui Ismaili, B. Bounabat, « Stratégie e-Maroc 2010-Réalisations, Orientations & Plans d'action », Septembre 2007.
- [Didonet et al., 2005a] M. Didonet Del Fabro et F. Jouault, « Model Transformation and Weaving in the AMMA Platform », *Generative and Transformational Techniques in Software Engineering (GTTSE)*, pp. 71-77, Braga, Portugal, 2005.
- [Didonet et al., 2005b] M. Didonet Del Fabro, J. Bézivin, F. Jouault, E. Breton, et G. Gueltas, « AMW : A Generic Model Weaver », *Proceedings 1ères Journées sur l'Ingénierie Dirigée par les Modèles, IDM'05*, 2005.
- [Didonet et al., 2006] M. Didonet Del Fabro, J. Bézivin, et P. Valduriez, « Weaving Models with the Eclipse AMW plugin », *Proceedings Eclipse Modeling Symposium, Eclipse Summit Europe 2006*, Esslingen, Allemagne, 2006.
- [Dimitrios et al., 2006a] S. K. Dimitrios, R. F. Paige, et F. A. C. Polack, « Merging Models with the Epsilon Merging Language (EML) », *Proceedings ACM/IEEE 9th International Conference on*

Model Driven Engineering Languages and Systems (Models/ UML 2006), Vol. LNCS, Genova, Italie, Octobre 2006.

- [Dimitrios et al., 2006b] S. K. Dimitrios, R. F. Paige, et F. A. C. Polack, « Model Comparison : A Foundation for Model Composition and Model Transformation Testing », *Proceedings GaMMa 2006, 1st International Workshop on Global Integrated Model Management*, pp. 13–20, New York, 2006.
- [Dimitrios et al., 2006c] S. K. Dimitrios, R. F. Paige, et F. A. C. Polack., « Eclipse Development Tools for Epsilon », *Eclipse Summit Europe, Eclipse Modeling Symposium*, Esslingen, Allemagne, Octobre 2006.

E

- [Ehrig et al., 2005] M. Ehrig, S. Staab, et Y. Sure, « Bootstrapping Ontology Alignment Methods with APFEL », *Proceedings WWW '05 Special interest tracks and posters of the 14th international conference on World Wide Web*, pp. 1148–1149, New York, 2005.
- [Euzenat, 2004] J. Euzenat, « An API for Ontology Alignment », *International Semantic Web Conference (ISWC 2004)*, pp. 698-712, Hiroshima, Japan, 2004.
- [Euzenat et al., 2010] J. Euzenat, A. Ferrara, C. Meilicke, A. Nikolov, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Šváb-Zamazal, V. Svátek et C. Trojahn, « Final Results of the Ontology Alignment Evaluation Initiative 2010 », *Proceedings 5th ISWC workshop on Ontology Matching*, Shanghai, Chine, 2010.

F

- [Fagin et al., 2011] R. Fagin, P.G. Kolaitis, L. Popa, et W.-C. Tan, « Schema Mapping Evolution Through Composition and Inversion », *Proceedings Schema Matching and Mapping*, Springer, pp. 191-222, 2011.
- [Felicissimo, 2004] C. Felicissimo, « Semantic Interioperability on Web: a Strategy to Taxonomical Alignment of Ontology» Master's thesis, Département d'Informatique, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brézil, Août, 2004.
- [Fleurey et al., 2007a] F. Fleurey, B. Baudry, R. France et S. Ghosh, « A Generic Approach For Model Composition », *Proceedings of the Aspect Oriented Modeling, Workshop at Models 2007*, Nashville USA, 2007.
- [Fleurey et al., 2007b] F. Fleurey, R. Reddy, R France, B. Baudry et M. Clavreul, « Kompose : a generic model composition tool », 2007, Disponible sur http://www.kermeta.org/kompose/
- [France et al., 2004] R. France, I. Ray, G. Georg, et S. Ghosh, « An aspect-oriented approach to design modeling », *IEEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design,* Vol. 151, pp. 173-185, 2004

- [France et al., 2007] R. France, F. Fleurey, R. Reddy, B. Baudry et S. Ghosh, « Providing Support for Model Composition in Metamodels », *Proceeding 11th IEEE Enterprise Distributed Object Computing Conference (EDOC)*, pp. 253-264, Annapolis, Maryland, USA, Octobre 2007.
- [Fürst, 2004] F. Fürst, « Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationalisation », Thèse de doctorat en informatique de l'École Polytechnique de l'Université de Nantes (EPUN), Nantes, France, Novembre 2004.

G

- [Gagnon, 2011] C. Gagnon, « Fusion des banques - le cas de la Grèce », Présentation, Université de Montréal, 2011. Disponible sur http://fr.slideshare.net/chantgag/contextualisation-fusion-des-banques-10433349
- [Gómez-Pérez et al., 2004] A. Gómez-Pérez, M.L. Fernández, et O. Chorcho, « Ontological Engineering. Advanced Information and Knowledge Processing », ISBN 978-1-85233-551-9, Springer, 2004.
- [Gruber, 1993] T. Gruber, « A Translation Approach to Portable Ontology Specifications», *Knwledge Acquisition*, Vol. 5, No. 2, pp. 199-220, 1993.
- [Gruber et al., 1994] T. Gruber et G. Olsen, « An ontology for engineering mathematics », *Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Doyle, P. Torasso, and E. Sandewall (Eds.), Gustav Stresemann Institut, Bonn, Allemagne, Morgan Kaufmann, 1994.
- [Guarino et al., 1994] N. Guarino, M. Carrara, et P. Giaretta, « Formalizing ontological commitments », *National Conference of the American Association on Artificial Intelligence (AAAI)*, pp. 560–567, 1994.
- [Guarino et al., 1995] N. Guarino, A. Gangemi, C. Masolo et A. Oltrari. « Understanding top-level ontological distinctions », *Workshop on Basic Ontological Issues in Knowledge Sharing, The International Joint Conference on Artificial Intelligence (IJCAI)*, Montréal, Québec, Canada, 1995.
- [Guarino, 1996] N. Guarino, « Understanding, building and using ontologies », *Workshop on Knowledge Acquisition for Knowledge-Based Systems (KAW)*, Banff, Canada, Novembre 1996, Disponible sur : http://ksi.cpsc.ucalgary.ca/KAW/KAW96/guarino/guarino.html
- [Guarino, 1997] N. Guarino, « Some organizing principles for a unified top-level ontology », *National Conference of the American Association on Artificial Intelligence (AAAI)*, pp. 57–63, Stanford, Etats Unis, 1997.
- [Guarino, 1998] N. Guarino, « Formal ontology in information systems », *1st International Conference on Formal Ontology in Information Systems (FOIS)*, pp. 3–15, Trento, Italie, 1998.
- [Guarino, 2000] N. Guarino et C. Welty, «A Formal Ontology of Properties », *12th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, Lecture Notes in Computer Science, Juan-les-Pins, France : Springer Verlag ,Vol. 1937 pp. 97–112, 2000.

H

- [Hacene, 2003] Hacene, « Une approche d'intégration de modèles UML basée sur les treillis de Galois », Rapport d'activité, Université de Montréal, Département d'Informatique et de Recherche Opérationnelle, Laboratoire du Génie Logiciel, 2003.
- [Haddar et al., 2002] N. Haddar, F. Gargouri et A.Ben Hamadou, « Une approche formelle pour l'intégration des aspects structuraux et comportementaux de représentations conceptuelles », *ISDM Journal*, No. 19, 2002
- [Heijst et al., 1997] G. Van Heijst, A. Schreiber, et B. Weielinga, « Using explicit ontologies in kbs development », *International Journal of Human and Computer Studies - Knowledge Acquisition*, Vol. 46, No. 2-3, pp. 183–292, Mars 1997.
- [Herrmann et al., 2007] C. Herrmann, H. Krahn, B. Rumpe, M. Schindler et S. Volkel, « An Algebric View on the Semantics of Model Composition », *D. H. Akehurst, R. Vogel, et R. F. Paige (eds.), Lecture Notes in Computer Science*, Springer, Vol 4530, pp 99-113, 2007.
- [Hu, 2008] G. Hu, « A Formal Specification of UML Class and State Diagrams», *9th ACIS International Conference on Software Engineering, Artificial Intelligence, Network and Parallel and Distributed Computer (SPDN),* Phuket, Thailand, 2008.

I

- [Izza, 2006] S. Izza, « Intégration des systèmes d'information industriels : Une approche flexible basée sur les services sémantiques », Thèse soutenue à l'Ecole Nationale Supérieure des Mines à Saint-Etienne, le 20 novembre 2006.

J

- [Jai, 2006] B. Jai, Attijariwafa bank, « Opérations de fusions des banques dans le développement économique et bancaire: un mal nécessaire? », Présentation, Casablanca, le 28 avril 2006.
- [Jakimi et al., 2007] A. Jakimi, M. El Koutbi, A. Sarbraoui, et A. Idri, « Fusion de scénarios UML et Génération de code », *4th International Conference: Sciences of Electronic, Technologies of Information and Telecommunications (SETIT)*, Tunisie, 2007.
- [Janssen et al., 2011] M. Janssen, Y. Charalabibis, G. Kuk, et T. Cresswell, « E-government Interoperability, Infrastructure and Architecture: State-of-the-art and Challenges », Journal of Theoretical and Applied Electronic Commerce Research, ISSN 0718-1876, Vol. 6, No.1, pp. I-VIII., 2011
- [Jing et al., 2005] Y. Jing, L. Quan, L. Xiaoshan, et Z. Liu « A Predicative Semantic Model for Integrating UML Models», *Lecture notes in computer science, ISSN 0302-9743* Vol. 3407, pp. 170-186, 2005.

- [Jossic et al., 2007] A. Jossic, M. Didonet Del Fabro, J-P. Lerat, J. Bézivin et F. Jouault, « Model Integration with Model Weaving-a Case Study in System Architecture », *International Conference on Systems Engineering and Modeling (ICSEM)*, Herzeliya and Haifa, Israel, 2007.
- [Jouault, 2007] F. Jouault, « Contribution à l'étude des langages de transformation de modèles », Thèse de Doctorat Ecole doctorale STIM (Sciences et Technologies de l'Information et des matériaux), Soutenu le 26 septembre 2006

K

- [Kassel, 2002] G. Kassel, « Ontospec : une méthode de spécification semi-formelle d'ontologies », *13e journées francophones d'Ingénierie des Connaissances (IC)*, pp. 75–87, Rouen, France, 2002.
- [Kiczales et al., 1997] G. Kiczales, J. Lampng, A. Mendhekar, C. Maeda, et L.C. Videira, «Aspect-Oriented Programming », *11th European Conference Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241*, Finland, 1997.
- [Kleppe et al., 2003] A. Kleppe, S. Warmer, et W. Bast, « MDA Explained. The Model Driven Architecture: Practice and Promise », Addison-Wesley, Avril 2003.
- [Knublauch et al., 2004] H. Knublauch, R. Fergerson, N. Noy, et M. Musen, « The protégé owl plugin : An open development environment for semantic web applications », *The Third International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, Vol. 3298, pp. 229, Hiroshima, Japan : Springer, 2004, Disponible sur: http: //smi-web.stanford.edu/auslese/smi-web/reports/SMI-2004-1011.pdf.

M

- [Madhavan et al., 2001] J. Madhavan, P. Bernstein, et E. Rahm, « Generic schema matching using Cupid », *27th International Conference on Very Large Data Bases (VLDB),* pp. 48–58, Rome Italie, 2001, Disponible sur : http://research.microsoft.com/ philbe/CupidVLDB01.pdf.
- [Maedche et al., 2001] A. Maedche et S. Staab, « Ontology learning for the semantic web», *Journal IEEE Intelligent System*s, Vol. 16, No. 2, pp. 72-79, 2001.
- [Manning, 1999] C. Manning et H. Schutze, « Foundations of Statistical Natural Language Processing », ISBN 978-0262133609, MIT Press, Cambridge, US, 1999.
- [MEF-Docs, 2012] Documents internes, fournis par le Ministère de l'Economie et des Finances en 2012
- [Mhiri, 2007] M. Mhiri, « Méthodologie de construction des ontologies pour la résolution de conflits de Systèmes d'Information », *1ères journées francophones sur les ontologies (JFO)*, Sousse, Tunisie, 2007.
- [Mizoguchi et al., 1995] R. Mizoguchi, J. Vanwelkenhuysen, et M. Ikeda, « Task ontology for reuse of problem solving knowledge », *Towards Very Large Knowledge Bases : Knowledge Building and Knowledge Sharing (KBKS)*, pp. 46–57, 1995.

- [MOF, 2002] OMG, « Meta Object Facility (MOF) Specification » Version 1.4, Avril 2002.

N

- [Nassar et al., 2005] M. Nassar, B. Coulette, J. Guiochet, S. Ebersold, B. El Asri, X. Crégut et A. Kriouile, « Vers un profil UML pour la conception de composants multivues », *L'Objet*, Vol. 11, No. 4, pp. 83-113, 2005.
- [Nassar, 2005] Nassar M., « Analyse/conception par points de vue : le profil VUML », Thèse de l'Institut National Polytechnique de Toulouse, Septembre 2005.
- [Navathe et al., 1986] S. Navathe, R. Elmasri, et J. Larson, « Integrating User Views in Database Design », *IEEE Journal Computers*, Vol. 19, No. 1, pp. 50-62, Janvier 1986.

O

- [ODM, 2009] Ontology Definition Metamodel, OMG Document Number: formal/2009-05-01, Mai 2009, Disponible sur http://www.omg.org/spec/ODM/1.0/PDF
- [Oliveira et al., 2007] K.S.F. Oliveira et T.C. Oliveira, « A Guidance for Model Composition », *The Second International Conference on Software Engineering Advances (ICSEA),* Cap Esterel, French Riviera, France, 2007.
- [Oliveira et al., 2008] K.S.F. Oliveira, « Model Comparison: a Strategy-Based Approach», *Software Engineering and Knowledge Engineering (SEKE)*, San Francisco, USA, 2008.
- [Oliveira et al., 2009] K.S.F. Oliveira et K. Breitman, « A Flexible Strategy-Based Model Comparison Approach: Bridging the Syntactic and Semantic Gap », *Journal of Universal Computer Science*, Vol. 15, No. 1, 2009.
- [Ouchettou et al., 2007] H. Ouchettou, M. Fredj, et O. Roudiès, « Des patterns processus pour l'e-gouvernement », *la revue électronique des technologies de l'information e-TI*, 2007.

P

- [Pim et al. 1997] B. Pim, A. Hans et T. Jan, « Engineering ontologies », *International Journal of Human-Computer Studies*, Vol. 46, pp. 365–406, 1997.
- [Pottinger et al., 2003] R. A. Pottinger et P. A. Bernstein, « Merging Models Based on Given Correspondences », University of Washington, Technical Report UW-CSE-03-02-03, 2003.

Q

- [QVT, 2011] Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, OMG Document Number: formal/2011-01-01, Version 1.1, Janvier 2011, Disponible sur : http://www.omg.org/spec/QVT/1.1

R

- [Reddy et al., 2005] R. Reddy, R.B. France, S. Ghosh, F. Fleury, et B. Baudry, « Model composition - a signature based approach », *Proceedings Aspect Oriented Modeling workshop held with MODELS/UML 2005*, Montego Bay, Jamaica, 2005.
- [Reddy et al., 2006] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, et G. Georg, « Directives for Composing Aspect-Oriented Design Class Models », *Transactions of Aspect-Oriented Software Development*, Vol.1, No. 1, LNCS 3880, pp. 75-105, Springer, 2006.
- [Roche, 2005] C. Roche, « Terminologie et ontologie », *Revue Langages,* No. 157, Mars 2005.
- [Rumbaugh et al. 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, et W. Lorensen. « Object-Oriented Modeling and Design », Prentice-Hall, 1991
- [Rungworawut, 2010] W. Rungworawut, « A Framework for UML Class Diagrams and Software Patterns Integration », *Journal of Electronics and Computer Science 2010*, Vol. 12, No.1, pp. 53-59, 2010.

S

- [Sabetzadeh et al., 2007] M. Sabetzadeh, S. Nejati, M. Chechik, et S. Easterbrook, « A Relationship-Driven Framework for Model Merging », *Proceeding The International Workshop on Modeling in Software Engineering*, pp. 2, USA, 2007.
- [Scholl et al., 2007] H. J. Scholl et R. Klischewski, « E-Government Integration and Interoperability: Framing the Research Agenda », *International Journal of Public Administration*, Vol. 30, No. 8-9, pp. 889-920, 2007.
- [Shan et al, 2008] L. Shan et H.Zhu, « A formal Descriptive Semantics of UML », *Lecture Notes in Computer Science,* Vol. 5256, pp. 375-396, 2008
- [Shlaer et al., 1988] S. Shlaer et S. Mellor. « Object-Oriented Systems Analysis. Modeling the World in Data », Yourdon Press Computing Series, 1988
- [Smith et al., 2004] M. K. Smith, C. Welty et D. L. McGuinness, « OWL Web Ontology Language – Guide », 2004, Disponible sur : http://www.w3.org/TR/2004/REC-owl-guide-20040210/ Traduction française : « Le langage d'ontologie Web OWL – Guide », Disponible sur : http://www.yoyodesign.org/doc/w3c/owl-guide-20040210/
- [Smith, 2000] G. Smith, *The Object-Z Specification Language*, Kluwer Academic Publishers, 2000.

- [Sowa, 1984] J. Sowa, « Conceptual Structures: Information Processing in Mind and Machine », *Addison-Wesley Systems Programming Series*, 1984.
- [Sowa, 2000] J. Sowa, « Ontology, metadata and semiotics », *8th International Conference on Conceptual Structures (ICCS'2000)*, Vol. 1867, pp. 55–81, Springer-Verlag LNCS, 2000.
- [Sure et al., 2002] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, et D. Wenke, « Ontoedit : Collaborative ontology development for the semantic web », *The first International Semantic Web Conference (ISWC)*, 2002, Disponible sur : http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/2002_iswc_ontoedit.pdf

T

- [Troncy et al., 2002] R. Troncy et A. Isaac, « DOE : une mise en œuvre d'une méthode de structuration différentielle pour les ontologies », *13e journées francophones d'Ingénierie des Connaissances (IC)*, pp. 63–74, Rouen, 2002.
- [Troncy et al., 2003] R. Troncy, A. Isaac et V. Malaisé, « Using xslt for interoperability : Doe and the travelling domain experiment », *The 2nd International Workshop Evaluation of Ontology-based Tools (EON)*, Vol. 87, pp. 92–102, Sanibel Island, Florida, United States, 2003, Disponible sur : http://homepages.cwi.nl/~troncy/Publications/Troncy_Isaac_Malaise-eon03.pdf

U

- [Uhrig, 2008] S. Uhrig, « Matching Class Diagrams: With Estimated Costs Towards The Exact Solution? », *Proceedings 2008 Intl. Workshop on Comparison and Versioning of software Models*, ACM, Leipzig, Allemagne, 2008.
- [UML, 2011] Specification UML, version 2.4, 2011
- [UNDESA, 2008] UN (United Nations) Department of Economic and Social Affairs, Division for Public Administration and Development Management, UN E-Government Survey 2008: From E-Government to Connected Governance, New York, 2008 disponible sur: http://unpan1.un.org/intradoc/groups/public/documents/UN/UNPAN028607.pdf
- [Uschold et al., 1995] M. Uschold et M. King, « Towards a methodology for building ontologies », *Workshop on Basic Ontological Issues in Knowledge Sharing, The International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal, Canada, 1995, Disponible sur : http://www.aiai.ed.ac.uk/project/pub/documents/1995/95-ont-ijcai95-ont-method.ps
- [Uschold et al. 1996] M. Uschold et M. Gruninger, « Ontologies: Principes, Methods and Applications », *Knowledge Engineering Review*, Vol. 11, No. 2, 1996.

V

- [Vernadat, 1996] F. B. Vernadat, « Enterprise modelling and integration: principles and applications», Chapman & Hall. ISBN: 978-0-412-60550-5, 1996.

W

- [Weerakkody et al., 2007] V. Weerakkody, M. Janssen, et K. Hjort-Madsen, « Integration and Enterprise Architecture Challenges in EGovernment : A European Perspective », *International Journal of Cases on Electronic Commerce*, Vol. 3, No. 2, pp. 14-38, 2007.
- [Wegner, 1996] P. Wegner, « Interoperability », *ACM Computing Survey*, Vol. 28 No. 1, pp. 285–287, 1996.
- [Weston, 1993] R. H. Weston, « Steps towards enterprise wide integration: a definition of needs and first generation open solutions », *International Journal of Production Research*, Vol. 31 No. 9, pp. 2235-2254, 1993.
- [Williams et al., 2001] T. J. Williams, G. A. Rathwell et H. Li, « A handbook on master planning and implementation for enterprise integration programs Based On The Purdue Enterprise Reference Architecture and the Purdue Methodology », Purdue Laboratory for Applied Industrial Control, Février 2001
- [Wimmer et al., 2001] M. Wimmer et J. Krenner, « An Integrated Online One-Stop Government Platform : The eGOV Project », *Proceeding The 9th Interdisciplinary Information Management Talks, Schriftenreihe Informatik, Universitätsverlag Trauner(Linz)*, pp. 329-337, 2001.

X

- [XMI, 2011] OMG MOF 2 XMI Mapping Specification, Version 2.4.1, OMG Document Number: formal/2011-08-09, 2011, Disponible sur : http://www.omg.org/spec/XMI/2.4.1

Annexes

Annexe 1 : Ontologie de domaine de Banque

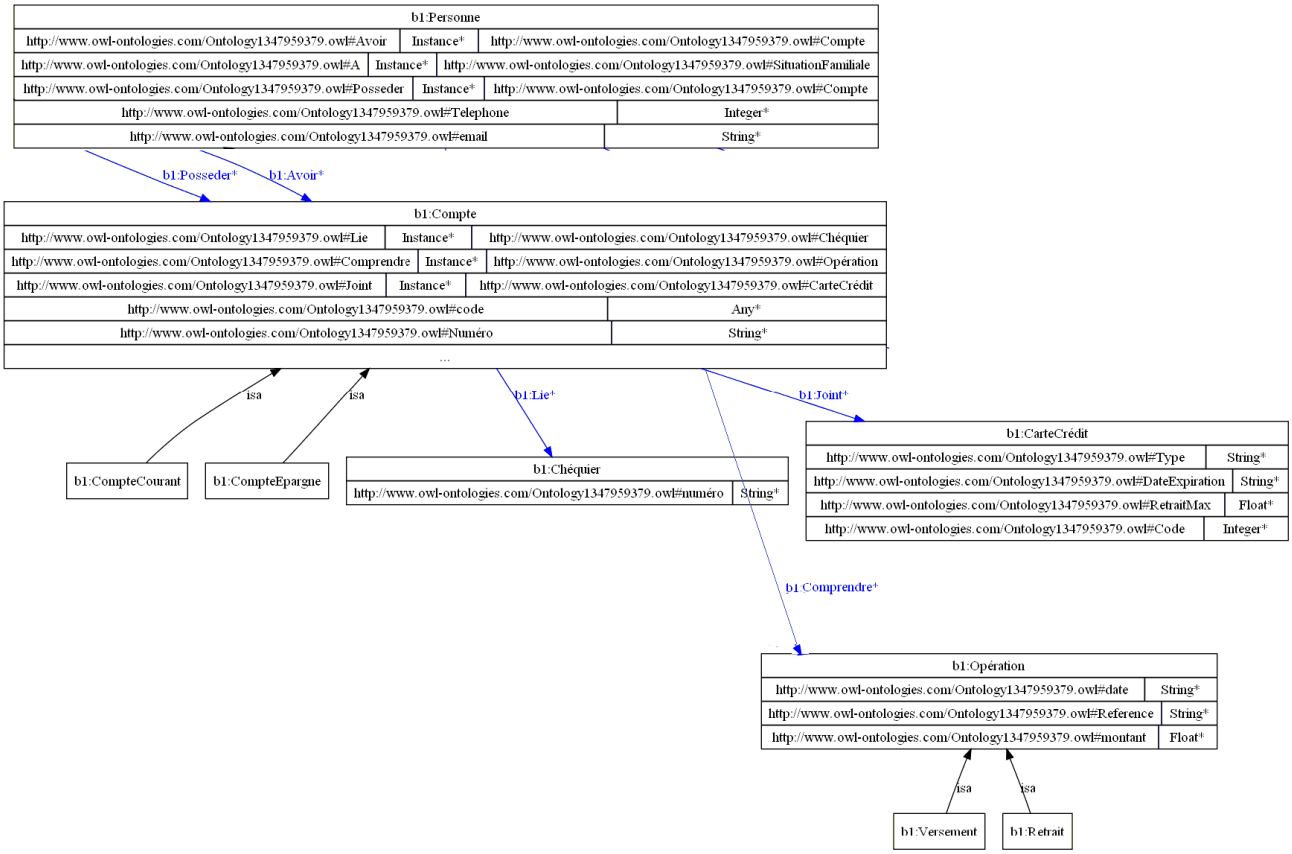
L'ontologie de domaine de Banque en langage OWL :

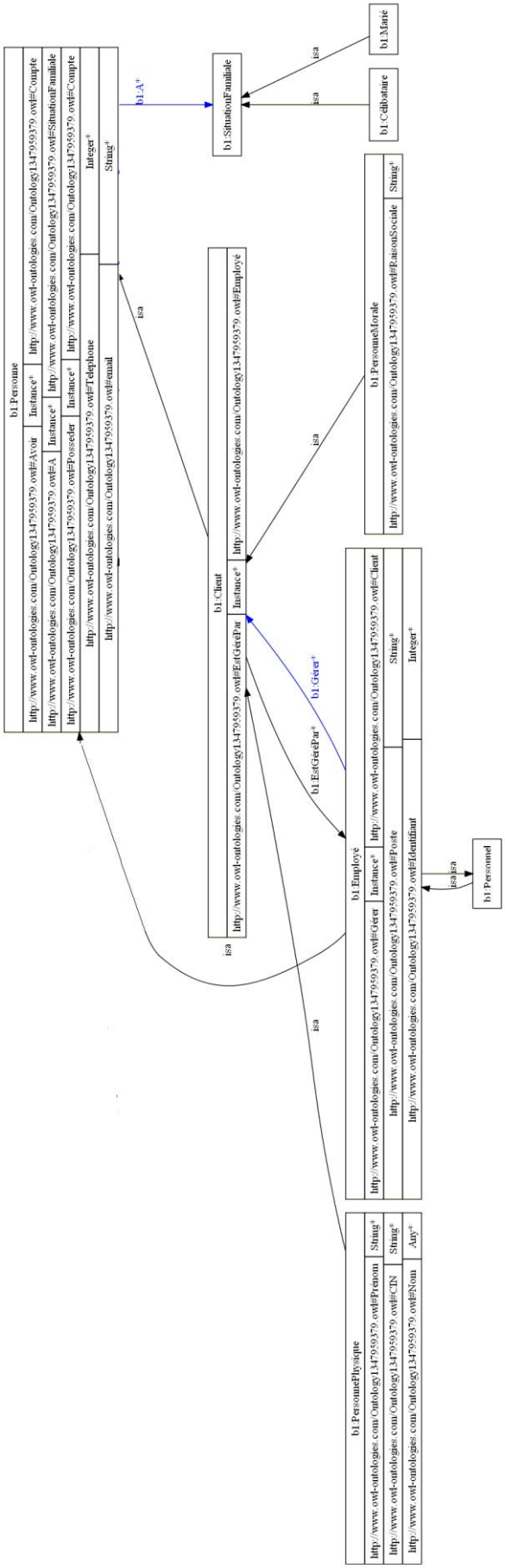
```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY protege "http://protege.stanford.edu/plugins/owl/protege#" >
    <!ENTITY xsp "http://www.owl-ontologies.com/2005/08/07/xsp.owl#" >
]>


<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1347959379.owl#"
     xml:base="http://www.owl-ontologies.com/Ontology1347959379.owl"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#">
    <owl:Ontology rdf:about=""/>
    <owl:ObjectProperty rdf:ID="A">
        <rdfs:domain rdf:resource="#Personne"/>
        <rdfs:range rdf:resource="#SituationFamiliale"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="Avoir">
        <rdfs:domain rdf:resource="#Personne"/>
        <owl:equivalentProperty rdf:resource="#Posseder"/>
        <rdfs:range rdf:resource="#Compte"/>
    </owl:ObjectProperty>
    <owl:Class rdf:ID="CarteCr&#233;dit"/>
    <owl:Class rdf:ID="Ch&#233;quier"/>
    <owl:DatatypeProperty rdf:ID="CIN">
        <rdfs:domain rdf:resource="#PersonnePhysique"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Client">
        <rdfs:subClassOf rdf:resource="#Personne"/>
    </owl:Class>
    <owl:DatatypeProperty rdf:ID="Code">
        <rdfs:domain rdf:resource="#CarteCr&#233;dit"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="code">
        <rdfs:domain rdf:resource="#Compte"/>
        <owl:equivalentProperty rdf:resource="#Num&#233;ro"/>
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:ID="Comprendre">
        <rdfs:domain rdf:resource="#Compte"/>
        <rdfs:range rdf:resource="#Op&#233;ration"/>
    </owl:ObjectProperty>
    <owl:Class rdf:ID="Compte"/>
    <owl:Class rdf:ID="CompteCourant">
        <rdfs:subClassOf rdf:resource="#Compte"/>
    </owl:Class>
    <owl:Class rdf:ID="CompteEpargne">
        <rdfs:subClassOf rdf:resource="#Compte"/>
    </owl:Class>
    <owl:Class rdf:ID="C&#233;libataire">
        <rdfs:subClassOf rdf:resource="#SituationFamiliale"/>
        <owl:disjointWith rdf:resource="#Mari&#233;"/>
    </owl:Class>
    <owl:DatatypeProperty rdf:ID="date">
        <rdfs:domain rdf:resource="#Op&#233;ration"/>
        <rdfs:range rdf:resource="&xsd;date"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="DateExpiration">
```

```
    <rdfs:domain rdf:resource="#CarteCr&#233;dit"/>
    <rdfs:range rdf:resource="&xsd;date"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="email">
    <rdfs:domain rdf:resource="#Personne"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="Employ&#233;">
    <owl:equivalentClass rdf:resource="#Personnel"/>
    <rdfs:subClassOf rdf:resource="#Personne"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="EstG&#233;r&#233;Par">
    <rdfs:domain rdf:resource="#Client"/>
    <owl:inverseOf rdf:resource="#G&#233;rer"/>
    <rdfs:range rdf:resource="#Employ&#233;"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="G&#233;rer">
    <rdfs:domain rdf:resource="#Employ&#233;"/>
    <owl:inverseOf rdf:resource="#EstG&#233;r&#233;Par"/>
    <rdfs:range rdf:resource="#Client"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="Identifiant">
    <rdfs:domain rdf:resource="#Employ&#233;"/>
    <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="Joint">
    <rdfs:domain rdf:resource="#Compte"/>
    <rdfs:range rdf:resource="#CarteCr&#233;dit"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Lie">
    <rdfs:domain rdf:resource="#Compte"/>
    <rdfs:range rdf:resource="#Ch&#233;quier"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Mari&#233;">
    <rdfs:subClassOf rdf:resource="#SituationFamiliale"/>
    <owl:disjointWith rdf:resource="#C&#233;libataire"/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="montant">
    <rdfs:domain rdf:resource="#Op&#233;ration"/>
    <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Nom">
    <rdfs:domain rdf:resource="#PersonnePhysique"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Num&#233;ro">
    <rdfs:domain rdf:resource="#Compte"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="num&#233;ro">
    <rdfs:domain rdf:resource="#Ch&#233;quier"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="Op&#233;ration"/>
<owl:Class rdf:ID="Personne"/>
<owl:Class rdf:ID="Personnel">
    <owl:equivalentClass rdf:resource="#Employ&#233;"/>
</owl:Class>
<owl:Class rdf:ID="PersonneMorale">
    <rdfs:subClassOf rdf:resource="#Client"/>
</owl:Class>
<owl:Class rdf:ID="PersonnePhysique">
    <rdfs:subClassOf rdf:resource="#Client"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="Posseder">
    <rdfs:domain rdf:resource="#Personne"/>
    <rdfs:range rdf:resource="#Compte"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="Poste">
    <rdfs:domain rdf:resource="#Employ&#233;"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Pr&#233;nom">
    <rdfs:domain rdf:resource="#PersonnePhysique"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="RaisonSociale">
    <rdfs:domain rdf:resource="#PersonneMorale"/>
```

```
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="Reference">
        <rdfs:domain rdf:resource="#Op&#233;ration"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Retrait">
        <rdfs:subClassOf rdf:resource="#Op&#233;ration"/>
    </owl:Class>
    <owl:DatatypeProperty rdf:ID="RetraitMax">
        <rdfs:domain rdf:resource="#CarteCr&#233;dit"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="SituationFamiliale"/>
    <owl:DatatypeProperty rdf:ID="Solde">
        <rdfs:domain rdf:resource="#Compte"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="Telephone">
        <rdfs:domain rdf:resource="#Personne"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="Type">
        <rdfs:domain rdf:resource="#CarteCr&#233;dit"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Versement">
        <rdfs:subClassOf rdf:resource="#Op&#233;ration"/>
    </owl:Class>
</rdf:RDF>
```

La représentation graphique de l'ontologie (divisée en deux pour une meilleure visibilité) :




b1:Personne
http://www.owl-ontologies.com/Ontology1347959379.owl#Avoir
Instance*
http://www.owl-ontologies.com/Ontology1347959379.owl#Compte
http://www.owl-ontologies.com/Ontology1347959379.owl#A
Instance*
http://www.owl-ontologies.com/Ontology1347959379.owl#SituationFamiliale
http://www.owl-ontologies.com/Ontology1347959379.owl#Posseder
Instance*
http://www.owl-ontologies.com/Ontology1347959379.owl#Compte
http://www.owl-ontologies.com/Ontology1347959379.owl#Telephone
Integer*
http://www.owl-ontologies.com/Ontology1347959379.owl#email
String*
b1:A*
b1:SituationFamiliale
b1:Marié
b1:Célibataire
isa
b1:Client
http://www.owl-ontologies.com/Ontology1347959379.owl#EstGereePar
Instance*
http://www.owl-ontologies.com/Ontology1347959379.owl#Employé
b1:Gérer*
b1:EstGereePar*
b1:PersonneMorale
http://www.owl-ontologies.com/Ontology1347959379.owl#RaisonSociale
String*
b1:Employé
http://www.owl-ontologies.com/Ontology1347959379.owl#Gérer
Instance*
http://www.owl-ontologies.com/Ontology1347959379.owl#Client
http://www.owl-ontologies.com/Ontology1347959379.owl#Poste
String*
http://www.owl-ontologies.com/Ontology1347959379.owl#Identifiant
Integer*
b1:Personnel
b1:PersonnePhysique
http://www.owl-ontologies.com/Ontology1347959379.owl#Prenom
String*
http://www.owl-ontologies.com/Ontology1347959379.owl#CIN
String*
http://www.owl-ontologies.com/Ontology1347959379.owl#Nom
Any*

Annexe 2 : Ontologie de domaine du egov

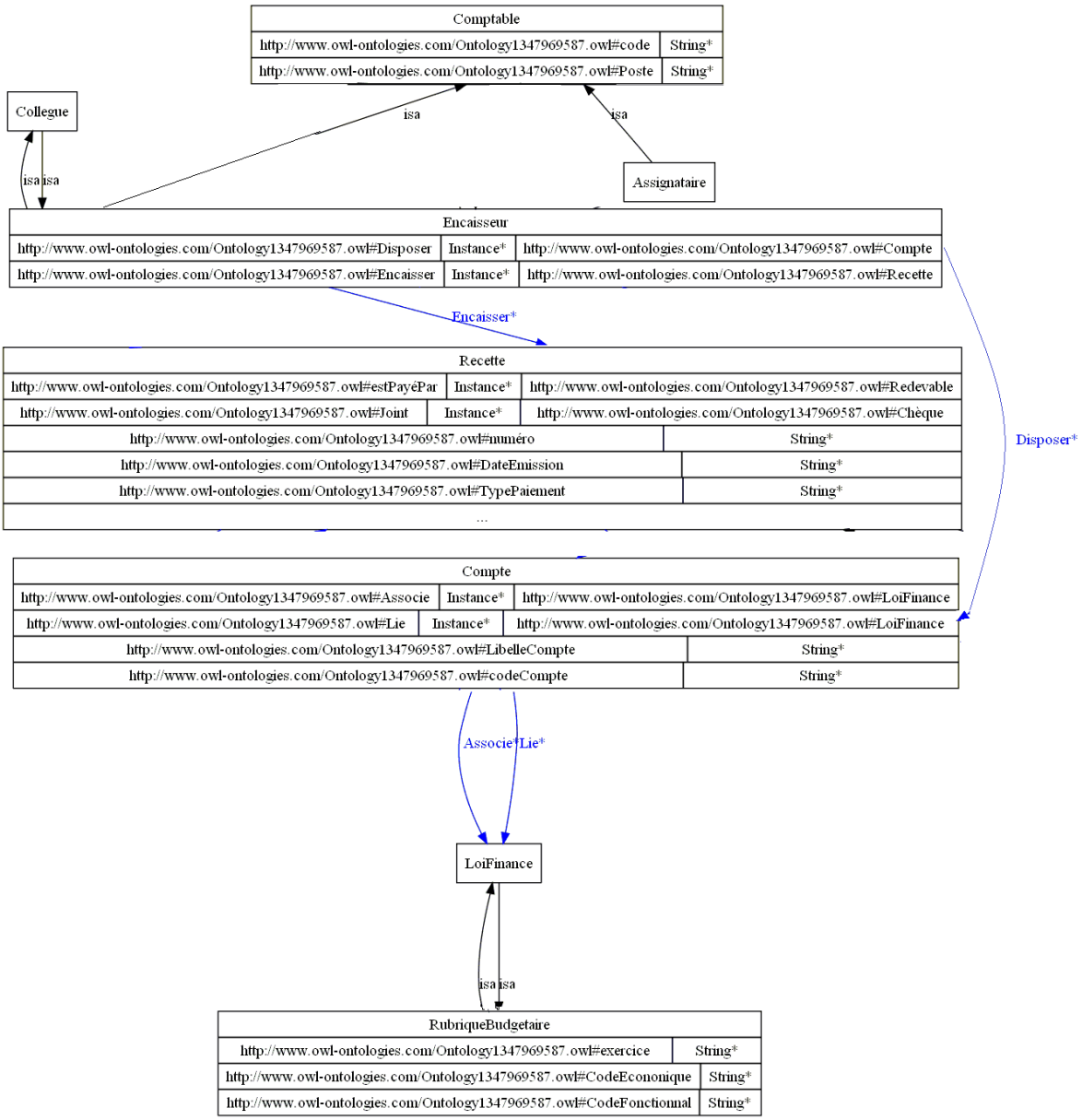
L'ontologie e-gov en langage OWL :

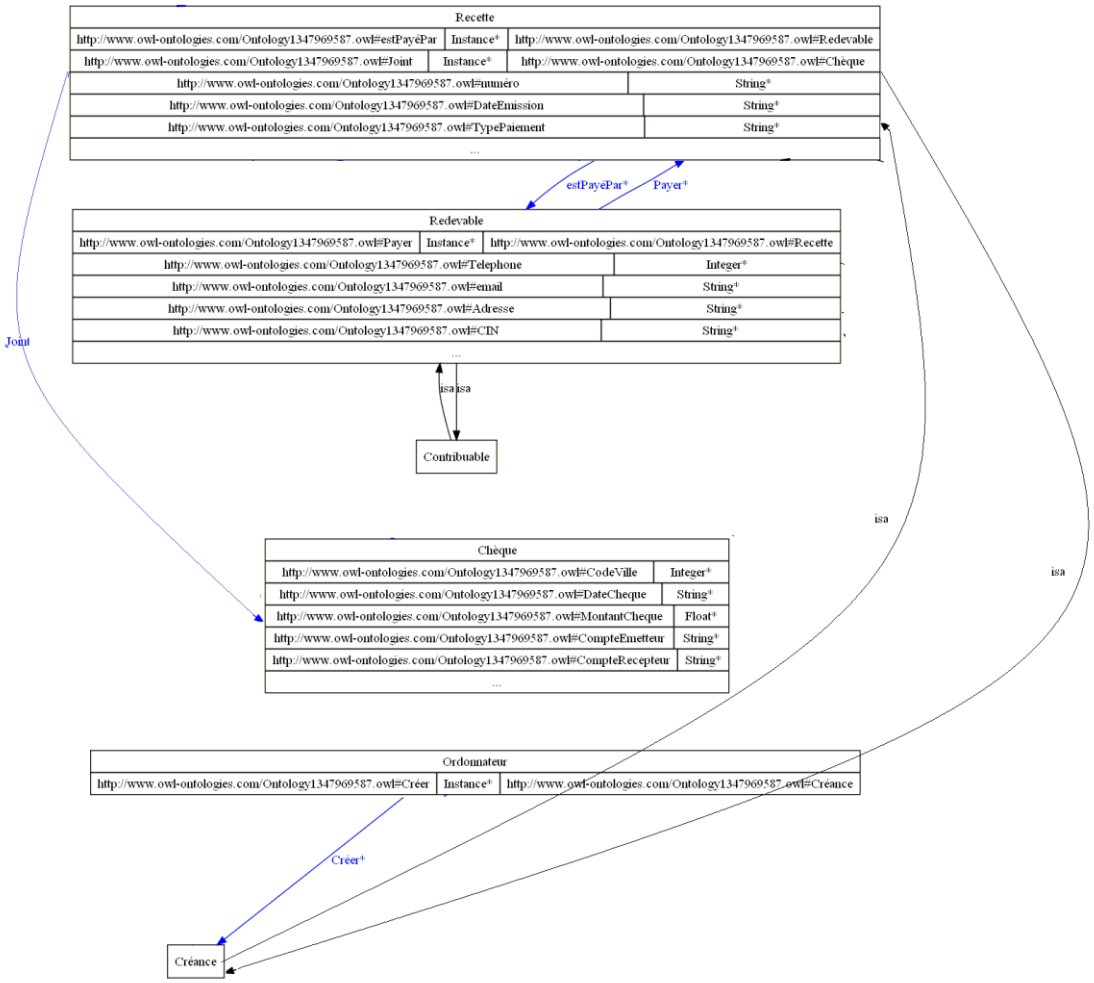
```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY protege "http://protege.stanford.edu/plugins/owl/protege#" >
    <!ENTITY xsp "http://www.owl-ontologies.com/2005/08/07/xsp.owl#" >
]>


<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1347969587.owl#"
     xml:base="http://www.owl-ontologies.com/Ontology1347969587.owl"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#">
    <owl:Ontology rdf:about=""/>
    <owl:DatatypeProperty rdf:ID="Adresse">
        <rdfs:domain rdf:resource="#Redevable"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Assignataire">
        <rdfs:subClassOf rdf:resource="#Comptable"/>
    </owl:Class>
    <owl:ObjectProperty rdf:ID="Associe">
        <rdfs:domain rdf:resource="#Compte"/>
        <owl:equivalentProperty rdf:resource="#Lie"/>
        <rdfs:range rdf:resource="#LoiFinance"/>
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:ID="Banque">
        <rdfs:domain rdf:resource="#Ch&#232;que"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Ch&#232;que"/>
    <owl:DatatypeProperty rdf:ID="CIN">
        <rdfs:domain rdf:resource="#Redevable"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="code">
        <rdfs:domain rdf:resource="#Comptable"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="codeCompte">
        <rdfs:domain rdf:resource="#Compte"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="CodeEcononique">
        <rdfs:domain rdf:resource="#RubriqueBudgetaire"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="CodeFonctionnal">
        <rdfs:domain rdf:resource="#RubriqueBudgetaire"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="CodeVille">
        <rdfs:domain rdf:resource="#Ch&#232;que"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Collegue">
        <owl:equivalentClass rdf:resource="#Encaisseur"/>
    </owl:Class>
    <owl:Class rdf:ID="Comptable"/>
    <owl:Class rdf:ID="Compte"/>
```

```
<owl:DatatypeProperty rdf:ID="CompteEmetteur">
    <rdfs:domain rdf:resource="#Ch&#232;que"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CompteRecepteur">
    <rdfs:domain rdf:resource="#Ch&#232;que"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="Contribuable">
    <owl:equivalentClass rdf:resource="#Redevable"/>
</owl:Class>
<owl:Class rdf:ID="Cr&#233;ance">
    <owl:equivalentClass rdf:resource="#Recette"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="Cr&#233;er">
    <rdfs:domain rdf:resource="#Ordonnateur"/>
    <rdfs:range rdf:resource="#Cr&#233;ance"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="DateCheque">
    <rdfs:domain rdf:resource="#Ch&#232;que"/>
    <rdfs:range rdf:resource="&xsd;date"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="DateEmission">
    <rdfs:domain rdf:resource="#Recette"/>
    <rdfs:range rdf:resource="&xsd;date"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="DateEnvoie">
    <rdfs:domain rdf:resource="#Ch&#232;que"/>
    <rdfs:range rdf:resource="&xsd;date"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="DateExigibilite">
    <rdfs:domain rdf:resource="#Recette"/>
    <rdfs:range rdf:resource="&xsd;date"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="Disposer">
    <rdfs:domain rdf:resource="#Encaisseur"/>
    <rdfs:range rdf:resource="#Compte"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="email">
    <rdfs:domain rdf:resource="#Redevable"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="Encaisser">
    <rdfs:domain rdf:resource="#Encaisseur"/>
    <rdfs:range rdf:resource="#Recette"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Encaisseur">
    <owl:equivalentClass rdf:resource="#Collegue"/>
    <rdfs:subClassOf rdf:resource="#Comptable"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="estPay&#233;Par">
    <rdfs:domain rdf:resource="#Recette"/>
    <owl:inverseOf rdf:resource="#Payer"/>
    <rdfs:range rdf:resource="#Redevable"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="EtatCheque">
    <rdfs:domain rdf:resource="#Ch&#232;que"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="exercice">
    <rdfs:domain rdf:resource="#RubriqueBudgetaire"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="Joint">
    <rdfs:domain rdf:resource="#Recette"/>
    <rdfs:range rdf:resource="#Ch&#232;que"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="LibelleCompte">
    <rdfs:domain rdf:resource="#Compte"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="Lie">
    <rdfs:domain rdf:resource="#Compte"/>
    <rdfs:range rdf:resource="#LoiFinance"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="LoiFinance">
    <owl:equivalentClass rdf:resource="#RubriqueBudgetaire"/>
```

```
    </owl:Class>
    <owl:DatatypeProperty rdf:ID="montant">
        <rdfs:domain rdf:resource="#Recette"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="MontantCheque">
        <rdfs:domain rdf:resource="#Ch&#232;que"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="Nom">
        <rdfs:domain rdf:resource="#Redevable"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="NumeroCheque">
        <rdfs:domain rdf:resource="#Ch&#232;que"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="num&#233;ro">
        <rdfs:domain rdf:resource="#Recette"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Ordonnateur"/>
    <owl:ObjectProperty rdf:ID="Payer">
        <rdfs:domain rdf:resource="#Redevable"/>
        <owl:inverseOf rdf:resource="#estPay&#233;Par"/>
        <rdfs:range rdf:resource="#Recette"/>
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:ID="Poste">
        <rdfs:domain rdf:resource="#Comptable"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="Pr&#233;nom">
        <rdfs:domain rdf:resource="#Redevable"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:Class rdf:ID="Recette">
        <owl:equivalentClass rdf:resource="#Cr&#233;ance"/>
    </owl:Class>
    <owl:Class rdf:ID="Redevable">
        <owl:equivalentClass rdf:resource="#Contribuable"/>
    </owl:Class>
    <owl:Class rdf:ID="RubriqueBudgetaire">
        <owl:equivalentClass rdf:resource="#LoiFinance"/>
    </owl:Class>
    <owl:DatatypeProperty rdf:ID="Telephone">
        <rdfs:domain rdf:resource="#Redevable"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:ID="TypePaiement">
        <rdfs:domain rdf:resource="#Recette"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
</rdf:RDF>
```

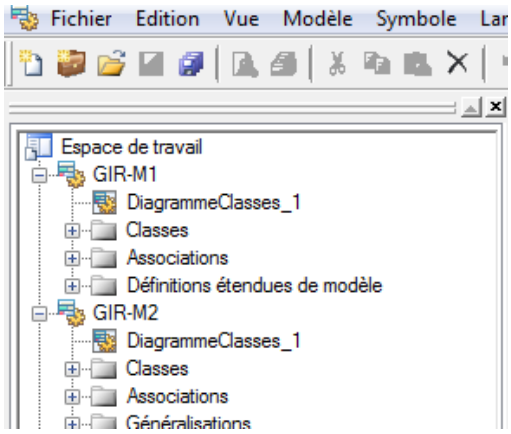
La représentation graphique de l'ontologie e-gov :



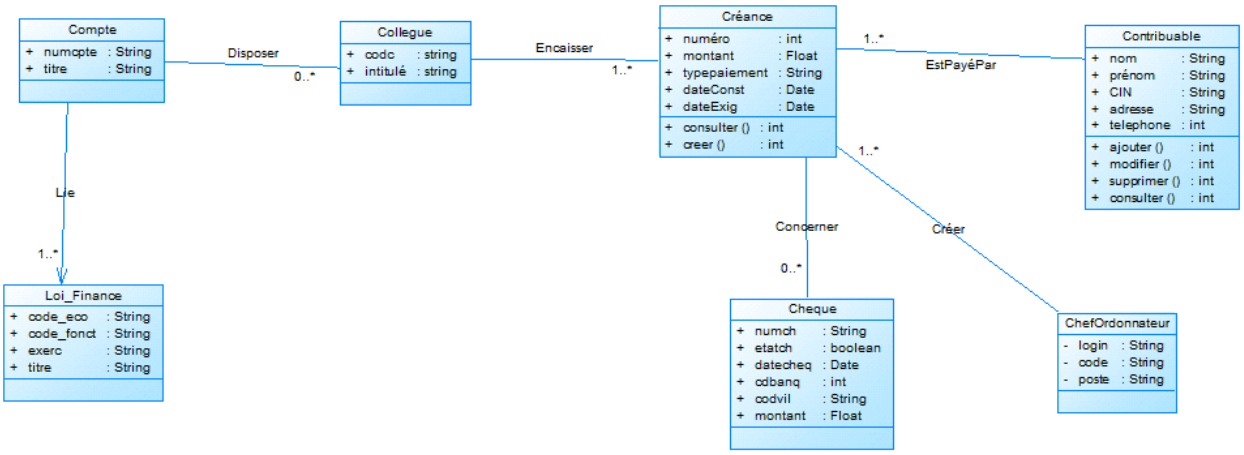

Recette
http://www.owl-ontologies.com/Ontology1347969587.owl#estPayéPar
Instance*
http://www.owl-ontologies.com/Ontology1347969587.owl#Redevable
http://www.owl-ontologies.com/Ontology1347969587.owl#Joint
Instance*
http://www.owl-ontologies.com/Ontology1347969587.owl#Chèque
http://www.owl-ontologies.com/Ontology1347969587.owl#numéro
String*
http://www.owl-ontologies.com/Ontology1347969587.owl#DateEmission
String*
http://www.owl-ontologies.com/Ontology1347969587.owl#TypePaiement
String*
...
estPayéPar*
Payer*
Redevable
http://www.owl-ontologies.com/Ontology1347969587.owl#Payer
Instance*
http://www.owl-ontologies.com/Ontology1347969587.owl#Recette
http://www.owl-ontologies.com/Ontology1347969587.owl#Telephone
Integer*
http://www.owl-ontologies.com/Ontology1347969587.owl#email
String*
http://www.owl-ontologies.com/Ontology1347969587.owl#Adresse
String*
http://www.owl-ontologies.com/Ontology1347969587.owl#CIN
String*
...
Joint
isa
isa
Contribuable
isa
isa
Chèque
http://www.owl-ontologies.com/Ontology1347969587.owl#CodeVille
Integer*
http://www.owl-ontologies.com/Ontology1347969587.owl#DateCheque
String*
http://www.owl-ontologies.com/Ontology1347969587.owl#MontantCheque
Float*
http://www.owl-ontologies.com/Ontology1347969587.owl#CompteEmetteur
String*
http://www.owl-ontologies.com/Ontology1347969587.owl#CompteRecepteur
String*
...
Ordonnateur
http://www.owl-ontologies.com/Ontology1347969587.owl#Créer
Instance*
http://www.owl-ontologies.com/Ontology1347969587.owl#Créance
Créer*
Créance

Annexe 3 : Fonctionnement de Co-Models

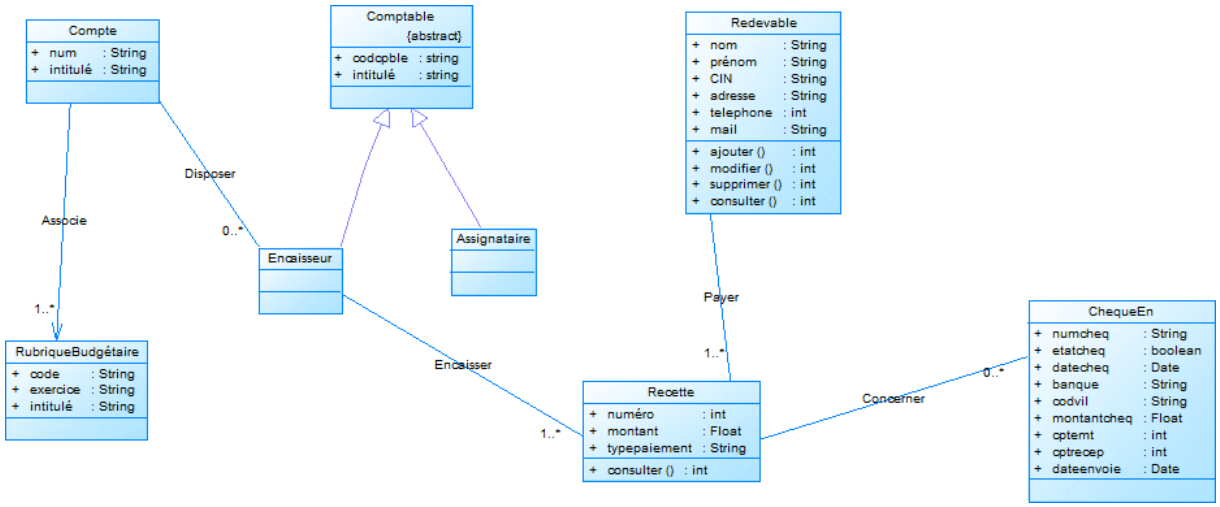
- Espace de travail de Power AMC : Les deux modèles M1 et M2 :



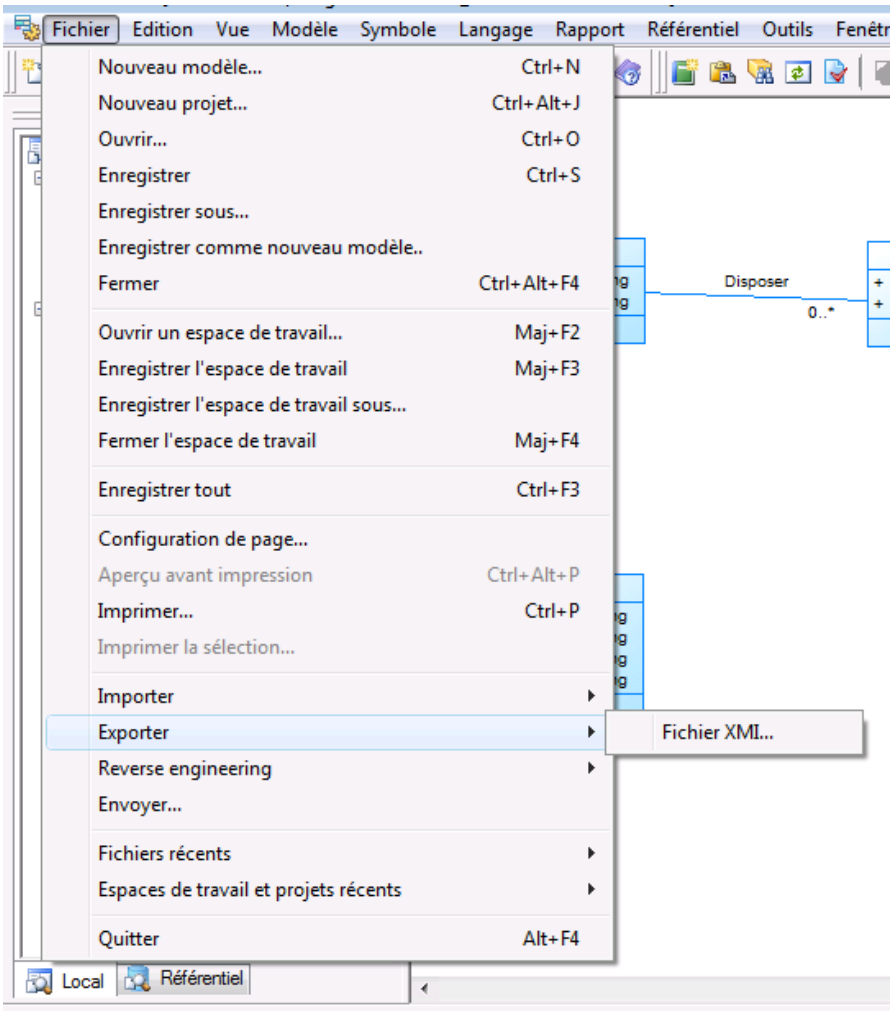
Modèle M1 :



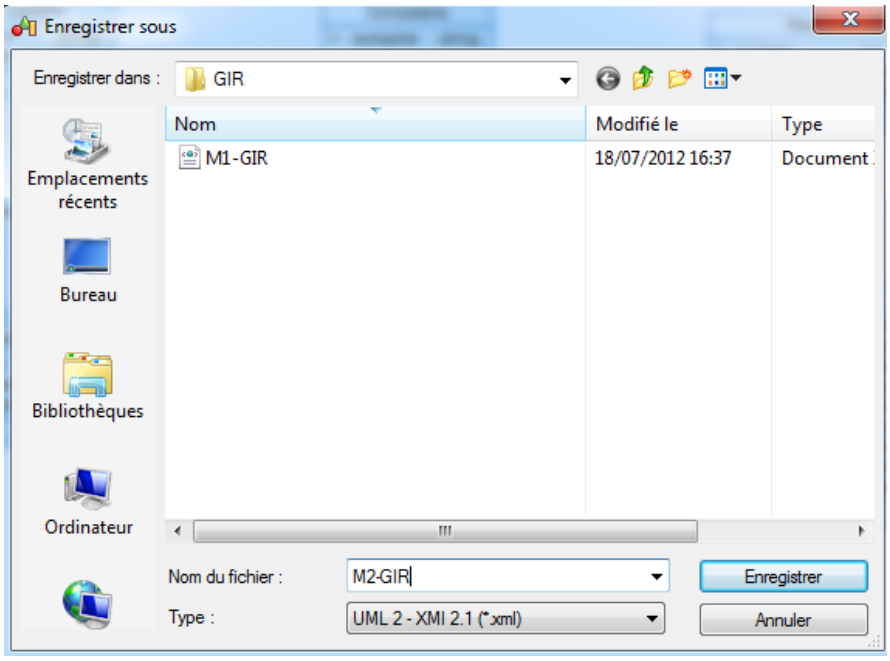
Modèle M2 :



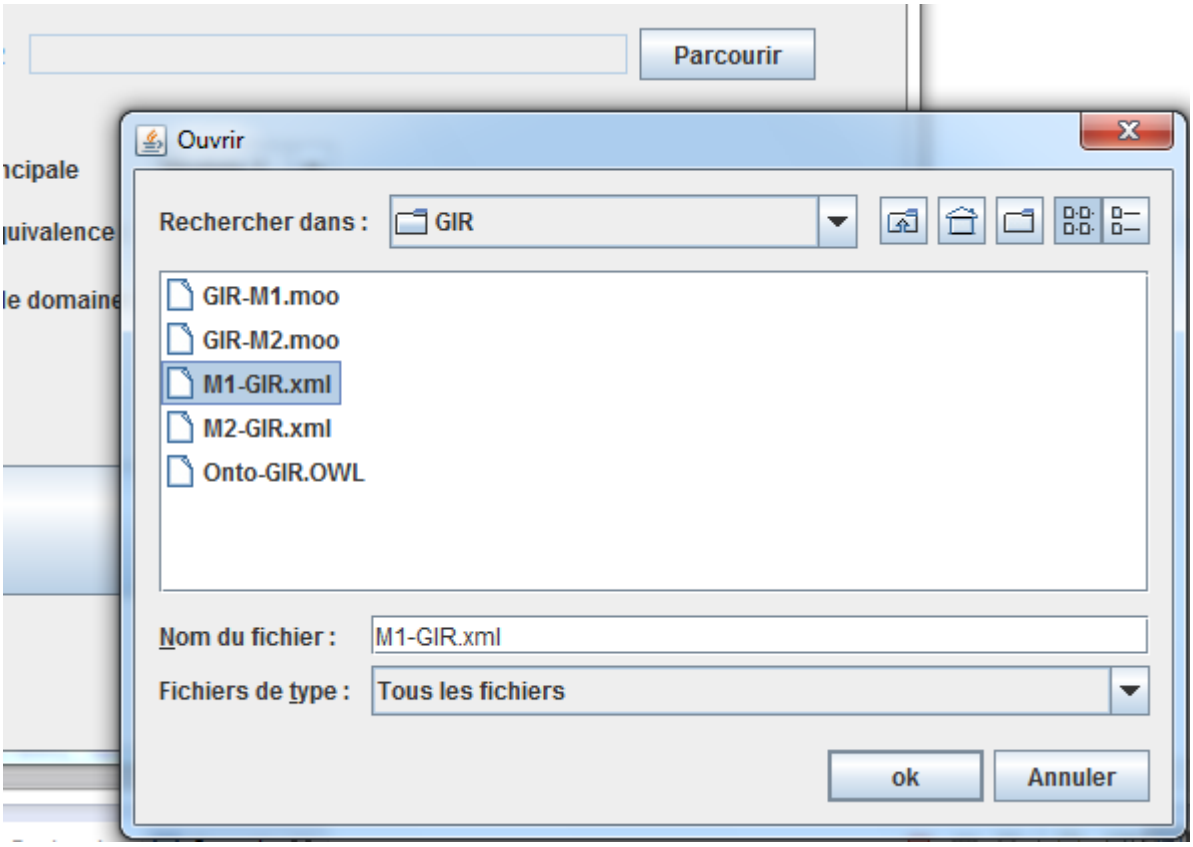
- Power AMC : Importation des modèles UML en fichiers XMI



- Power AMC : Enregistrement des fichiers XMI

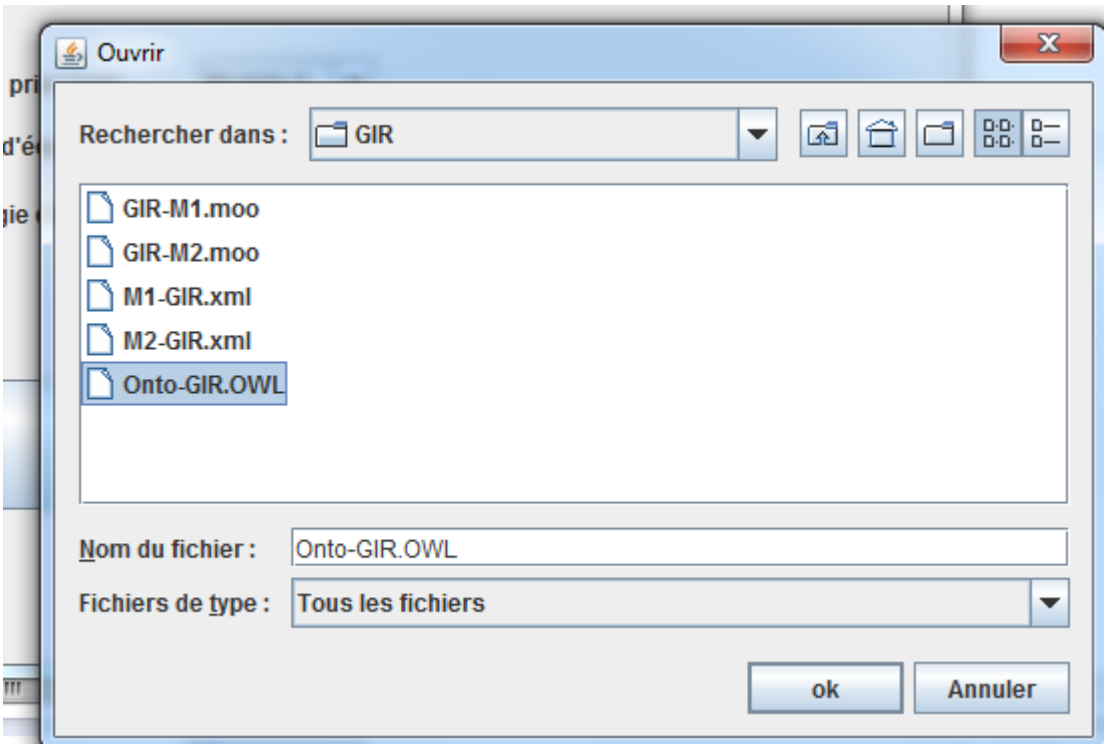


- Au niveau de CO-MODELS : importation des modèles

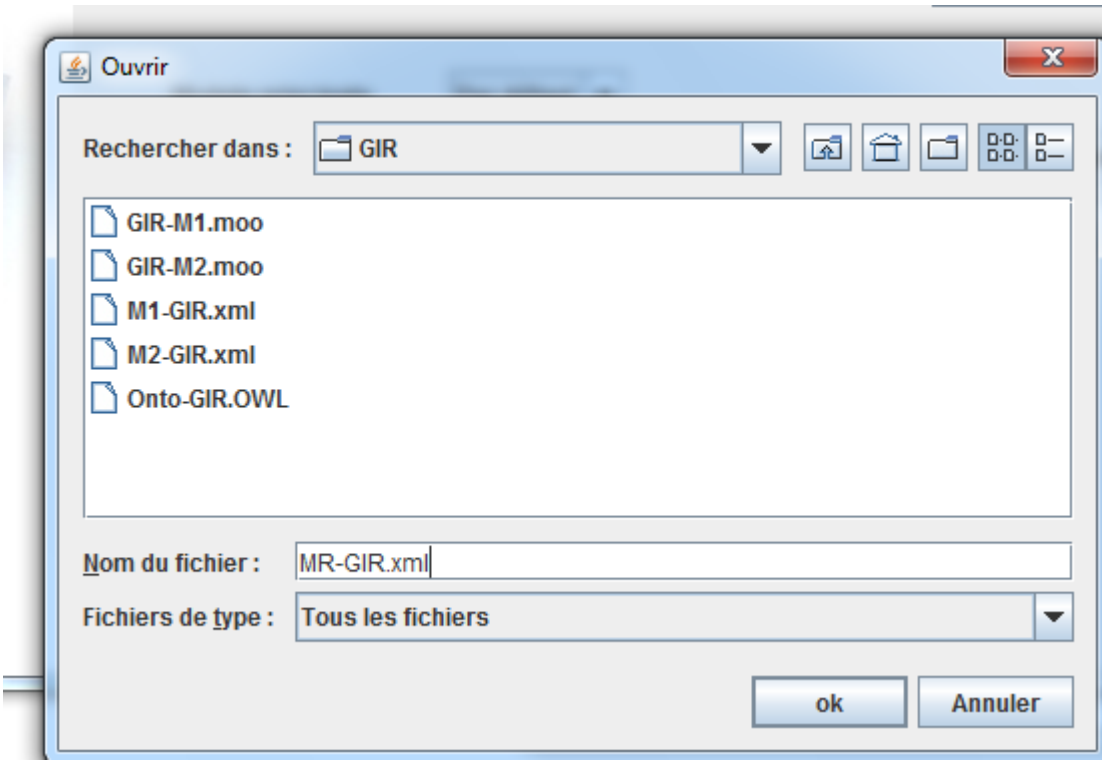


- Au niveau de CO-MODELS : choix du modèle principal

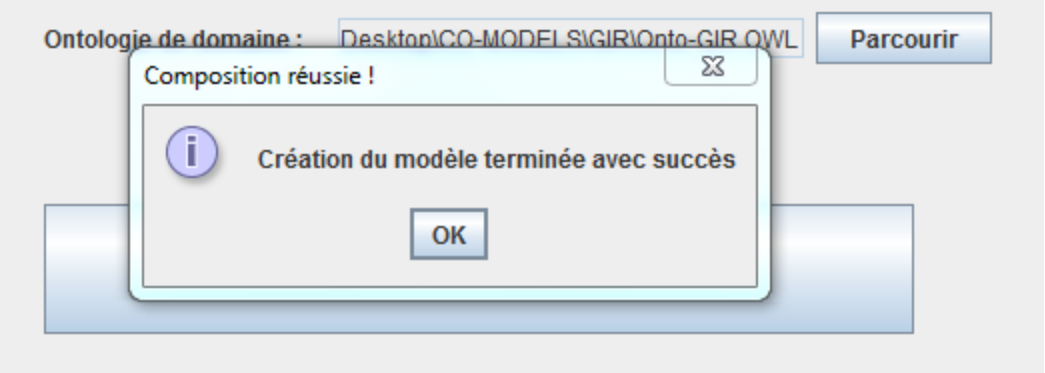
- Au niveau de CO-MODELS : importation de l'ontologie de domine



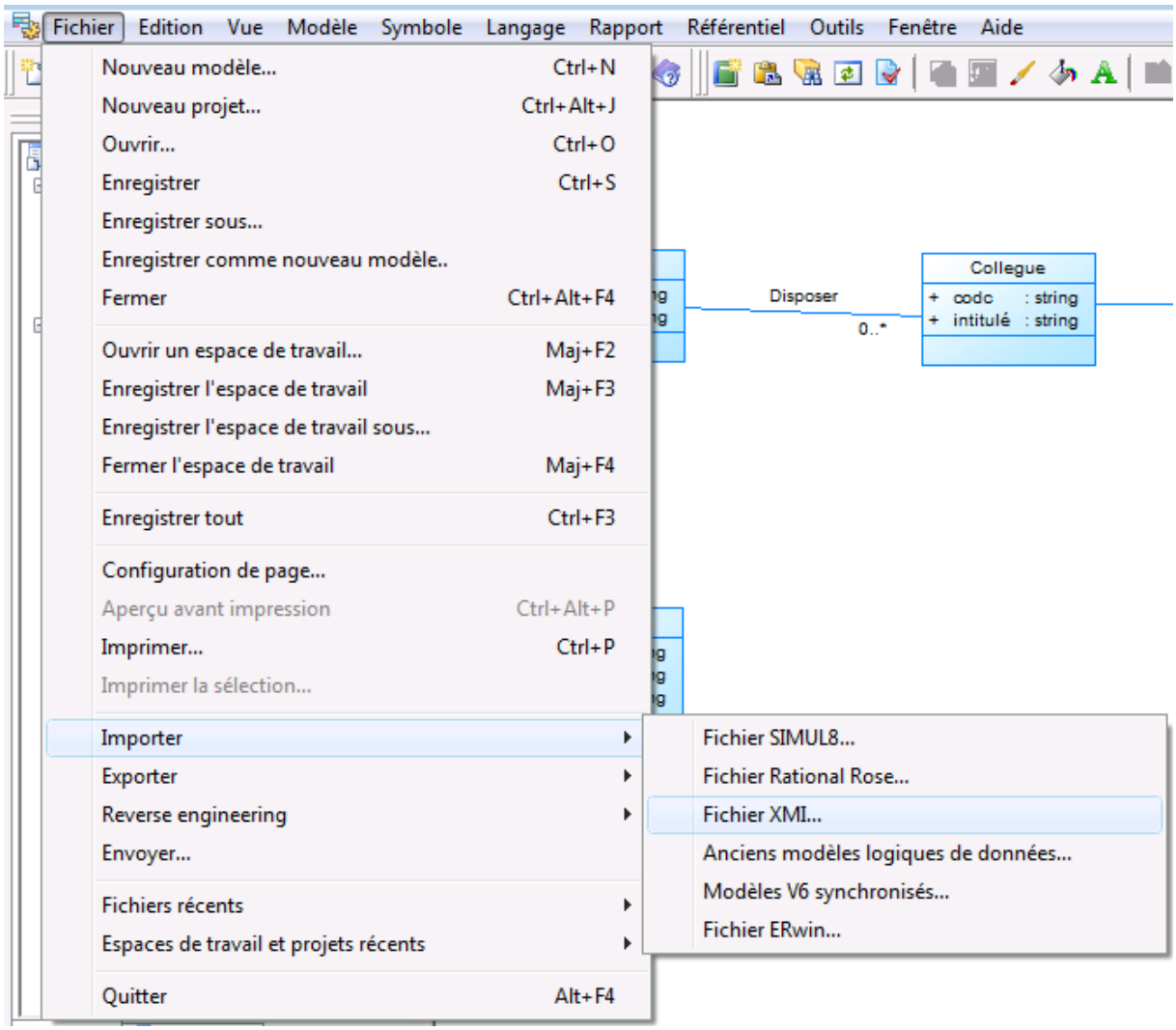
- Au niveau de CO-MODELS : nommer le modèle résultat

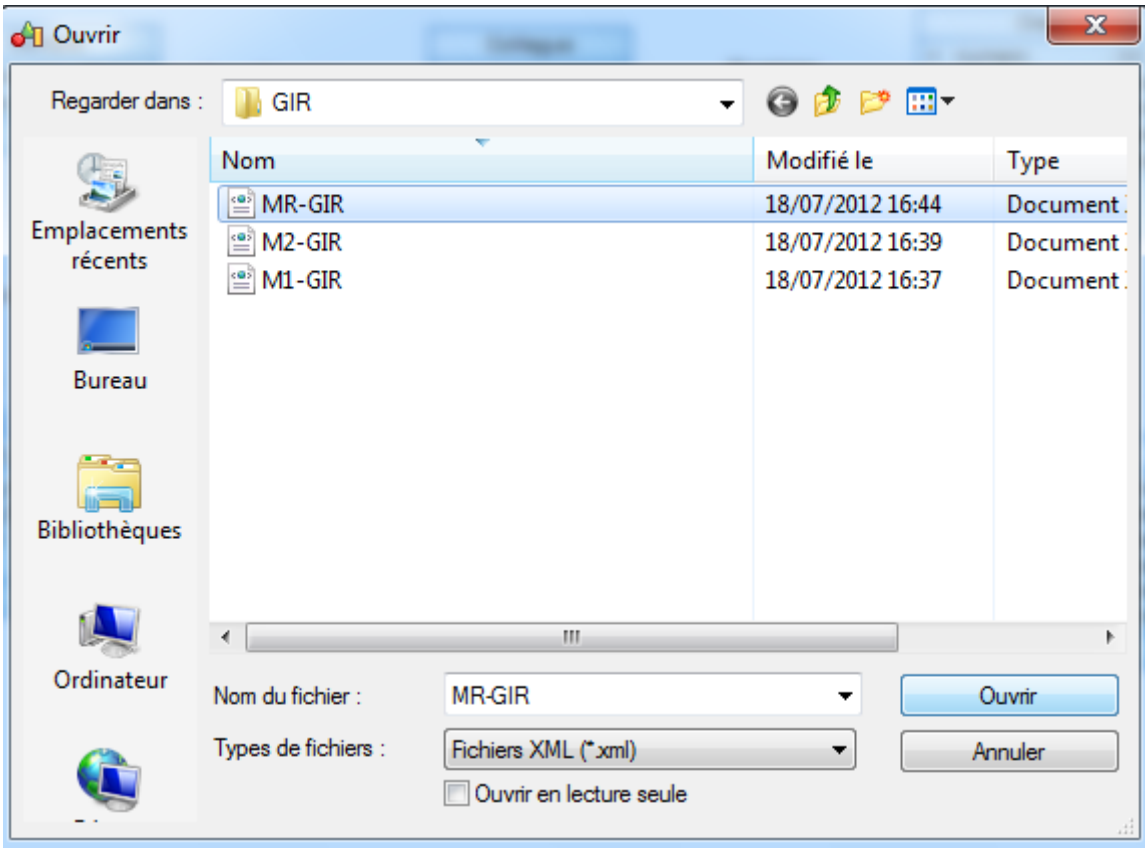


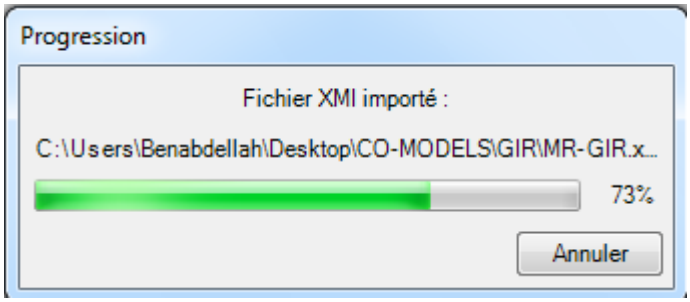
- Au niveau de CO-MODELS : fenêtre de création du modèle résultat avec succès



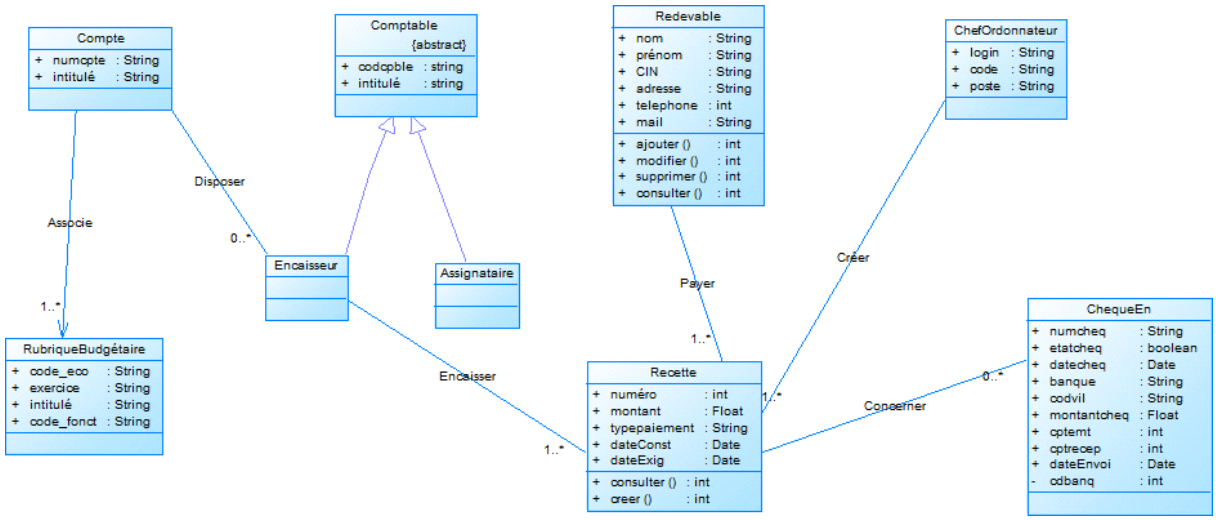
- Au niveau de POWER AMC : Importation du modèle résultat en XMI vers un modèle UML(extension : .moo).







- Au niveau de POWER AMC : Visualisation du modèle résultat



Annexe 4 : Expérimentations

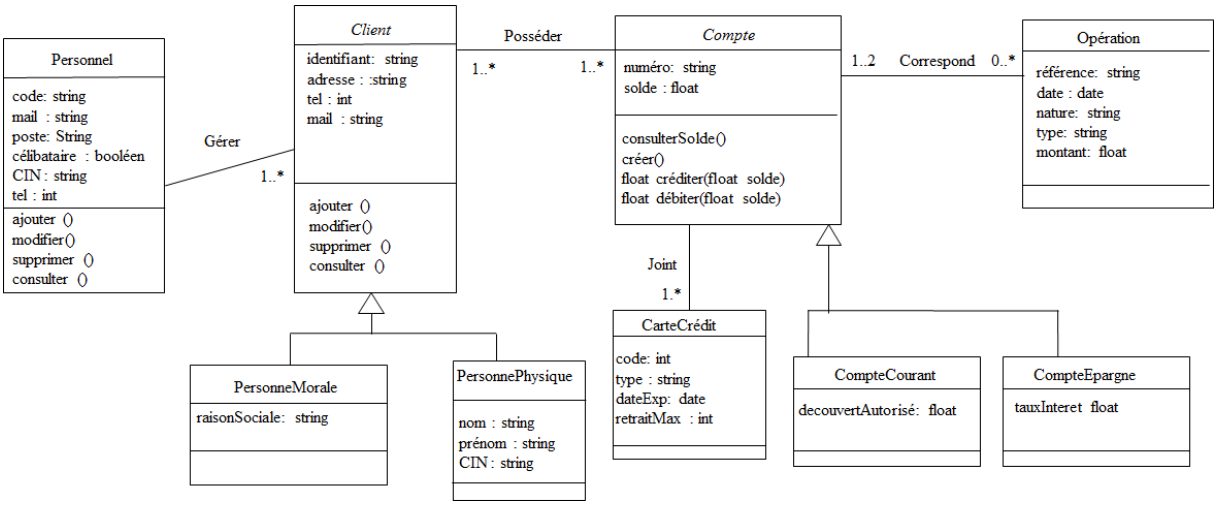
Profil de l'utilisateur

- Profession
- Spécialité
- Age
- Sexe
- Avez-vous l'habitude de travailler avec les modèles UML ?
  - Souvent
  - Assez souvent
  - Parfois
  - Jamais
- Faites-vous parti d'une équipe qui utilise la composition de modèles
  - Oui
  - Non
- Si, oui, comment, est-elle réalisée ?
  - Manuellement
  - Automatiquement, à l'aide d'un outil
- Est-ce que vous avez une idée de ce que c'est qu'une ontologie ?
  - Notions
  - Niveau Moyen
  - Maitrise

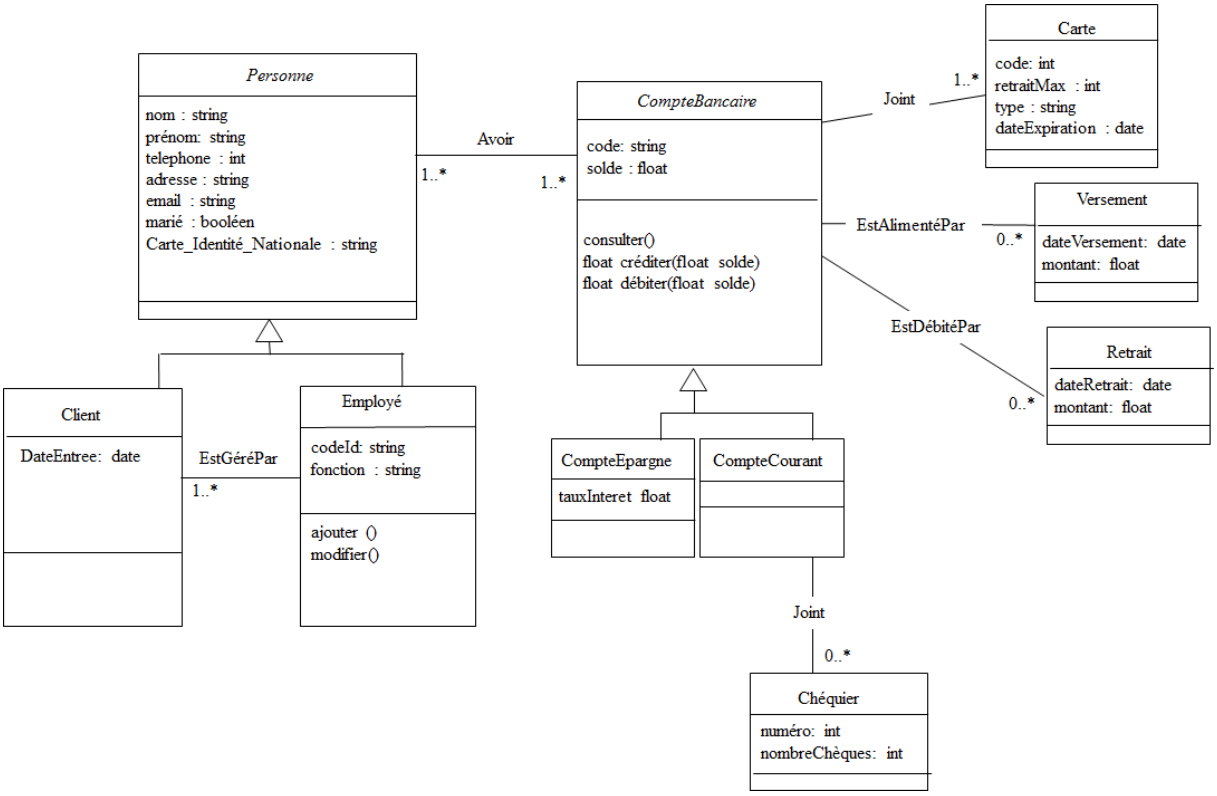
1ère partie : Test manuel de la composition de modèles

Objectif : fusionner les deux diagrammes de classes UML ci-dessous issus du domaine du bancaire afin d'avoir un nouveau modèle résultat fusionné.

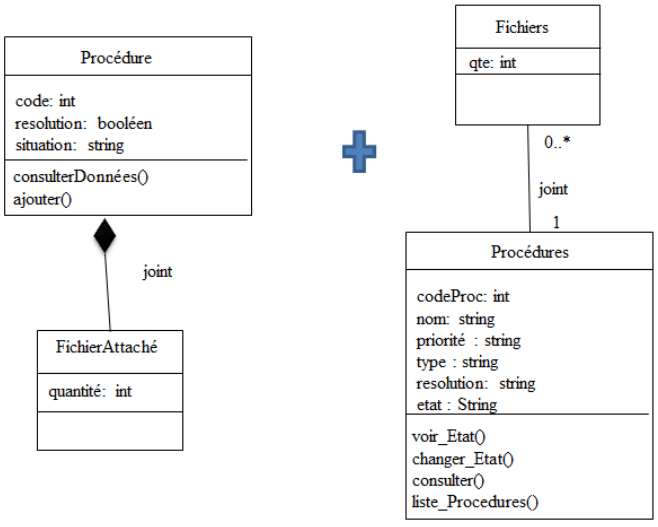
1er modèle



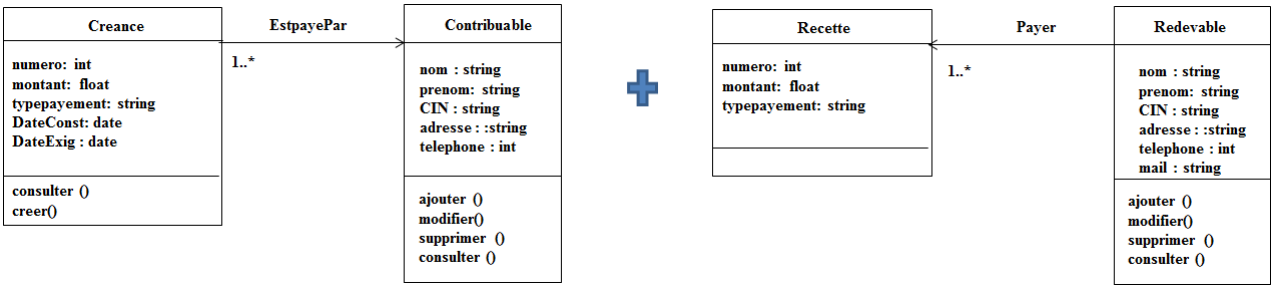
2ème modèle



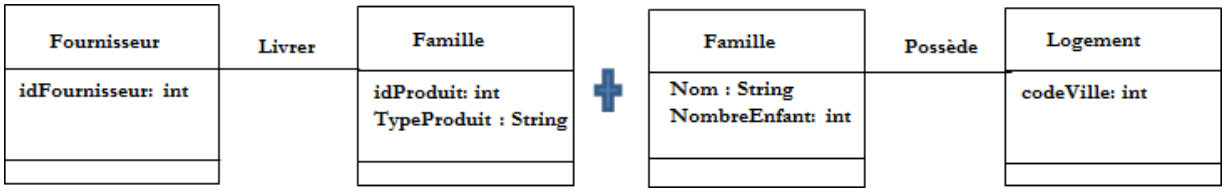
- Préciser le temps de réalisation que vous avez utilisé pour une composition manuelle.
- Réaliser la composition des trois cas suivants :
    - Cas 1 :



- Cas 2 :



- Cas 3



3ème partie : Test de l'outil

- Lors d'une éventuelle composition de modèles, utiliseriez-vous cet outil ?
  - Oui
  - Non
  - Ne sait pas
- A quelle mesure trouvez- vous cet outil utile ?
  - Pas du tout utile
  - Moyennement utile
  - Très utile
- Comment trouvez-vous l'utilisation de l'outil ?
  - Simple
  - Moyennement simple
  - Pas simple
- Est-ce que vous estimez que c'est un gain significatif en termes de temps ?
  - Oui
  - Moyennement
  - Non

Annexe 5 : Modèle Résultat final de l'étude de cas bancaire

Ci-dessous le modèle résultat des deux modèles bancaire, modifié par l'utilisateur.