Vers une approche à base de règles pour la composition de modèles. Application au profil VUML.[1]

Adil Anwar*,*—Sophie Ebersold*— Bernard Coulette*— Mahmoud Nassar**—Abdelaziz Kriouile****

** Laboratoire IRIT, Université de Toulouse II le Mirail 5, allées A. Machado 31058 Toulouse France {anwar, coulette, ebersold}@univ-tlse2.fr*
*** Laboratoire SI2M, ENSIAS, B.P. 713 Agdal – Rabat, Maroc {nassar, kriouile}@ensias.ma*
****Laboratoire LRIMIARF, Université Mohammed V-Agdal, Faculté des sciences Rabat, Maroc*

RÉSUMÉ. *La modélisation par point 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 développé dans notre équipe. Selon la démarche de développement associée à VUML, plusieurs modèles de conception doivent être composés (fusionnés) pour obtenir un modèle global du système. L'objectif de cet article est d'appliquer les principes de l'Ingénierie Dirigée par les Modèles pour automatiser cette composition. Plus précisément, nous définissons la composition de modèles statiques UML à l'aide de règles de transformation déclinées en règles de correspondance, de composition et de translation. Ces règles permettent d'abord de mettre en correspondance les modèles en entrée, puis de fusionner les modèles afin de produire un modèle global VUML. Nous illustrons l'approche tout au long de l'article par un système de gestion de dossiers médicaux partagés.*

ABSTRACT. *Viewpoint-oriented modelling allows to model separately the needs of actors interacting with the system to study. This approach was applied in the VUML profile developed in our team. In the process associated to VUML, several design models have to be composed (merged) to produce a global model of the system. The goal of this paper is to apply the Model Driven Engineering principles to automate this composition. More precisely, we define the composition of static UML models through transformation rules refined as correspondency, composition and translation rules. These rules allow first to establish correspondencies between input models, then to merge these models so as to produce the global VUML model. The approach is illustrated all along the paper by a Shared Medical Files Management System.*

MOTS-CLÉS : *Composition de modèles, points de vue, profil VUML, transformation, règles de composition, processus de composition, relations de correspondance.*

KEYWORDS: *Model composition, viewpoints,, VUML profile, transformation, composition rules, composition process, correspondence relationships.*

[1] Cet article est une extension significative de l'article publié à IDM 2007 à Toulouse. Nous développons ici les différentes règles de transformation nécessaires pour réaliser la composition de modèles dans le contexte du profil VUML, et nous en donnons une implantation en ATL.

1. Introduction

Dans un contexte de complexité croissante des systèmes logiciels, la multi-modélisation devient une activité majeure du processus de développement, notamment dans la phase de conception où plusieurs modèles sont généralement produits et doivent cohabiter et/ou être composés pour obtenir un modèle global cohérent du système. Plusieurs approches s'intéressent à la multi-modélisation, que ce soient les approches traditionnelles comme la modélisation par points de vue (Finkelstein et al., 1990) ou la modélisation par sujets (Ossher et al., 1996), ou plus récemment le développement par aspects (Baniassad et al., 2004) (Reddy et al., 2006), et l'ingénierie multi-modèles (Muller et al., 2007). Même si ces travaux sont reconnus ou prometteurs, l'automatisation de la composition de modèles (appelée aussi fréquemment *fusion*) constitue encore un enjeu majeur du Génie Logiciel.

Le cadre général de nos travaux de recherche a pour objectif la définition et la mise en œuvre d'une méthodologie de développement orientée points de vue. C'est dans cette optique que nous avons défini un profil UML appelé VUML (*V*iew based *U*nified *M*odeling *L*anguage) (Nassar et al., 2005ab). VUML offre un formalisme pour modéliser un système logiciel par une approche combinant objets et points de vue. Sur le plan méthodologique, VUML propose une démarche centrée utilisateur qui permet d'intégrer la notion de point de vue dans le processus de développement des systèmes. Dans cette approche, plusieurs modèles de conception modélisant les besoins de chaque acteur interagissant avec le système sont développés par différents concepteurs. Ces modèles doivent être ensuite composés (fusionnés) pour produire un modèle VUML unique représentant le modèle de conception global du système. L'ajout d'un point de vue (par exemple pour prendre en compte un nouvel acteur) peut nécessiter le développement d'un modèle de conception supplémentaire qui devra être composé à son tour avec le modèle VUML existant. Nous avons fait une étude préliminaire de la fusion de modèles au sein du profil VUML (Nassar et al., 2005b), mais à ce jour, en l'absence d'une approche systématique et réutilisable, nous n'avons pas encore pu la formaliser et donc l'automatiser. En effet, la tâche de fusion — même réduite dans un premier temps à la fusion de deux diagrammes de classes— est un processus relativement complexe qui consiste, en étant synthétique à : (i) harmoniser les diagrammes de classes de façon à éliminer les éventuelles conflits résultant de modélisations séparées ; (ii) mettre les modèles en correspondance pour identifier d'une part les classes identiques qui doivent rester comme telles dans le modèle final, d'autre part les classes devant être fusionnées pour produire une classe multivue, et enfin les relations entre classes qui peuvent varier d'un modèle à l'autre. Dans le cas de différences entre modèles sur les relations entre classes de même nom — par exemple association versus spécialisation —, il est nécessaire d'appliquer une heuristique fondée éventuellement sur un patron de conception ou de demander l'intervention du concepteur qui contrôle la fusion ; (iii) réaliser la fusion proprement dite en élaborant notamment les classes multivue sous la forme de structures composées d'une base et de vues.

Par ailleurs, depuis la proposition bien connue de l'architecture MDA (Soley et al., 2000), l'Ingénierie Dirigée par les Modèles (IDM) tend à prendre une place grandissante dans la chaîne de développement du logiciel. Elle consiste à centrer les activités de développement sur le paradigme de *modèle* qui est considéré dorénavant comme une entité de première classe (Bézivin et al., 2006). L'un des intérêts de cette approche est de considérer les modèles à différents niveaux d'abstraction afin de faciliter leur réutilisation au cours du processus de développement. L'apport de l'IDM est aussi de conceptualiser le développement sous la forme de transformations de modèles, ces entités (aussi bien les modèles que les transformations) étant décrites conformément à des métamodèles.

Face à la problématique de la composition de modèles, l'IDM apparaît comme une solution prometteuse puisque l'on peut considérer certaines étapes de la composition de modèles comme des transformations de modèles. Pour ces raisons, nous souhaitons utiliser l'approche IDM et en particulier la notion de transformation pour automatiser en partie la composition de modèles en VUML.

Dans cet article, nous circonscrivons l'étude à la composition de deux modèles UML statiques (les diagrammes de classes) pour produire un modèle VUML global, sachant que l'approche adoptée est généralisable à la composition de plusieurs modèles UML et/ou VUML.

La principale contribution de ce travail, qui étend l'article proposé dans (Anwar et al., 2007), est d'expliciter et implémenter les règles de transformation pour effectuer la composition de modèles dans le contexte VUML. Ce travail repose en premier lieu sur l'identification d'un ensemble de relations de correspondance entre les modèles à composer. Ces relations sont stockées dans un *modèle de correspondance*, qui est ensuite fourni à l'activité de composition proprement dite.

Dans la section 2 de l'article, nous présentons le cadre général motivant cette recherche. Pour cela nous commençons par donner une brève présentation du profil VUML, en illustrant la démarche à travers l'exemple d'un Système de Gestion de Dossiers Médicaux Partagés (SGDMP) qui servira de guide applicatif tout au long de l'article. Dans la section 3, nous présentons le processus de composition de modèles associé au profil VUML, qui met en évidence les différentes étapes de la composition. La section 4 présente les règles de composition et la sémantique de leur exécution. Nous discutons les détails de l'implantation de l'approche dans la section 5. La section 6 décrit les travaux connexes sur la composition de modèles. La section 7 est consacrée à une discussion sur les limites et les questions posées par notre approche, dans un objectif prospectif. Nous concluons cet article en récapitulant son apport, et en présentant les travaux en cours et futurs.

2. Modélisation avec VUML et exemple d'application

2.1 *Principes du profil VUML*

VUML est un profil UML qui a été développé récemment dans notre équipe (Nassar et al., 2005). Son objectif est de fournir un formalisme adapté à la modélisation objet par points de vue. Un point de vue, associé à un acteur, exprime une perspective selon laquelle un système peut être modélisé pour décrire les besoins de l'acteur considéré. A ce jour, VUML ne supporte que la modélisation statique et principalement le diagramme de classes, mais des travaux sont en cours sur la modélisation comportementale multivue. L'ajout principal à UML est le concept de classe multivue qui est composée d'une base (partie partagée par tous les points de vue) et d'un ensemble de vues (extensions de la base). VUML offre des mécanismes pour gérer les droits d'accès aux caractéristiques des classes multivue, spécialiser une classe multivue, spécifier les dépendances entre les vues, assurer la cohérence du modèle en cas de mises à jour, administrer les vues à l'exécution, etc. A l'instar d'UML, la sémantique de VUML est décrite par un métamodèle, des règles de bonne formation exprimées en OCL (OMG, 2003a), et des descriptions textuelles informelles. Sur le plan méthodologique, nous avons proposé une démarche d'analyse/conception permettant de développer un système de l'analyse jusqu'à l'implantation.

2.2 *Démarche de conception associée au profil VUML*

Cette démarche comporte cinq phases principales (Figure 1). La première phase est une analyse générale visant à définir les concepts de base déterminant le domaine d'application. A l'issue de cette phase le référentiel du système est établi. La deuxième phase est une phase d'analyse des besoins qui consiste à identifier les acteurs et leurs besoins en terme d'activités. La troisième phase est une phase de conception décentralisée, au cours de laquelle plusieurs (équipes de) concepteurs peuvent travailler séparément pour réaliser des modèles de conception par point de vue (appelés aussi *modèles partiels*). La quatrième phase est une phase de résolution des conflits entre modèles partiels. Nous nous intéressons pour l'instant aux conflits d'ordre syntaxique liés aux problèmes d'appellation des éléments de modélisation, et aux incohérences structurelles (Anwar et al., 2006). La cinquième phase, la phase de composition proprement dite, consiste à fusionner les modèles partiels afin d'obtenir le modèle global VUML.

2.3 *Illustration de la démarche sur un cas d'étude*

Pour illustrer notre approche, nous nous appuyons sur la modélisation d'un Système de Gestion de Dossiers Médicaux Partagés (SGDMP) (Anwar et al., 2006).

Pour simplifier, nous limitons notre étude aux acteurs et activités suivants :

- Les patients suivent des traitements, effectuent des consultations chez les médecins et passent des examens et des tests dans les laboratoires d'analyse. Ils peuvent consulter leur dossier.
- Les médecins effectuent des diagnostics et des examens, rédigent des rapports de consultation, prescrivent des médicaments, consultent des rapports d'antécédents médicaux, etc.

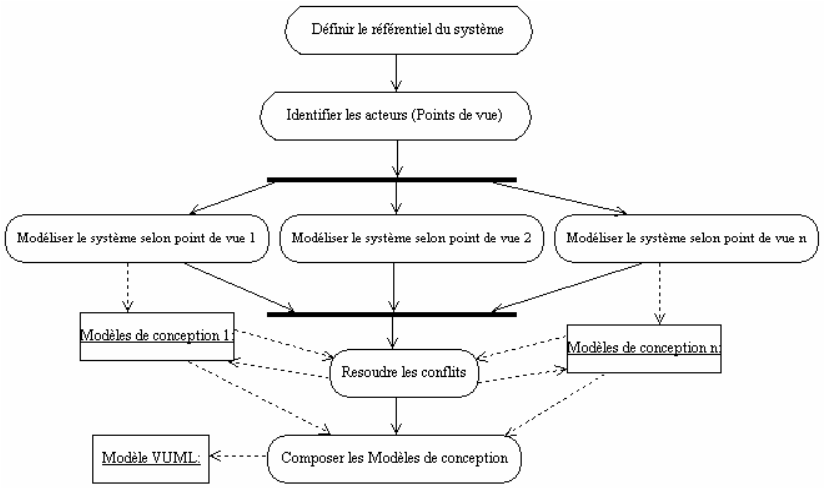

Figure 1. *Démarche de conception VUML*

2.3.1 Modélisation des exigences

La première phase d'analyse du système consiste à capturer ses exigences fonctionnelles. Cette phase centrée sur les acteurs donne lieu à un diagramme de cas d'utilisation par point de vue. La figure 2 ci-dessous illustre un sous-ensemble des cas d'utilisation identifiés pour le point de vue *médecin*.

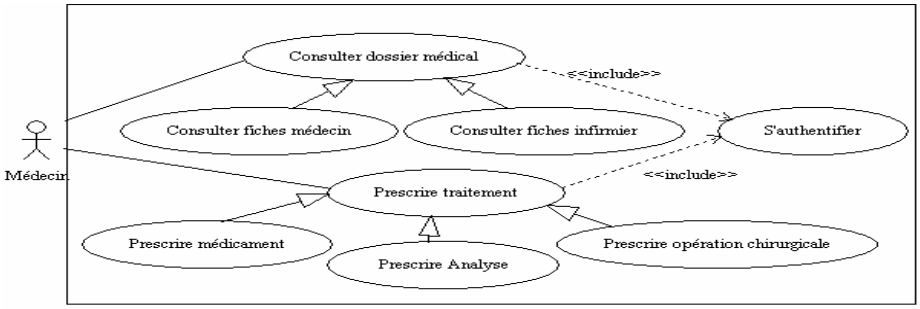

Figure 2. Diagramme de *Cas d'utilisation du point de vue* médecin *(extrait)*

2.3.2 Modélisation UML par point de vue

Il s'agit de modéliser le système selon un point de vue donné. Prenons dans le point de vue *médecin*, l'exemple du scénario «Enregistrer un Traitement» du cas d'utilisation «Prescrire Traitement». Ce scénario, déclenché par le médecin suite à une consultation, a pour but de créer une instance de la classe 'ConsultationForm' qui sert à enregistrer tous les actes cliniques et les décisions prises par le médecin lors de la consultation. En procédant de façon incrémentale avec les différents cas d'utilisation, les objets identifiés, ainsi que les méthodes associées figurant dans les diagrammes de séquences, permettent de construire le diagramme de classes du point de vue médecin (Figure 3) qui contient les informations pertinentes auxquelles peut accéder l'acteur associé. Nous pouvons remarquer par exemple que le médecin a accès aux informations sur les analyses effectuées dans les laboratoires, ainsi qu'aux rapports créés par d'autres professionnels de santé (médecin, infirmiers, etc).

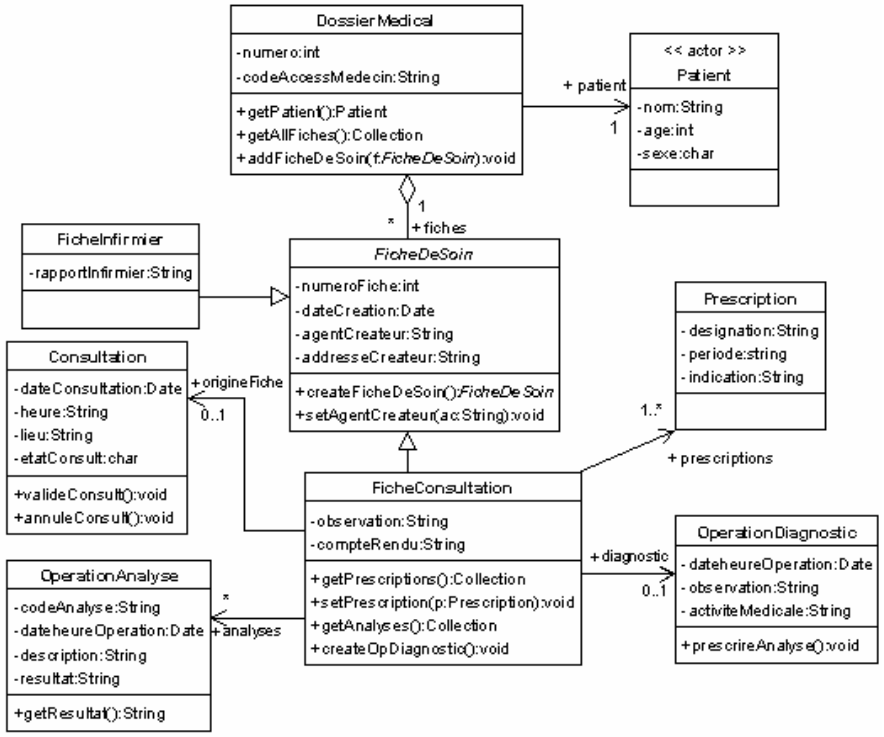

Figure 3 *Extrait du digramme de classes UML - Point de vue médecin*

Une démarche similaire effectuée pour le point de vue *patient* donne lieu au diagramme de classes de la figure 4. Il apparaît sur ce diagramme que le patient peut accéder aux prescriptions des médecins qui le concernent, à la liste des fiches de paiement associées à ses traitements, mais pas directement aux rapports des laboratoires d'analyses, ni aux rapports rédigés par les infirmiers qui sont réservés au médecin.

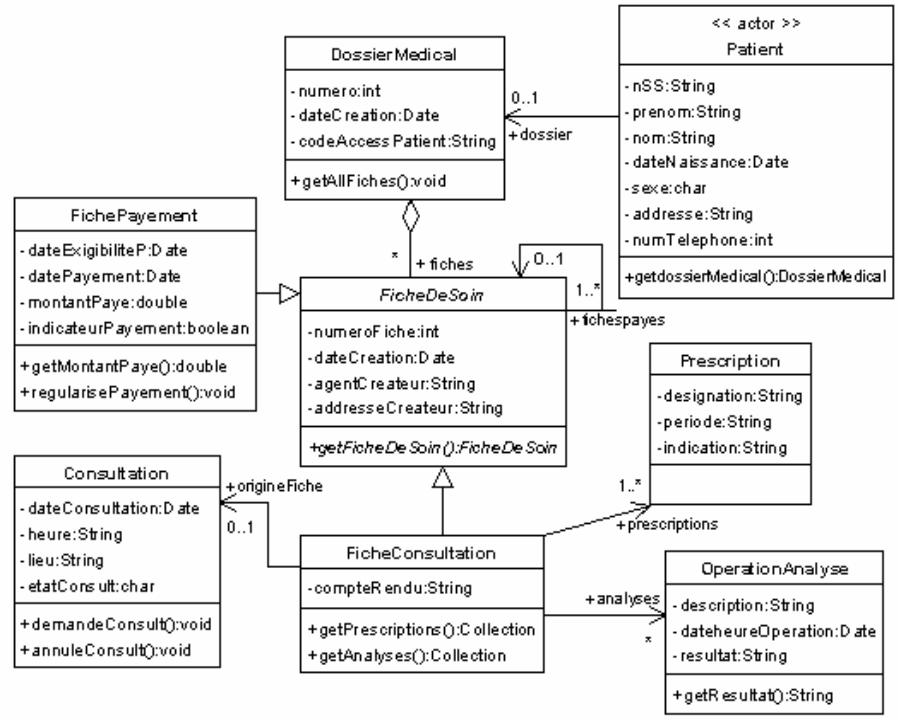

Figure 4 *Extrait du diagramme de classes UML - Point de vue patient*

2.3.3 Modèle VUML de l'étude de cas SGMDP

La figure 5 présente le modèle VUML résultant de la composition des deux modèles correspondant aux points de vue du médecin et du patient (figures 3 et 4). Les classes apparaissant dans deux modèles par point de vue avec le même nom et des propriétés différentes (attributs, opérations, etc.) sont fusionnées en une classe multivue.

Pour illustrer le concept de classe multivue de VUML, nous choisissons les entités DossierMedical et FicheDeSoin. On remarque que les propriétés communes

aux classes DossierMedical des figures 3 et 4 sont regroupés dans la classe stéréotypée « base » ; par contre les classes stéréotypées « view » encapsulent les propriétés propres à chaque point de vue. Pour des raisons de lisibilité, certaines classes multivue ne sont pas éclatées (classes stéréotypées par « multiViewsClass »). Nous avons adopté, pour nommer les classes stéréotypées « view » la notation préconisée par le profil VUML (nom de l'acteur + nom de la classe base). Cette stratégie permet d'assurer la traçabilité entre les éléments des modèles partiels et le modèle VUML. La classe Prescription est stéréotypée par « merged » car elle est le résultat d'une fusion de classes identiques dans les 2 modèles par point de vue.

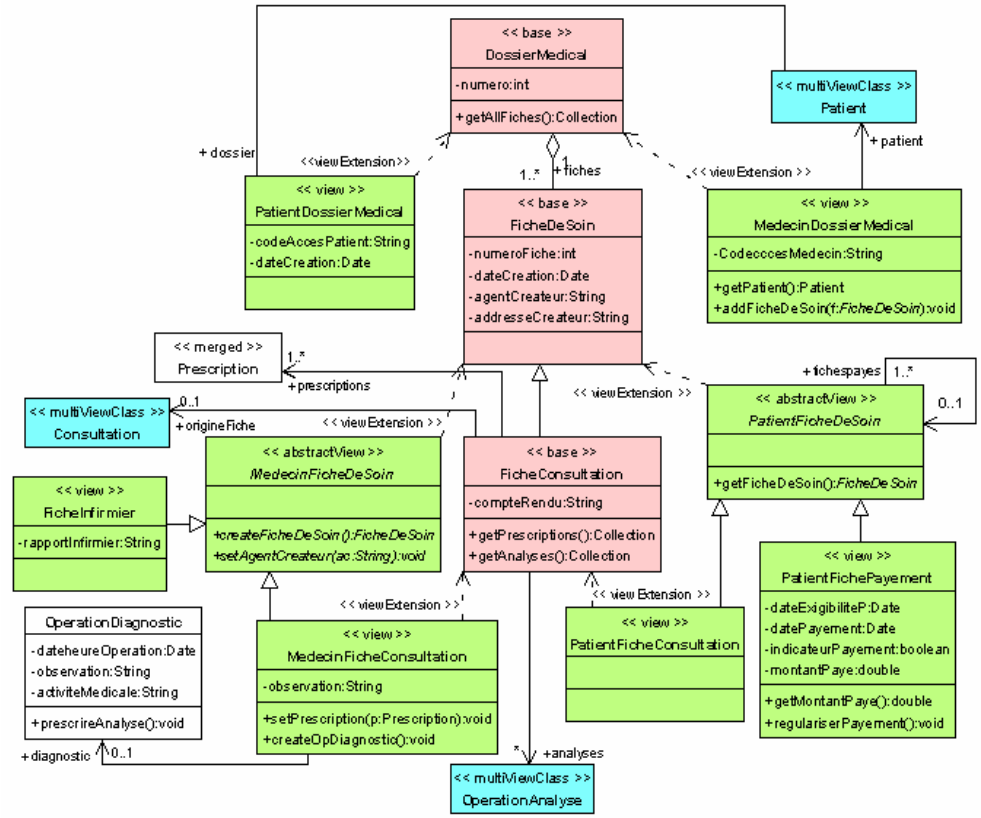

Figure 5 *Extrait du modèle VUML du système SGDMP*

3. Processus de composition de modèles

Nous avons présenté dans la section 2 la démarche d'analyse/conception VUML. Dans cette démarche, la phase de composition de modèles est centrale puisque l'objectif est de produire le modèle VUML global. Dans cette section, nous

détaillons cette phase à travers un processus de composition de modèles représenté par le diagramme d'activités de la figure 6.

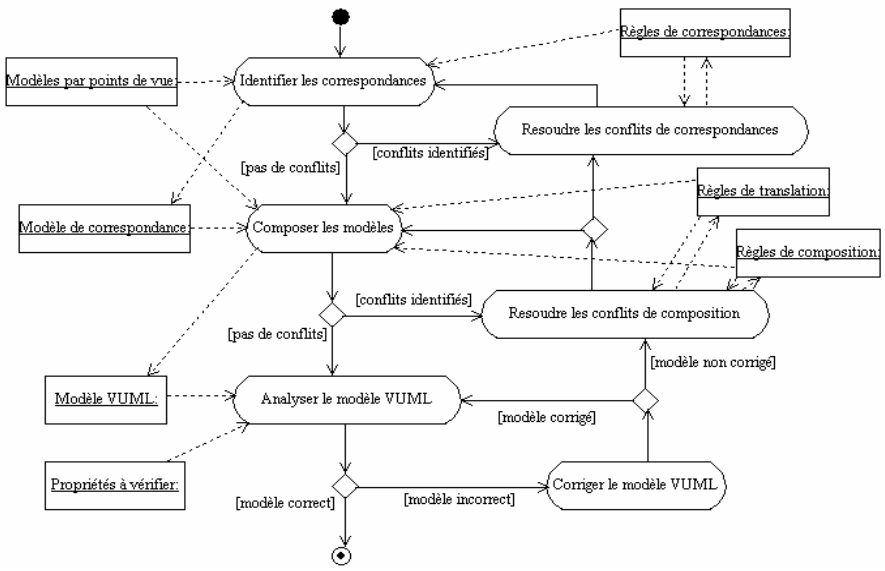

Figure 6 *Processus de composition*

Le processus est composé de trois étapes. La première étape s'appuie sur un ensemble de règles qui permettent d'identifier les relations de correspondance entre éléments des modèles partiels. Par exemple les modèles partiels des figures 3 et 4 seront reliés par deux relations respectivement de similarité et de conformité (que nous définissons dans la section 4 suivante), établies entre deux classes respectivement DossierMedical et Prescription. Les règles de correspondance sont basées sur les propriétés définies dans le métamodèle d'entrée UML (OMG, 2003b). Une autre technique de comparaison consiste à utiliser les identifiants xml (Alanen et al., 2003). Cette technique exige que les éléments comparés soient dérivés de la même entité, et ne peut donc pas s'appliquer ici. La définition des stratégies de comparaison est détaillée dans la section 4. Les différentes relations de correspondance créées au cours de cette étape sont stockées dans un modèle séparé appelé *modèle de correspondances*.

La deuxième étape du processus de composition des modèles consiste à composer les modèles initialement mis en correspondance. Cette étape est automatisable par l'application d'un ensemble de règles (composition, translation). La stratégie de composition s'appuie en effet sur des règles prédéfinies que nous

détaillerons dans la section 4. L'idée est de parcourir les liens de correspondance établis dans l'étape précédente et de déterminer selon le cas les règles de composition à appliquer. Les éléments qui ne sont pas reliés sont simplement retranscrits dans le modèle VUML par le biais de règles de translation. Dans le cas où des conflits de composition ont été révélés durant la composition, le concepteur doit identifier de nouvelles règles à appliquer ou modifier les règles existantes.

Une fois le modèle VUML généré, la troisième étape consiste à vérifier les propriétés syntaxiques et sémantiques propres au profil VUML (Nassar et al., 2005) (par exemple : « une classe stéréotypée par 'view' a au plus un parent », etc.). On peut effectivement analyser la cohérence statique du modèle VUML en vérifiant le respect des règles de bonne formation (W.F.R). Si le modèle VUML présente des incohérences, une activité de *refactoring* est nécessaire afin de corriger les erreurs de composition.

4. Composition de modèles : une approche basée sur des règles de transformation

Pour automatiser le processus de composition, nous avons décidé de l'ancrer dans une approche IDM en explicitant les transformations sous forme de règles. Dans cette section, nous présentons les différentes règles de transformation que nous avons classées en règles de correspondance, règles de composition et règles de translation.

La composition de modèles par point de vue est clairement une suite de transformations de modèles (transformations exogènes puisque les métamodèles en entrée et en sortie sont différents). En effet la première transformation permet de créer un modèle de correspondances dans lequel sont stockés 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. La figure 7 présente le schéma global de la chaîne de transformation.

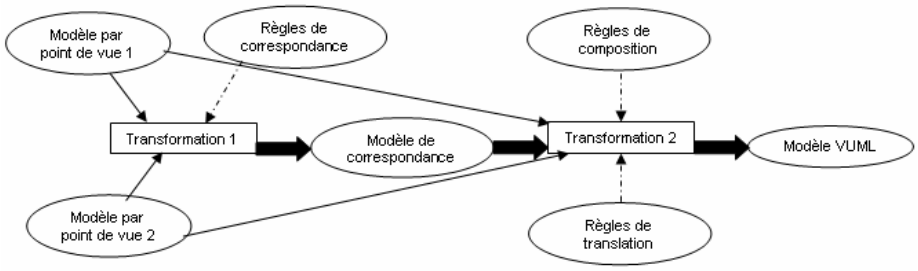

Figure 7 *Chaîne de transformation dans la composition avec VUML*

4.1. *Métamodèle de correspondance*

Les relations de correspondance identifiées dans la première étape du processus de composition sont stockées dans un modèle de correspondances. Les stratégies de comparaison entre les éléments des modèles sont définies par des règles de correspondance. Ces règles sont définies pour chaque type d'élément du métamodèle d'entrée (par exemple UML).

La figure 8 présente un extrait du métamodèle de correspondance. Pour introduire la sémantique de la correspondance entre les éléments, nous avons étendu le métamodèle de AMW (Atlas Model Weaver) défini dans (Didonet et al., 2005), le but étant de fournir un ensemble de modèles appelés 'weaving models'. Ces modèles de tissage peuvent être utilisés par la suite par un outil de transformation. Par ailleurs, le métamodèle de AMW est générique et permet de faire des extensions en définissant de nouveaux types de relations, en fonction du domaine d'application.

Nous présentons dans ce qui suit les éléments clés du métamodèle :

- La métaclasse *CorrespondenceModel* représente le modèle de correspondance qui contient la description des correspondances entre éléments, et des éléments provenant des modèles par point de vue.
- La métaclasse abstraite *CorrespondenceRelationship* définit la notion de relation de correspondance entre les éléments des modèles par point de vue. La définition d'une nouvelle relation de correspondance se fait par un mécanisme de spécialisation de cette métaclasse, ce qui permet de définir la sémantique de chaque relation.
- L'élément *CorrespondanceRelationshipEnd* représente l'extrémité d'une relation de correspondance. Il sert de référence aux éléments des modèles mis en correspondance.
- Enfin, la métaclasse abstraite *ModelElement* représente un élément d'un modèle par point de vue susceptible d'être fusionné avec un élément d'un

autre modèle. Cette métaclasse peut représenter soit des éléments composites (modèles par point de vue, classes, associations), soit des éléments primitifs (attributs, opérations, généralisations).

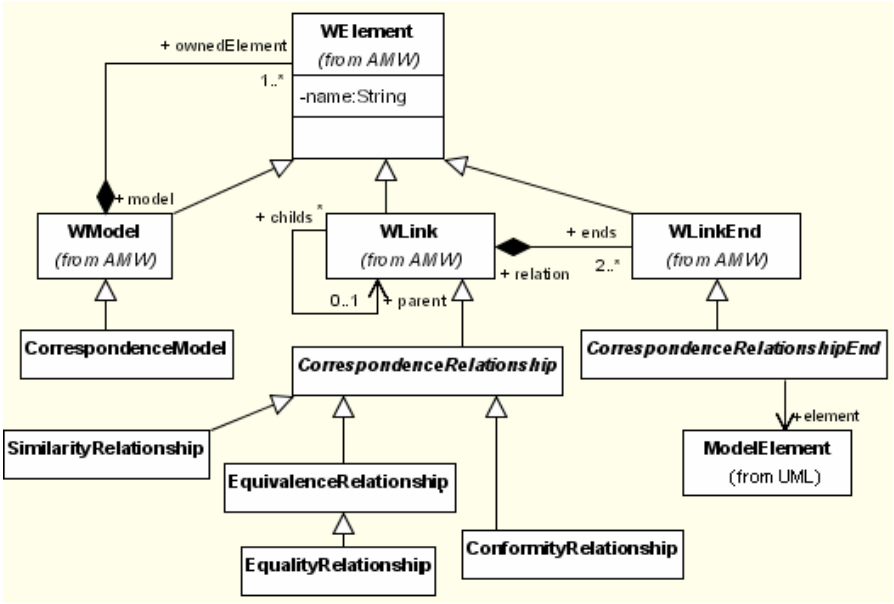

Figure 8 *Extrait du métamodèle de correspondance*

4.2. *Règles de transformation*

Plusieurs stratégies sont possibles pour organiser les règles de transformation (Czarnecki et al., 2003). En fonction du niveau de granularité des éléments auxquels elles s'appliquent, nous distinguons les règles relatives aux éléments composés tels que les classes, et celles qui concernent les éléments primitifs (attributs, opérations, relations). Dans notre approche, les règles de transformation sont organisées en trois catégories. La première est celle des règles de correspondance qui permettent de créer les relations (liens) de correspondance entres les éléments correspondants, en offrant une navigation dans les modèles sources pour obtenir certaines informations nécessaires à la composition. La deuxième catégorie concerne les règles de composition qui exploitent les éléments du modèle de correspondance pour créer les éléments du modèle VUML. Enfin, les règles de translation constituent la troisième catégorie. Elles expriment la transformation des éléments qui n'ont pas été identifiés par les règles de correspondance. Pour illustrer la présentation des règles de composition, nous considérons en entrée deux modèles de conception par points de

vue MPV1 et MPV2 (diagrammes de classe UML). Les exemples sont issus des diagrammes par point de vue présentés sur les figures 3 et 4.

4.2.1. Règles de correspondance

Les règles de mise en correspondance permettent d'abord d'identifier les correspondances entre les modèles par point de vue, puis de créer les relations de correspondance entre les éléments. D'une manière plus formelle, une règle de correspondance est une fonction à plusieurs variables dont les éléments des ensembles de départ sont définis dans le métamodèle UML et les éléments de l'ensemble d'arrivée sont définis dans le métamodèle de correspondance (MMC) décrit dans la section précédente,.

Si on considère le cas où l'on a deux modèles par point de vue, une règle de correspondance est définie comme suit :

Rc : UML* UML → MMC
(Element, Element) |→ CorrespondenceRelationship

Dans la suite, la présentation des règles est faite selon le schéma suivant : nom de la règle, définition de la correspondance qu'elle produit, description informelle de son comportement, et exemple d'application. Nous décrivons ci-dessous un sous-ensemble représentatif des règles de correspondance, conformément à la classification des relations données dans le métamodèle de la figure 8.

4.2.1.1. Règle d'équivalence des associations

Nom de la règle : AssociationEquivalenceRule

Définition de l'équivalence d'associations :
Deux Associations sont équivalentes si elles vérifient les conditions suivantes:

1. Elles ont le même nom,
2. Elles ont des extrémités d'associations équivalentes.

Deux extrémités d'associations sont équivalentes si elles vérifient les conditions suivantes :

1. Les classes participantes ont le même nom,
2. Leurs noms, multiplicité, navigabilité sont les mêmes,
3. Elles sont du même type (simple, agrégation, etc.).

Description:

Pour chaque association du modèle par point de vue MPV1, s'il existe une association du modèle par point de vue MPV2, telle que ces deux associations soient équivalentes, alors on crée dans le modèle de correspondances une relation d'équivalence (instance de '*EquivalenceRelationship*') reliant ces deux associations.

Exemple :

Sur les modèles par point de vue Patient et Médecin (figure 3 et 4) les associations entre les classes FicheConsultation et Prescription sont équivalentes.

4.2.1.2. Règle de conformité de classes

Nom de la règle : ClassConformityRule

Définition de la conformité de classes :
Deux classes de modèles par point de vue différents sont conformes si les contraintes suivantes sont vérifiées :

1. Elles ont le même nom,
2. Elles sont toutes deux concrètes ou toutes deux abstraites,
3. Elles ont les mêmes attributs,
4. Elles ont les mêmes opérations (avec la même signature),
5. Elles ont des relations d'associations équivalentes.

Description :

Pour chaque classe du modèle MPV1, s'il existe une classe du modèle MPV2 telle que ces deux classes sont conformes, alors on crée dans le modèle de correspondance une relation de conformité (instance de 'ConformityRelationship') reliant ces deux classes.

Exemple :

Sur les modèles Patient et Médecin (figure 3 et 4), les classes Prescription sont conformes.

4.2.1.3. Règle de similarité de classes

Nom de la règle : ClassSimilarityRule

Définition de la similarité de classes :
Deux classes de modèles par point de vue différents sont similaires si les contraintes suivantes sont vérifiées :

1. Elles ont le même nom,
2. Elles ne sont pas conformes

Description :

Pour chaque classe du modèle MPV1, s'il existe une classe du modèle MPV2 telle que ces deux classes sont similaires, alors on crée dans le modèle de correspondance une relation de similarité (instance de 'SimilarityRelationship') reliant les deux classes.

Exemple:

Sur les modèles Patient et Médecin (figure 3 et 4), les classes DossierMedical sont similaires. Il en est de même pour les classes FicheDeSoin, Patient, et FicheConsultation.

4.2.1.4. Règle d'égalité d'attributs

Nom de la règle : AttributeEqualityRule

Définition de l'égalité d'attributs :
Deux attributs sont égaux si :

1. Ils ont le même nom,
2. Ils ont le même type,
3. Ils ont la même visibilité,
4. Ils sont définis dans deux classes ayant le même nom.

Description:

Pour chaque attribut appartenant à une classe du modèle MPV1, s'il existe un attribut d'une classe du modèle MPV2, telle que ces deux attributs sont égaux, alors on crée dans le modèle de correspondance une relation d'égalité (instance de 'EqualityRelationship') reliant ces deux attributs.

Exemple :

Sur les modèles Patient et Médecin (figure 3 et 4), les attributs *numero* des classes DossierMedical sont égaux.

4.2.1.5. Autres règles de correspondance

Certaines règles de correspondance portent sur plusieurs types d'éléments, comme les opérations, les relations de généralisation, etc. Pour des raisons de place, nous ne les décrivons pas ici.

4.2.2. Règles de composition

Une fois le modèle de correspondances produit, la deuxième étape du processus consiste à appliquer des règles de composition. La stratégie de composition dépend de la nature des relations entre les éléments. Par exemple la stratégie de composition de deux classes reliées par une relation de similarité, est différente de celle adoptée lorsque les classes sont conformes. Pour cela nous avons identifié un ensemble de règles de composition. Nous présentons dans la suite quelques unes de ces règles en illustrant leurs applications par des exemples. D'une manière générale une règle de composition est une fonction dont les éléments de l'ensemble de départ sont définis par le métamodèle de correspondance (MMC), et les éléments de l'ensemble d'arrivée sont définis par le métamodèle VUML :

Rcp : MCC→ VUML
(CorrespondenceRelationship) |→ VumlElement

4.2.2.1. Règle de Composition de classes similaires

Nom de la règle : SimilarityCompositionRule

Description :

Pour chaque relation de similarité du modèle de correspondance reliant deux classes, on crée dans le modèle VUML une classe stéréotypée par « *base* ». Autrement dit, on produit une classe multivue en commençant par sa base.

La classe engendrée a les propriétés suivantes :

- Elle a le même nom que les classes similaires.
- Ses attributs et ses opérations sont les attributs et les opérations respectivement reliés par des liens d'égalité et d'équivalence.
- Les relations d'associations et de généralisation équivalentes des deux classes deviennent des relations de la base.

Exemple :

Soit la relation de similarité établie pour les classes DossierMedical (cf. 4.2.1.3). On crée dans le modèle VUML une classe stéréotypée « base » ayant les propriétés communes aux deux points de vue (cf. figure 5).

4.2.2.2. Règle de création de classes *vues*

Nom de la règle : SimilarityEndCompositionRule

Description :

Pour chaque extrémité d'une relation de similarité du modèle de correspondances reliant deux classes, on crée dans le modèle VUML une classe stéréotypée par «*view* » si elle est concrète, «*abstractView* » sinon. Cette classe est reliée par une relation de dépendance stéréotypée par « *viewExtension* » à la classe *base* générée par l'application de la règle précédente. Autrement dit, on produit des vues correspondant aux classes similaires des modèles par point de vue sources. Les caractéristiques (attributs, opérations, associations) de la classe vue créée sont les propriétés de la classe correspondante qui ne sont reliées par aucune relation de correspondance.

Exemple:

Considérons encore la relation de similarité établie pour les classes DossierMedical. On crée dans le modèle VUML (cf. figure 5) :

- La classe 'PatientDossierMedical' stéréotypée par « view» correspondant à la classe DossierMedical du point de vue Patient avec les propriétés définies par cette règle.

- La classe 'MedecinDossierMedical' stéréotypée par « view» correspondant à la classe DossierMedical du point de vue Médecin avec les propriétés définies de même.

4.2.2.3. Règle de composition de classes conformes

Nom de la règle : ConformityCompositionRule

Description :

Pour chaque relation de conformité du modèle de correspondances, on crée dans le modèle VUML une classe VUML stéréotypée par «*merged* » (avec les propriétés des classes reliées par la relation de conformité). Autrement dit, la composition consiste ici à définir dans le modèle VUML cible une classe identique aux deux classes conformes des modèles sources.

Exemple:

Soit la relation de conformité établie pour les classes Prescription (section 4.2.1.2) des modèles Patient et Médecin. Dans le modèle VUML (cf figure 5), une classe Prescription identique stéréotypée par « merged » a été créée.

4.2.2.4. Règle de composition d'attributs

Nom de la règle : EqualityCompositionRule

Description :

Pour chaque relation d'égalité du modèle de correspondance reliant deux attributs, on crée un attribut dans le modèle VUML (mêmes nom, type, et visibilité). L'attribut créé est défini dans la classe générée par l'application de règles de composition des classes selon que les classes dont il est issu étaient similaires ou conformes.

Exemple:

Soit la relation d'égalité établie pour les attributs *numéro* des classes DossierMedical (section 4.2.1.4). Le modèle VUML de la figure 5 comporte un attribut *numéro* dans la classe DossierMedical stéréotypée par «base ».

4.2.3. Règles de translation

A l'issue de l'étape de mise en correspondance, les éléments des modèles par point de vue sont divisés en deux catégories : les éléments qui ont été stockés dans le modèle de correspondance en tant qu'extrémités d'une relation, et les éléments qui n'ont pas été identifiés par les règles de correspondance. Les éléments de cette catégorie sont transformés dans le modèle VUML par des règles de translation. La stratégie de translation des éléments varie en fonction de plusieurs paramètres. Ainsi,

une classe sera transformée différemment selon qu'elle est une sous-classe d'une classe devenue multivue, ou qu'elle est une classe simplement impliquée dans une association. Nous présentons ci-dessous quelques règles de translation en discutant la stratégie appliquée. D'une manière plus formelle, une règle de translation est une fonction dont les éléments de l'ensemble de départ sont définis par le métamodèle d'entrée UML, et les éléments de l'ensemble d'arrivée sont définis par le métamodèle VUML :

Rt : UML→ VUML
(Element) |→ VumlElement

4.2.3.1. Règle de translation de classes en sous-vues

Nom de la règle : Class2SubViewTranslationRule

Description :

Pour chaque classe d'un modèle par point de vue qui spécialise une classe reliée avec une autre par une relation de similarité, on crée une classe dans le modèle VUML avec les caractéristiques suivantes :

- Le nom de la classe créée est calculé en concaténant le nom de la classe source et le nom du point de vue auquel elle appartient,
- Elle a les mêmes propriétés (attributs, opérations) que la classe source,
- Elle est stéréotypée par « view ».

Exemple:

Soit la classe FichePayement du point de vue Patient (figure 4). En appliquant la règle ci-dessus, cette classe sera transformée en une classe PatientFichePayement stéréotypée par « view ».

4.2.3.2. Règle de translation de classe à l'identique

Nom de la règle : Class2ClassRule

Description :

Pour chaque classe du modèle par point de vue qui ne figure pas dans le modèle de correspondances, on crée dans le modèle VUML une classe de même nom et de mêmes propriétés. Les entités qui lui seront associées dans le modèle VUML sont les entités obtenues par composition/transformation des entités qui lui étaient associées dans le modèle par point de vue.

Exemple :

Soit la classe OperationDiagnostic du point de vue Médecin (figure 3) ; en appliquant cette règle, elle sera transformée en une classe de mêmes propriétés dans le modèle VUML (Figure 5).

5. Mise en œuvre de l'approche

Pour mettre en œuvre et valider notre approche, il nous fallait un langage de transformation à base de règles. Nous souhaitions privilégier l'approche déclarative pour permettre au développeur de définir sa transformation à un niveau d'abstraction relativement élevé, sans se préoccuper de l'exécution finale. A ce stade il ne tient pas compte des contraintes relatives à l'implantation, notamment l'ordre d'exécution des règles. La gestion de l'ordre d'exécution des règles peut être déléguée à un moteur de règles, mais il est souvent nécessaire d'assister ce moteur en introduisant du code impératif en fonction de la complexité des règles.

Le choix du langage s'est porté sur ATL (Jouault et al., 2005) qui est un langage de transformation hybride permettant donc de combiner les approches déclaratives et impératives. Pour implémenter les stratégies de comparaison des éléments de modèles, nous avons utilisé la notion de 'helpers', qui sont l'équivalent de méthodes globales dans les langages de programmation.

La première étape de la réalisation a consisté à traduire les métamodèles mis en jeu tout au long du processus de composition, i.e le métamodèle d'entrée (UML), le métamodèle de correspondance, et le métamodèle de sortie VUML, sous un format textuel en utilisant le langage KM3 (Jouault et al., 2006), langage de spécification de métamodèles conformes au MOF (OMG 2002).

L'utilisation de ce langage facilite l'édition et la conception de métamodèles dans un format textuel. L'intégration de ce nouveau format textuel est assurée par un certain nombre d'injecteurs et d'extracteurs qui permettent d'obtenir un modèle Ecore (Budinsky et al. 2004). Nous explicitons par la suite quelques règles de transformations développées en grande partie en utilisant les caractéristiques déclaratives du langage ATL.

5.1. *Coda*ge des *Règles de correspondance*

Comme nous l'avons expliqué dans le processus de composition de modèles (cf. section 3), la première étape du processus est une étape d'identification des correspondances destinées à être stockées dans le modèle de correspondance.

Le code présenté dans la figure 9 donne un exemple de règles de correspondance codées sous formes de règles déclaratives ATL.

```
abstract rule ModelElementMatchingRule {
                from e1 : UML!ModelElement,e2 : UML!ModelElement(
                        e1.name = e2.name and
                        e1.isLeft() and
                        e2.isRight()
                )
                to r : MMC!CorrespondenceRelationship(
                        name <-e1.name + '_' + e2.name,
                        model <- thisModule.aModel
                ),
                element1 : MMC!MergeableElement(
                        name <- e1.name,
                        model <- thisModule.aModel,
                        ref <- e1.__xmiID__
                 ),
                 end1 : MMC!CorrespondenceRelationshipEnd(
                        element <-element1,
                        relation <-r
                ),
                element2 : MMC!MergeableElement(
                        name <- e2.name,
                        model <- thisModule.aModel,
                        ref <- e2.__xmiID__
                 ),
                 end2 : MMC!CorrespondenceRelationshipEnd(
                        element <-element2,
                        relation <-r
                )

}

rule ClassSimilarityRule extends ModelElementMatchingRule{
                from e1 : UML!Class, e2 : UML!Class(
                        not e1.match(e2))
               to  r : MMC!SimilarityRelationship
    }

rule ClassConformityRule extends ModelElementMatchingRule{
                from e1 : UML!Class, e2 : UML!Class(
                     e1.match(e2))
                to r : MMC!ConformityRelationship
       }
```

Figure 9. *Extrait de règles de correspondance*

La règle `ModelElementMatchingRule` est déclarée comme une règle abstraite, ce qui signifie qu'elle ne peut pas être appliquée directement et est destinée à être étendue par héritage (voir par exemple les règles concrètes *ClassSimilarityRule* et *ClassConformityRule*). Prenons l'exemple de la règle *ClassConformityRule* ; elle permet de créer une relation de conformité entre deux classes UML figurant dans deux modèles par point de vue. Elle appelle le helper *match* qui implante la conformité entre deux classes UML. La figure 10 ci-dessous en présente un extrait.

```
helper context UML!Class def : match(p : UML!Class) : Boolean =
        self.isAbstract = p.isAbstract and
        self.ownedAttribute->size()= p.ownedAttribute->size()and
        self.ownedAttribute->iterate(i ; attr : UML!Property =
       self.ownedAttribute |thisModule.checkAttributes(p,i))and
        self.ownedOperation->size() = p.ownedOperation->size()and
        self.ownedOperation->iterate(i ; op :
       UML!Operation=self.ownedOperation |
        thisModule.checkOperations(p,i))and
        self.association->iterate(ae ; assoc : UML!AssociationEnd =
       self.association |thisModule.checkAssociations(p,ae));
```

Figure 10 *Définition de la conformité entre deux classes*

5.2. *Coda*ge des *Règles de composition et de translation*

Nous présentons sur les figures 11 et 12 quelques règles de composition et de translation. Nous rappelons que les règles de composition permettent de traduire les éléments du modèle de correspondance en éléments du modèle VUML, les règles de translation servant à retranscrire les éléments d'un modèle par point de vue n'ayant pas de correspondant dans le modèle opposé.

```
Rule SimilarityEndCompositionRule{

        from re : MMC!CorrespondenceRelationshipEnd(
                re.relation.oclIsTypeOf(MMC!SimilarityRelationship)
        )
        to v : VUML! View(
                name <- re.element.namespace + re.element.name,
                isAbstract <- re.element.isAbstract,
                isPublic <- re.element.isPublic,
                stereotype <- let st : VUML!Stereotype =
Sequence{VUML!Stereotype.allInstances()}->asSequence()
                 in
                        if not re.element.isAbstract
                         then   st.append(thisModule.view)->
                                 asSequence()
                         else st.append(thisModule.abstractView)->
                                asSequence()
                         endif
        )
do
{
                thisModule.ViewExtensionRule(v);
  }
}

lazy rule ViewExtensionRule{
        from re : MMC!CorrespondenceRelationshipEnd
        to ve : VUML!ViewExtension(
                client <-Sequence{VUML!View.allInstances().
```

```
                    append(thisModule.resolveTemp(re,'v'))->
                     asSequence()},
               supplier <-Sequence{VUML!Base.allInstances().
                    append(thisModule.resolveTemp(re.relation,'b'))->
                     asSequence()},
       stereotype <-Sequence{VUML!Stereotype.allInstances().
  append(thisModule.viewExtension)->asSequence()}
       )
}
```

Figure 11. *Extrait des règles de composition*

La règle *SimilarityEndCompositionRule* transforme toute extrémité de relation de correspondance instance de *SimilarityRelationshipEnd* en une classe stéréotypée par '*view*'. La partie impérative de la règle comprend un appel à la règle *ViewExtensionRule*. Cette règle peut être appelable explicitement par d'autres règles tout en restant dans un contexte déclaratif (lazy rule). L'opérateur *resolveTemp* d'ATL est nécessaire car plusieurs classes vues et relations stéréotypées par *viewExtension* sont créées pour la même classe stéréotypée par *base*.

```
abstract rule Element2VUMLElement{
       from s : UML!ModelElement(
                     s.isLeft() or
                     s.isRight() and
                     not s.isInCorrespendenceModel
              )
       to t : VUML!ModelElement(
              name <- s.name,
              namespace <- thisModule.amodel
       )
}
```

Figure 12. *Définition de la stratégie par défaut de translation des éléments*

La règle abstraite `Element2VUMLElement` définit le comportement par défaut d'une règle de translation. La condition de garde de la règle vérifie que l'élément à translater est dans un des modèles par point de vue, et n'est pas dans le modèle de correspondance. La stratégie de translation des éléments peut être surchargée en redéfinissant cette règle pour chaque type d'élément ; par exemple la classe *FichePayement* du point de vue patient (figure 4) ne peut être translatée par la stratégie définie par défaut, car elle hérite d'une classe (FicheDeSoin) qui donnera lieu à une classe multivue. Pour résumer, les contraintes relatives aux stratégies de translation et de composition des éléments sont essentiellement dépendantes de la sémantique du langage VUML.

6. Travaux relatifs

La problématique de composition de modèles n'est pas nouvelle et a été étudiée dans plusieurs domaines de l'informatique tels que l'ingénierie des exigences (Sabetzadeh et al., 2005) (Chitchyan et al., 2007), la conception par aspects (Baniassad et al., 2004), (Reddy et al., 2006), le développement par sujets (Clarke, 2002), (Ossher el al., 1996), l'intégration de schémas de base de données (Spaccapietra et al., 1994), la fusion de versions de programmes (Mens, 2002), la composition des préoccupations dans les architecture logicielles (Barais, 2005) , etc. Nous nous intéressons dans la suite de cette section aux approches qui nous semblent les plus proches de notre problématique.

Dans la démarche VUML, la première phase d'analyse des besoins fournit une modélisation du système selon différents points de vue (c'est-à-dire selon la vision des différents acteurs du système). En ce sens, notre façon de considérer les vues est très proche de ce qui se fait dans le domaine de ingénierie des exigences. L'un des apports fondamentaux de la prise en compte des points de vue par l'Ingénierie des exigences concerne la richesse apportée par la possibilité de prendre en compte des préoccupations diverses, transverses, parfois même contradictoires. Les approches de Finkelstein et al. (Nuseibeh et al., 2003), Easterbrook (Easterbrook et al., 1995) ou encore Jakson (Jackson, 2001) considèrent des points de vue pouvant être exprimés dans des formalismes différents, qui sont ensuite reliés via un système de mise en cohérence, afin d'obtenir un système global cohérent. Pour notre part, nous partons du principe que le formalisme UML peut être un formalisme unificateur pouvant servir pour modéliser les différents points de vue ; nous composons les modèles par points de vue afin d'obtenir un modèle global unique partageable et non une organisation de modèles reliés.

Les problèmes de composition de modèles auxquels nous nous intéressons dans ce papier sont relativement comparables à ceux qui ont traités de l'intégration des schémas dans le domaine des bases de données (Batini et al., 1986), (Navathe et al., 1986), même si le contexte est différent. La méthodologie proposée par Spaccapietra et al (Spaccapietra et al., 1994) repose sur la définition formelle de correspondances entre les vues par le biais d'assertions. Cette technique réutilise la notion de RWS (Real-Word State) d'un objet type proposé par (Larson et al., 1989). Ces assertions sont utilisées par la suite par un algorithme d'intégration, qui, moyennant des règles appropriées, produit le schéma global et l'ensemble des 'mappings' entre les schémas initiaux et le schéma final. Bien que cette approche permette une intégration automatique des vues avec (ou sans) conflits structurels entre les objets, une étape manuelle de définition des différentes correspondances est nécessaire.

La composition des modèles de conception dans les approches orientées aspect a fait l'objet de nombreux travaux. Dans (Reddy et al. 2006), la technique proposée permet de gérer la composition de plusieurs modèles d'aspects modélisant les préoccupations transverses de l'application (persistance, sécurité, etc.), avec un modèle primaire qui représente le cœur fonctionnel de l'application. Cette approche

utilise un algorithme de composition et un ensemble d'actions élémentaires appelées directives de composition. Ces directives permettent d'effectuer des opérations, soit sur les éléments des modèles comme la création/suppression d'un élément, la modification de la valeur d'une propriété, etc., soit sur les modèles d'aspects, en spécifiant par exemple l'ordre dans lequel deux modèles d'aspects sont composés. Un métamodèle de composition est également proposé. Il étend celui d'UML en ajoutant les spécifications des comportements de composition et inclut des méta-opérations implémentant la comparaison entre les éléments en se basant sur leurs signatures. L'utilisation de ces directives pour résoudre les conflits pouvant apparaître lors de la phase de composition s'avère intéressante car elle permet de réduire en partie une activité qui était souvent à la charge du concepteur. Cependant, cette approche est difficilement compatible avec un système à base de règles déclaratives tel que celui que nous avons développé.

Dans la continuité du travail précédent, l'approche de composition de modèles présentée récemment dans (Fleurey et al., 2007) offre un Framwork générique indépendant d'un langage de modélisation. Dans cette approche, les auteurs proposent un métamodèle générique décrivant le comportement d'un opérateur de composition basé sur la notion de directives de composition issues de (Reddy et al., 2006). Ces directives sont spécifiées par un langage indépendant du domaine ; la syntaxe abstraite est définie par un métamodèle générique et la syntaxe concrète est supportée par le langage Kermeta (Muller et al., 2005). Le mécanisme de spécialisation de ce Framework consiste dans une première étape à définir les stratégies de composition (égalité des éléments, définitions des signatures) propres à un langage de modélisation. Ces stratégies sont stockées dans un modèle séparé. La deuxième étape consiste à tisser ce modèle de stratégies avec le métamodèle du langage de modélisation via un mécanisme de composition défini par le langage Kermeta. Cette approche est intéressante car elle favorise la réutilisation du savoir-faire métier de la composition de modèles, et permet de concevoir des opérateurs de composition spécifiques à chaque langage de modélisation, tout en gardant un noyau générique commun. Cependant, comme nous l'avons dit ci-dessus, l'utilisation des directives de composition n'est pas naturelle pour notre approche principalement déclarative.

Le Framework Transat (Barais, 2005) propose la notion de 'tissage de plans' pour la composition des préoccupations, au sein d'une architecture logicielle définie selon le modèle de composants SafArchie (Barais et al., 2005). Le processus de composition est itératif et incrémental. Le résultat de l'étape d'intégration d'une préoccupation technique au sein d'une architecture logicielle constitue une nouvelle architecture capable d'intégrer de nouvelles préoccupations (Barais et al., 2007). Par analogie avec les concepts définis dans le domaine de la programmation par aspects, la composition dans Transat est vue comme une opération de tissage asymétrique d'une architecture logicielle existante (appelée aussi plan de base) avec le plan de la préoccupation à intégrer. Les modifications à apporter au plan de base sont spécifiées à l'aide des règles de transformation qui s'appuient sur un ensemble de

primitives que l'on peut diviser en deux catégories : primitives d'introduction (modifications intra-composants) d'une part et primitives de reconfiguration (modifications inter-composants) d'autre part. Cette approche offre une certaine liberté aux utilisateurs pour paramétrer la composition. Cependant la technique de transformation est fortement dépendante du modèle de composant SafArchie.

Le langage EML (Epsilon Merging Language) proposé par (Dimitirios et al., 2006) est un langage de fusion de modèles basé sur des règles, qui offre un cadre générique de fusion de modèles. EML fait partie de la plateforme Epsilon (Epsilon, 2006) supportée par l'environnement Eclipse, et intègre plusieurs langages spécifiques de domaines. Ainsi plusieurs tâches de manipulation de modèles sont supportées par cette plateforme, comme la comparaison, la fusion, la transformation, et la validation. Cette approche est relativement récente et prometteuse car à notre connaissance ce langage fait partie des rares langages dédiés qui définissent des opérateurs de fusion dans la syntaxe concrète du langage. Cependant le caractère impératif de ce langage s'est avéré peu adapté pour implémenter les règles de transformation décrites dans notre approche.

Pour terminer, nous situons notre approche par rapport au schéma canonique de composition de modèles présenté dans (Bézivin et al., 2006). Ce travail s'est basé sur la comparaison de trois approches de composition de modèles : GGT (Glue Generator Tool) (Bouzitouna et al., 2005), EML (Epsilon Merging Language) (Dimitrios et al., 2006), et AMW (Atlas Model Weaver) (Didonet et al., 2005). Dans cette étude, les auteurs définissent des fondements relatifs au domaine de la composition de modèles. Ces fondements incluent deux axes : le premier axe concerne la définition d'un vocabulaire et de définitions communes ; le deuxième axe définit un ensemble d'exigences que doit satisfaire au minimum toute approche de composition de modèles, à savoir : (i) la possibilité d'identifier les éléments en correspondance dans les différents modèles. Cette exigence est vérifiée dans notre approche grâce aux règles de correspondance ; (ii) la possibilité de définir comment les éléments correspondants sont fusionnés et composés dans le modèle destination. Nous utilisons la notion de règles de composition pour répondre à cette exigence ; (iii) la possibilité de transformer les éléments qui n'ont pas de correspondant. Cette catégorie d'éléments est prise en charge dans notre approche, par les règles de translation (cf section 4.2.3) ; (iv) la possibilité de réutiliser des règles, des liens, des stratégies, etc. Nous prenons en compte dans un premier temps ce type de besoin par l'extension du métamodèle de correspondance et par l'utilisation de l'héritage entre les règles.

7. Discussion

L'approche présentée dans cet article présente naturellement certaines limites et pose des questions qui sont abordées ci-dessous.

Composition de diagrammes dynamiques

Dans cette étude, nous avons traité la composition de diagrammes de classes, qui est le noyau essentiel de la composition au sein de VUML. Cependant, pour être complet, il faut aussi traiter la composition des diagrammes dynamiques d'UML tels que les diagrammes de séquence, les diagrammes états-transitions, les diagrammes d'activités. Ce travail fait l'objet d'une thèse en cours dans notre équipe.

Résolution des conflits

Nous avons identifié les conflits dus aux différences de désignation des éléments dans les modèles par point de vue à composer (homonymie, synonymie), mais le traitement de ces conflits, bien qu'important, sort du cadre de l'article. Ce type de conflit a été étudié par la communauté des bases de données (Navathe et al., 1986), (Spaccapietra et al., 1994), et a fait l'objet plus récemment de travaux dans le domaine des ontologies (Dhamank et al., 2004) (Aumueller et al., 2005) en rejoignant des préoccupations d'ingénierie des exigences (Nusebeih et al. 2003).

Un autre type de conflit concerne les aspects structurels, comme par exemple un conflit entre un héritage et une relation de clientélisme dans deux modèles structurels par point de vue. Nous avons étudié cette problématique dans un travail précédent (Kriouile, 1995) en proposant une approche basée sur un ensemble d'heuristiques qui permettent de guider le concepteur dans le processus d'harmonisation des modèles par point de vue.

Prise en compte de la sémantique dans les modèles en entrée

La stratégie de comparaison des éléments est définie par les règles de correspondance, mais ces règles s'appuient essentiellement à ce jour sur les propriétés syntaxiques définies dans le métamodèle d'entrée. Si nous prenons l'exemple de deux opérations dans deux modèles par point de vue apparaissant avec la même signature (nom, type, paramètres), elles seront considérées comme équivalentes dans la version actuelle de notre approche, ce qui bien sûr n'est pas toujours le cas. Pour remédier à ce problème, il faut soit prévoir une étape de réconciliation entre les concepteurs, soit renforcer la sémantique associée au métamodèle d'entrée, de façon à pouvoir implanter des stratégies de comparaison plus fines qui tiennent compte des comportements décrits par les méthodes.

Langage d'implantation des règles

Pour valider notre travail, nous avons choisi d'implanter les règles en ATL, pour bénéficier d'une approche semi-déclarative. Si ce choix s'est avéré judicieux sur notre étude de cas, il se peut qu'il présente des limites lors du passage à l'échelle avec des modèles de plus grande taille et un plus grand nombre de règles. Il nous faut donc poursuivre l'expérimentation en complétant la base de règles existantes et en composant de plus gros modèles. Par ailleurs, nous souhaitons in fine développer une approche indépendante d'un langage à base de règles spécifique et considérons d'ores et déjà d'autres langages cibles (Kermeta, …).

Exploitation du modèle fusionné VUML

Le modèle VUML obtenu à l'issue de la fusion est un modèle conforme au profil VUML, exploitable de plusieurs manières. Si le processus de conception par point de vue n'est pas terminé (point de vue supplémentaire à intégrer par exemple), un tel modèle peut être fusionné à son tour avec un modèle par point de vue développé séparément. Pour cela, on peut appliquer notre approche en remplaçant le métamodèle d'entrée UML par le métamodèle VUML. Lorsque le processus de conception est terminé, le modèle VUML peut être dérivé en classes d'un langage à objet cible. Nous avons ainsi développé un patron de génération de code générique, et réalisé une première instanciation de ce patron en Java (Crégut et al., 2005).

8. Conclusion

Le travail exposé dans cet article s'inscrit dans le cadre de travaux de recherche visant à enrichir le profil VUML développé au sein de notre équipe. L'un des axes prioritaires de notre travail est la définition d'une approche de composition (fusion) de modèles UML ou VUML permettant de rendre efficiente la multi-modélisation. Pour atteindre cet objectif, nous avons situé notre approche dans le cadre de l'Ingénierie Dirigée par les Modèles.

Après avoir présenté la démarche de développement par point de vue d'un système logiciel, nous avons décrit un processus de composition de modèles UML partiels en un modèle VUML multivue unique, en considérant cette composition comme une transformation de n modèles vers un modèle. Le processus est guidé par un ensemble de règles catégorisées en règles de correspondance, de composition, et de translation. Le processus de composition proposé s'appuie en premier lieu sur l'identification de différentes relations de correspondance entre les modèles. Ces relations sont stockées dans un modèle séparé, qui est exploité ensuite dans la phase de composition.

Pour valider notre travail, nous avons travaillé sur une étude de cas issue de besoins réels, en nous restreignant pour simplifier à deux modèles par points de vue (diagrammes de classes) conformes au métamodèle UML2.0 (OMG. 2003b). Nous avons appliqué le processus proposé, en mettant donc en œuvre les règles présentées dans cet article.

Le choix du langage d'implantation des règles s'est porté sur le langage ATL, dans un objectif d'expérimentation. Ce choix nous a permis de coder la plupart des règles de composition sous une forme déclarative. L'ordre d'exécution de ce type de règles est implicitement spécifié par le développeur, puis pris en charge par un algorithme de résolution implanté dans le moteur d'exécution de règles ATL. Les exigences relatives aux contraintes d'implantation telles que la sérialisation de modèles, les scénarii d'exécution et le débogage, sont pris en compte par l'environnement de développement ADT (ATL Development Tooling) (Allilaire et al., 2004) du langage ATL.

A ce jour, nous pouvons générer le modèle de correspondance. Nous travaillons actuellement sur l'affinage de l'implémentation des règles de composition/translation. En ce qui concerne la vérification de modèles VUML en sortie de composition, nous disposons d'une telle fonctionnalité dans le profil VUML développé dans (Nassar, 2005).

Perspectives du travail

Comme cela a été discuté dans la section 7, il reste un certain nombre de travaux de recherche et de réalisation pour compléter le champ de notre étude et expérimenter le passage à l'échelle.

Après validation de l'approche sur deux modèles UML statiques (diagrammes de classes), nous travaillons à sa généralisation en considérant n modèles UML en entrée. Une autre extension naturelle de ce travail serait de prendre en entrée des modèles VUML, résultant éventuellement d'un processus de composition de modèles UML et/ou VUML. L'approche développée dans cet article reste adaptée à cette configuration, mais il faudra enrichir les règles de transformation pour prendre en compte le métamodèle VUML en entrée. En effet, considérant en entrée deux modèles VUML modélisant deux systèmes selon différents points de vue, l'obtention d'un modèle VUML global s'obtiendrait par la définition de nouvelles stratégies de comparaison et de composition entre les éléments des deux modèles VUML.

Nous envisageons aussi de travailler sur la résolution des conflits d'ordre sémantique (homonymie, synonymie) inhérents à la composition de modèles développés séparément. Cela nécessitera la définition de stratégies de comparaison des éléments s'appuyant sur des propriétés de nature sémantique, l'utilisation d'heuristiques issues de l'expérience des concepteurs, l'application de techniques basées sur les ontologies, …

Nous souhaitons enfin étendre la portée de cette étude afin de couvrir d'autres types de modèles à composer, notamment les diagrammes dynamiques d'UML. Un travail de recherche est en cours sur les notions de machine à états et diagrammes d'activité multivue au sein du profil VUML, préalable au traitement de la fusion de modèles dynamiques.

9. Bibliographie

Alanen, M. and Porres, I.. « Difference and Union of Models ». *P.Steven et al (Eds) : UML 2003*, LNCS 2863, pp 2-17, 2003.

Allilaire, F., Idrissi, T. « ADT: Eclipse Development Tools for ATL ». *EWMDA-2*, Canterbury, England, 2004.

Aumueller, D, Do, H H, Massmann, S, Rahm, E. Schema and ontology matching with COMA++. In proc. of SIGMOD 2005. pp 906-908.

Anwar A, Nassar M., Coulette B.,Ebersold S, Kriouile A., « 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 A., Ebersold S., Coulette B., Nassar M., Kriouile A., « 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)*, Toulouse, France, 2007, Hermès Science, pp 41-58.

Baniassad, E., Clarke, S.: « Theme: An approach for aspect-oriented analysis and design », *In Proc. of the International Conference on Software Engineering*. (2004), pp 158 -167

Barais O., Construire et Maîtriser l'évolution d'une architecture logicielle à base de ,composants, Thèse de doctorat, Laboratoire d'Informatique Fondamentale de Lille, Université de Lille, Lille, France, Novembre, 2005.

Barais O., Duchien L., « SafArchie Studio: An ArgoUML extension to build Safe Architectures », Architecture Description Languages, Springer, p. 85-100, 2005.

Barais O, Philippe Lahire, Alexis Muller, Noêl Plouzeau, Gilles Vanwormhoudt. « *Evaluation de l'apport des aspects, des sujets et des vues pour la composition et la réutilisation des modèles* », *Revue RSTI-L'Objet*, vol.13 – n° 2-3/2007. pp. 177-212.

Batini, C., Lenzerini, M. and Navathe, S. B.. « A Comparative Analysis of Methodologies for Database schema Integration ». *ACM Computing Surveys*, Vol. 18, No. 4, Dec. 1986.

Bézivin, J, Bouzitouna, S, Del Fabro, MD, Gervais, M, Jouault, F, Kolovos, D, Kurtev, I, and Paige, R :« A Canonical Scheme for Model Composition ». *In proc. of ECMDA-FA*, 2006, LNCS 4066, Springer-Verlag, pages 346-360. 2006.

Bouzitouna S, M. P. Gervais and X. Blanc, « Model Reuse in MDA », *Proceedings of the International Conference on Software Engineering Research and Practice (SERP'05)*, Las Vegas, USA, June 2005.

Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. Eclipse Modeling Framework. Addison Wesley Professional. ISBN: 0131425420, 2004.

Chitchyan R., Rashid A., Rayson P., Waters R.. « Semantics-based Composition for Aspect-Oriented Requirements Engineering ». *AOSD 07*, pp 36- 48. March 12-16, 2007, Vancouver Canada.

Clarke, S.: « Extending Standard UML with Model Composition Semantics ». *Science of Computer Programming*, 44 (2002) 71.100.

Crégut X., Ebersold S., Nassar M., Coulette B., « Un patron de génération de code pour le profil VUML », *LMO-OCM'2005*, Berne, Suisse, 9-11 mars 2005. pp. 5-11.

Czarnecki K. and Helsen S. « Classification of Model Transformation Approaches ». In 2nd *OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. Anaheim, CA, USA. October 2003.

Dhamanka, R, Lee Y, Doan, A, Halevy, A, Domingos P. iMAP: Discovering Complex Semantic Matches between Database Schemas. In proc. of SIGMOD 2004, pp 383-394.

Didonet Del Fabro, M, Bézivin, J, Jouault, F, Breton, E, and Gueltas, G : « AMW: a generic model weaver ». *Actes IDM'05*. Paris, France, p. 105-114, juin 2005.

Dimitrios S. Kolovos, Richard F. Paige and Fiona A.C. Polack. «Merging Models with the Epsilon Merging Language (EML) ». *In Proc. ACM/IEEE 9th International Conference on Models/UML 2006)*, Genova, Italy, October 2006.

Easterbrook, S. M., Nuseibeh, B. "Managing inconsistencies in an evolving specification". In Second IEEE International symposium on Requirements Engineering , pp 48-55 , York, England. IEEE Computer Society, March 1995.

Finkelstein A., Kramer J., Goedicke M., "Viewpoint Oriented Software Development", *Proceedings of Software Engineering and Applications Conference*, Toulouse, December 1990, p. 337-351.

Fleurey F., "Kompose : a generic model composition tool. 2007". http://www.kermeta.org/kompose/.

Fleurey F, Benoit Baudrey, Robert France and Sudipto Ghosh. "A Generic Approach For Model Composition." In Proceedings of the Aspect Oriented Modeling. Workshop at Models 2007. Nashville USA 2007.

Jackson, M.: . Problem Frames. Addison-Wesley, (2001).

Jouault, F, Bézivin, J. « KM3: a DSL for Metamodel Specification ». *In proc. of 8th FMOODS*, LNCS 4037, Bologna, Italy, 2006, pp 171-185.Kolski C., Interfaces homme-machine, Paris, Hermès.

Jouault, F, Ivan Kurtev. «Transforming Models with ATL ». *In Proceedings of the Model Transformations in Practice, Workshop at Models 2005*, Montego Bay, Jamaica 2005.

Klein J., Baudry B., Barais O., Jackson A.. « Introduction du test dans la modélisation par aspects». Actes IDM'2007, Toulouse, France, 2007, Hermès, pp 83-99.

Kriouile A., "VBOOM, une méthode orientée objet d'analyse et de conception par points de vue", thèse d'Etat de l'université Mohammed V de Rabat, 1995

Larson. J. A, S. B. Navathe, and R. Elmasri, « A theory of attribute equivalence in databases with application to schema integration », *IEEE TSE.*, vol. 15, no. 4, Apr. 1989.

Mens, T., « A state-of-the-art survey on software merging », *Transactions on Software Engineering*, vol. 28, no 5, May 2002.

Muller A., Caron O., Carré B., VanWormhoudt G., Bouzitouna S., « Ingénierie multi-modèle : Projection flexible d'assemblages de modèles », *LMO 2007*, pp 167- 182, Hermès, 2007.

Muller P.-A, Fleurey, F. and Jézéquel, J.M. "Weaving executability into object-oriented meta-languages". In Proceedings of MoDELS'05, p. 264 - 278. Montego Bay, Jamaica, October 2005.

Nassar M., Coulette B., Crégut X., Marcaillou S., Kriouile A., « Towards a View based Unified Modeling Language », *Proceedings of 5th International Conference on Enterprise Information Systems ICEIS'03* , Angers, 23-26 April 2003a, pp. 257-265.

Nassar M., Coulette B., Guiochet J., Ebersold S., El Asri B., Crégut X., Kriouile A, « Vers un profil UML pour la conception de composants multivues ». *Revue RSTI-L'Objet*, vol.11 – n°4/2005. pp. 83-113.

Nassar M., « Analyse/conception par points de vue : le profil VUML », Thèse de l'Institut National Polytechnique de Toulouse - Septembre 2005.

Navathe, S. B., Elmasri, R., and Larson, J. « Integrating user views in database design ». *IEEE Computer* . 19,l (Jan.), 50-62. 1986.

Nuseibeh, B., Finkelstein A., and Kramer, J. « ViewPoints : meaningful relationships are difficult». International Conference on Software Engineering (ICSE 2003), Portland, Oregon, 2003.

OMG 2002. OMG/MOF Meta Object Facility (MOF) 1.4. Final Adopted Specification Document. formal/02-04-03, 2002.

OMG 2003a, UML 2 OCL Final Adopted Specification, http://www.omg.org/docs/ptc/03-10-14.pdf.

OMG 2003b, UML 2.0 Superstructure Final Adopted specification, Document - ptc/03-08-02, , http://www.omg.org/docs/ptc/03-08-02.pdf.

Ossher. H, M. Kaplan, A. Katz, W. Harrison, V. Ktl.lSkal, « Specifing Subject-Oriented Composition ». *Theory and Practice of Object Systems*, Volume 2(3), 179-202, 1996.

Reddy Y. R., Ghosh S., France R. B., Straw G., Bieman J. M., McEachen N., Song E., Georg G., «Directives for Composing Aspect-Oriented Design Class Models ». *Transactions of Aspect-Oriented Software Development*, Vol.1, No. 1, LNCS 3880, p75-105, Springer.

Sabetzadeh M and S. Easterbrook. « An Algebraic Framework for Merging Incomplete and Inconsistent Views ». *In 13th IEEE International Requirements Engineering Conference*, September 2005.

Soley et al., « MDA Model Driven Architecture », by Richard Soley and the OMG Staff Strategy Group, Object Management Group White Paper, Draft 3.2 - November 27, 2000.

Spaccapietra. S, Parent C., et Dupont, Y. « View Integration : a step forward in solving structural conflicts », *IEEE Transactions on Data and Knowledge Engineering*, vol. 6, no.2, April 1994.