MajecSTIC 2009
Avignon, France, du 16 au 18 novembre 2009

SPEM4MDE : un métamodèle basé sur SPEM 2 pour la spécification des procédés MDE[1]

Samba Diaw[1] et Rédouane Lbath[1] et Bernard Coulette[1]

1 : Université de Toulouse, Laboratoire IRIT-UTM, 5, allées A. Machado 31058 TOULOUSE CEDEX 9

Contact : diaw@univ-tlse2.fr

Résumé. L'approche MDA (*Model Driven Architecture)* place les modèles au cœur du processus de développement logiciel. Cependant l'utilisation de MDA requiert la définition d'un procédé logiciel qui doit guider les développeurs dans l'élaboration et la génération des modèles. Au moment où des procédés logiciels dirigés par les modèles émergent (MDA, OpenUP for MDD (Model Driven Development), processus de composition, etc.), nous notons une absence de LDP (Langage de Description de Procédés) permettant la description, la réutilisation et l'évolution de ces procédés. Dans ce contexte, nous proposons dans cet article une nouvelle approche pour la spécification des procédés de développement qui prend en compte les notions de base de l'IDM (Ingénierie Dirigée par les Modèles). Cette approche est basée sur un métamodèle générique et flexible dénommé SPEM4MDE qui étend certains concepts de SPEM 2.0 et d'UML 2.0. Il est dédié à la modélisation et à la mise en œuvre des procédés de développement logiciels dirigés par les modèles.

Mots-clés : Ingénierie dirigée par les modèles (IDM), Architectures dirigées par les modèles (MDA), Transformations de modèles, LDP (Langage de Description de Procédés)

Abstract. The MDA (*Model Driven Architecture)* approach puts models on the heart of software process development. However, the use of MDA requires a software process that guides developers in the elaboration and generation of models. While first model-driven software processes (MDA, OpenUP for MDD) have started to appear, an approach for describing, reusing and evolving them is lacking. In this context, this paper presents a new approach for the specification of model-driven development processes. This approach is based on a generic and flexible metamodel called SPEM4MDE (SPEM for MDE) that specializes some SPEM 2 and UML 2 concepts. SPEM4MDE is dedicated to model-driven software processes modelling and enactment.

Keywords: Model Driven Engineering (MDE), Model Driven Architecture (MDA), models transformations, PDL (Processes Description Languages)

[1] Le terme procédé MDE signifie procédé de développement logiciel dans une vision IDM

1. Introduction

L'Ingénierie Dirigée par les Modèles (IDM) est une discipline récente du génie logiciel qui promeut les modèles en entités de première importance dans le développement logiciel [5]. Elle fait l'objet d'un grand intérêt aussi bien de la part des équipes de recherche académiques [4, 22, 23, 24], que des laboratoires industriels [1, 7, 13, 21]. L'IDM est une forme d'ingénierie générative, par laquelle tout ou partie d'une application informatique est générée à partir de modèles. Le processus de développement des systèmes défini en général, comme un ensemble d'activités aussi bien techniques qu'administratives pour développer et maintenir un produit logiciel [25], est alors vu comme une séquence de transformations de modèles partiellement ordonnée, chaque transformation prenant un ou des modèles en entrée et produisant un ou des modèles en sortie, jusqu'à l'obtention d'artéfacts exécutables.

Aujourd'hui, de nombreux travaux portent sur l'élaboration et la représentation des procédés logiciels dans une vision IDM. C'est le cas de *MDA* [15], une application de l'Ingénierie dirigée par les modèles (IDM) qui propose un processus en Y qui sépare le métier et la technique.

D'autres projets tels que DOMINO (DOMaINes et prOcessus) [3] traite du problème de la représentation des processus IDM. *DOMINO* est un projet de l'ANR qui porte sur l'exploitation de l'ingénierie des modèles pour fiabiliser les processus de développement de logiciels. Les activités de DOMINO couvrent les différentes phases d'un processus MDA pour le développement des systèmes temps réel embarqués, à savoir : la prise en compte des exigences ; le processus de modélisation du système logiciel et matériel ; la génération de code ainsi que les phases de vérification, validation & simulation.

Le projet *NEPTUN*E [8,12] traite aussi du domaine de la modélisation des processus à base de modèles et propose une démarche de développement à base de modèles fortement outillée. L'intérêt de l'outillage de la démarche NEPTUNE est de s'assurer que chaque acteur suit le bon déroulement de la démarche à travers un diagnostic établi à différents points de contrôle.

OpenUP for MDD est un processus basé sur UP qui s'inscrit dans le cadre d'une démarche de développement à base de modèles. Il met l'accent sur les notions spécifiques de l'IDM. Son contenu est basé sur l'initiative MDA de l'OMG mais adapte cette approche en fournissant une implémentation plus pratique. Le nouveau standard de l'OMG (Object Management Group), SPEM 2.0 (Software Process Engineering Metamodel) [19] ainsi que d'autres langages de description de procédés (PROMENADE [20], UML4SPM [5], etc.), ne prennent pas en compte dans leurs métamodèles les notions de base de cette récente discipline

C'est dans ce contexte, que nous avons entrepris des travaux de thèse en 2008, portant sur la modélisation et la mise en œuvre de procédés de développement logiciel dirigés par les modèles. Dans cet article, nous proposons une nouvelle approche pour la spécification des processus de développement qui prend en compte les notions de base de l'IDM (modèles, métamodèles et transformations de modèles). Cette approche est basée sur un métamodèle générique et flexible dénommé SPEM4MDE (SPEM for MDE). C'est un métamodèle qui étend certains concepts de SPEM 2.0 et d'UML 2 [17, 18]. Il est dédié à la modélisation et à la mise en œuvre des procédés de développement logiciel dirigés par les modèles.

Cet article est organisé comme suit. Dans la section 2, nous présentons la problématique et les objectifs de nos travaux. Dans la section 3, nous présentons les paquetages qui composent la partie Process Structure de SPEM4MDE. Dans la section 4, nous présentons un extrait du procédé de composition VUML décrit avec les concepts de SPEM4MDE. Nous concluons par la section 5 qui résume les points abordés dans cet article et les perspectives. Par manque de place, la mise en œuvre n'est pas traitée dans cet article.

2. Problématique et objectifs

Avec l'émergence de l'ingénierie dirigée par les modèles (IDM) et des architectures orientées modèles (MDA) [15], la vue du cycle de vie d'un logiciel est en train d'évoluer progressivement vers

une séquence de transformations de modèles. La démarche MDA prône un processus de développement logiciel basé sur la production de modèles, des exigences jusqu'au code. Dans cette démarche, les notions de métamodèles, de modèles et de transformation se trouvent placées au centre du processus de développement. Notre approche vise à explorer la possibilité de prendre en compte la démarche MDA dans les processus de développement tant au niveau de la modélisation des processus qu'au niveau de leur mise en œuvre. Au niveau de la modélisation, l'objectif est de pouvoir spécifier les activités des processus en termes de transformations de modèles en précisant notamment leurs conditions et contextes d'applicabilité, ce qui impliquera la définition d'un formalisme de description de processus incluant des constructions spécifiquement dédiées aux processus MDA. Au niveau de la mise en œuvre des modèles de processus, l'objectif est d'être en mesure de proposer aux développeurs les transformations de modèles appropriées au contexte de développement et de superviser l'activation des outils correspondants. Dans cet article, seul l'aspect modélisation des processus sera présenté. L'aspect mise en œuvre fera l'objet d'une publication ultérieure.

3. SPEM4MDE : un métamodèle pour la spécification des procédés MDE

Le métamodèle SPEM4MDE étend un certain nombre de concepts de SPEM 2.0 et d'UML 2 0. Il est défini comme un modèle conforme à MOF (Meta-Object Facility) [14] et se situe au même niveau que SPEM 2.0 et UML 2.0 dans l'architecture à quatre niveaux de l'OMG. L'objectif principal visé est la description et la mise en œuvre des procédés de développement logiciel intégrant *les notions de base de l'ingénierie dirigée par les modèles* tout en gardant certains acquis de SPEM 2.0 tels que la flexibilité , la séparation des préoccupation et la traçabilité. SPEM4MDE est composé de deux paquetages : *Process Structure* qui permet de décrire statiquement les procédés de développement IDM et *Process Behavior* qui fournit les mécanismes d'exécution du modèle de procédé IDM (simuler l'exécution d'un projet de développement orienté modèle). Pour des raisons de place, nous ne présentons pas dans cet article le paquetage Process Behavior.

3.1. Concepts de base de SPEM4MDE

Notre approche (figure 1 ci-dessous) a pour but de fournir aux acteurs de l'IDM, un langage de description des procédés de développement orienté modèle qui va s'aligner au même niveau que les standards de ce domaine (SPEM 2.0 par exemple) et qui permettra aux concepteurs de procédé de décrire aussi bien une activité classique (non assimilable à une transformation de modèles) qu'une activité orientée modèle. L'idée de base de notre approche est de considérer qu'un processus de développement logiciel est une séquence de transformations de modèles jusqu'à aboutir à un artéfact exécutable : le produit logiciel. Chaque rôle (compétence particulière) apporte sa contribution en participant à la réalisation d'une activité (Work). Cette activité nécessite des outils spécifiques (Tool) que le rôle doit ou peut utiliser pour produire des produits, éventuellement à partir d'autres produits. L'activité requiert des compétences (Qualification) pour sa réalisation. Ces compétences sont fournies par les rôles. L'héritage entre Model et Product permet d'assimiler un modèle à un produit particulier du développement logiciel. La définition de la transformation consiste à préciser les paramètres de la transformation (modèles et métamodèles d'entrée et de sortie et /ou fichier de configuration), les rôles requis et les outils qui seront utilisés.

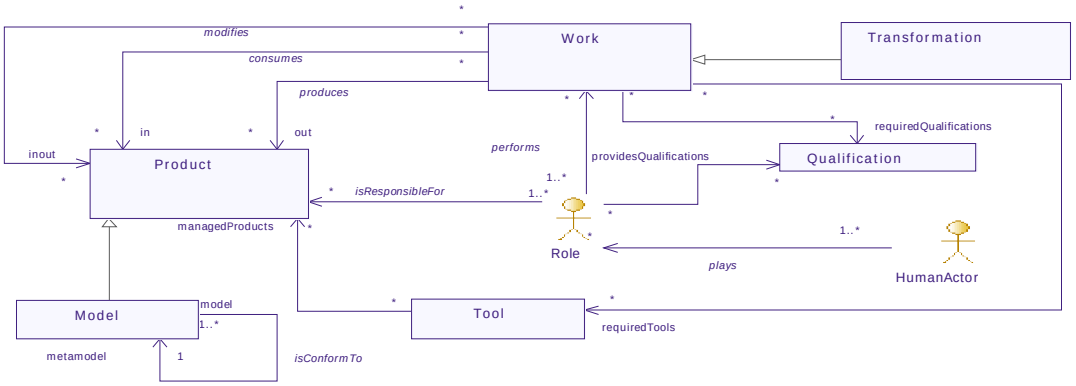

Figure 1. - Concepts de base de SPEM4MDE

3.2. Présentation du noyau de Process Structure

La figure 2 ci-après montre les principaux concepts nécessaires pour la description d'un processus de développement. Nous distinguons deux concepts clés : *ProcessElement* qui décrit tout élément d'un procédé (rôles, produits, activités, outils et qualifications) et *Process Relation* qui décrit les différentes relations entre les éléments d'un procédé (relations entre produits, paramètres d'une activité, responsabilité sur un produit, réalisateur d'une activité et relation de dépendance entre activités). Nous donnons dans la suite une description des concepts du noyau du paquetage Process Structure.

- *ProcessElement* : C'est un concept abstrait pour représenter une généralisation des éléments d'un procédé (rôles, produits, activités, outils et qualifications).
- *SoftwareActivity* : C'est un concept qui décrit une activité logicielle. Une activité est une unité de travail contrôlée et réalisée dans le but de créer ou de modifier un ou plusieurs produits.
- *Product :* C'est un concept qui représente un produit du développement logiciel. Un produit est un artéfact créé, consommé ou modifié durant le processus de développement. Exemples de produits : code Java, modèle UML, etc.

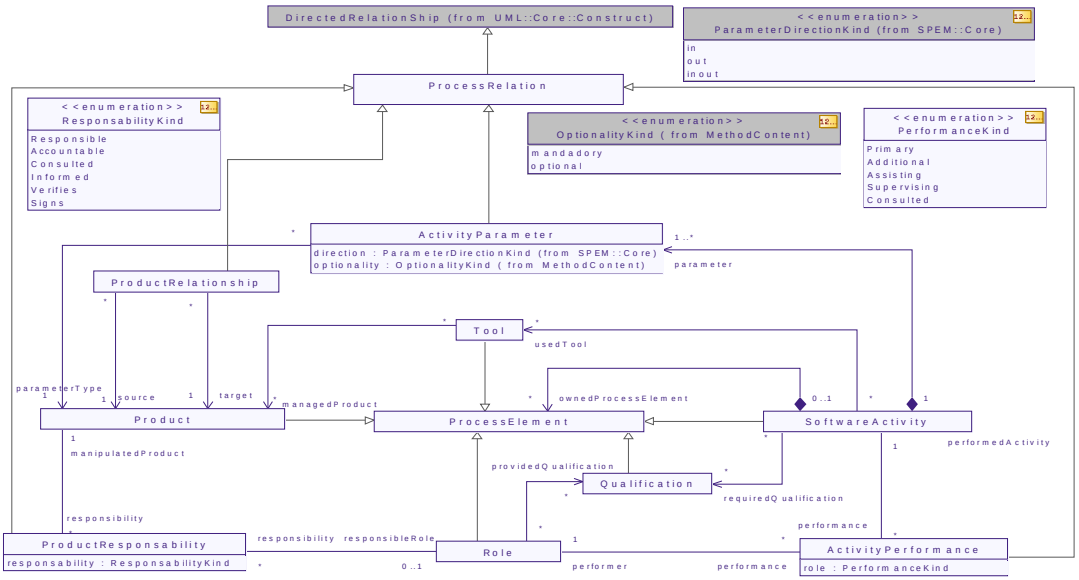

Figure 2. - Présentation du noyau du paquetage Process Structure

- *Tool* : C'est un concept utilisé pour représenter un outil dans le développement logiciel. Un outil assiste un rôle dans l'exécution d'une activité. Exemples d'outils : outil ATL, modeleur UML, etc.
- *Role* : C'est un concept abstrait décrivant les rôles (ensemble de responsabilités et de compétences nécessaires pour effectuer certaines activités de développement). Il décrit la fonction d'un individu ou d'un ensemble d'individus travaillant en équipe (par exemple, le rôle analyste, le rôle concepteur). Les rôles n'identifient pas des individus précis ; les membres individuels d'un projet peuvent jouer plusieurs rôles.
- *Qualification :* C'est un concept utilisé pour représenter les compétences requises pour réaliser une activité ou les compétences fournies par un rôle. Les compétences spécifient les connaissances et l'expérience en méthodes et techniques de développement pour manipuler des produits. Par exemple le rôle concepteur associé à l'activité *Concevoir système*, peut fournir plusieurs compétences (*compétences en UML, compétences en systèmes temps réel embarqués*).
- *ProcessRelation :* C'est un concept abstrait pour représenter une généralisation des relations entre éléments de procédé (relations entre produits, paramètres d'une activité, responsabilité sur un produit, réalisateur d'une activité, relation de dépendances entre activités).

- *ActivityParameter* : Ce concept décrit une relation qui permet d'expliciter les produits manipulés par les activités logicielles. Dans cette relation, la propriété *direction* indique la nature d'un paramètre (produit d'une activité) qui peut être en *entrée*, *sortie* ou en *entrée-sortie*. L'attribut *optionality* permet de spécifier si un paramètre est obligatoire (*mandatory*) ou optionnel (*optional*). Par exemple pour une activité logicielle (*Review Code*), le code à réviser est un paramètre requis et obligatoire en tant que produit en entrée et en sortie alors que le rapport de révision est un paramètre de sortie qui peut être optionnel.
- *ActivityPerformance :* Ce concept décrit une relation qui exprime la participation d'un rôle à l'exécution d'une activité de développement logiciel. Le type de participation est spécifié par l'attribut *role* qui peut avoir une des valeurs suivantes : *Primary* : le réalisateur de l'activité est un rôle primaire ; *Additional* : le rôle est additionnel à l'activité (rajout d'un rôle dans une activité) ; *Assisting* : le rôle est assistant dans l'activité ; *Supervising* : le rôle supervise la réalisation de l'activité ; *Consulting* : le rôle consulte les données de l'activité.
- *ProductResponsability* : Ce concept décrit une relation de responsabilité d'un rôle sur un produit. Le type de responsabilité est spécifié par l'attribut *responsibility* qui peut avoir une des valeurs suivantes : *Responsible :* le rôle est responsable de l'élaboration et de l'utilisation du produit ; *Accountable* : le rôle a une responsabilité quantitative du produit (nombre d'occurrences du produit) ; *Consulted :* le rôle consulte le produit. ; *Informed* : le rôle s'informe sur l'état du produit ; *Verifies :* le rôle vérifie la qualité du produit. ; *Signs* : le rôle approuve et valide le produit.
- *ProductRelationship :* C'est un concept abstrait qui décrit les différentes relations entre produits (agrégation, composition, raffinement, relation d'impact, conformité entre un modèle et son métamodèle). Par manque de place, le paquetage qui décrit la typologie des relations entre produits ne sera pas présenté.

3.3. Typologie des activités dans un procédé logiciel

La figure 3 ci-après décrit les concepts pour représenter les différents types d'activités (*TransformationActivity, Process, Phase, Iteration*) dans un processus de développement logiciel et les relations de dépendance (*ActivityPrecedence*) entre activités. Le mécanisme *Kind* hérité de SPEM 2.0 a été réutilisé pour étendre le noyau du paquetage Process Structure.

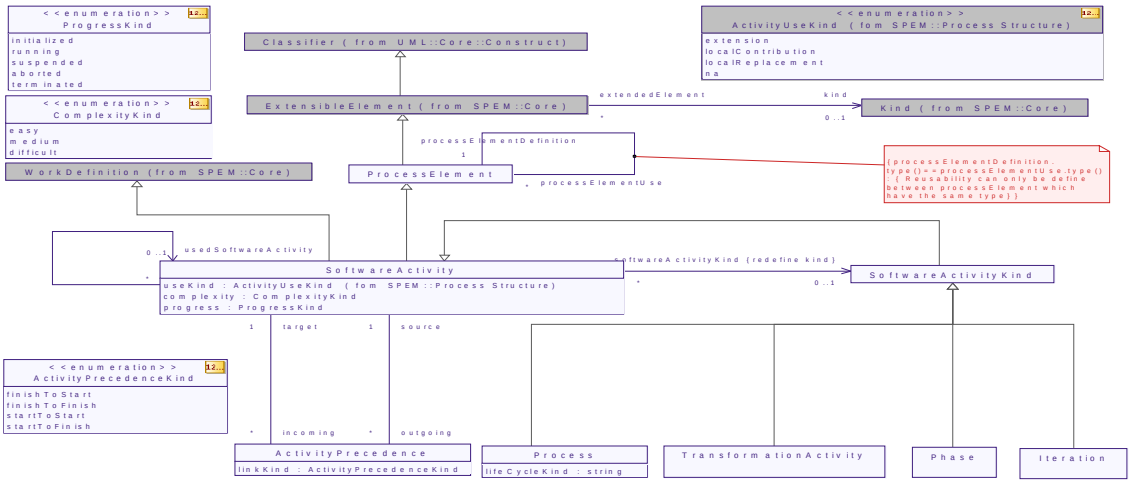

Figure 3. - Typologie des activités de procédé

Le concept *TransformationActivity* décrit toute activité de transformation de modèles dans un processus de développement IDM. La définition la plus générale et qui fait l'unanimité au sein de la communauté IDM [6] consiste à dire qu'une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources. L'association réflexive sur *ProcessElement* et la contrainte structurelle spécifient que tout élément de procédé peut être réutilisé par un ou plusieurs autres éléments de procédé de même nature. Le concept *SoftwareActivity* hérite de

WorkDefinition de SPEM 2 dans le but d'associer une *pré-condition* (condition avant exécution) et une *post-condition* (condition après exécution) à l'exécution d'une activité. L'association réflexive sur *SoftwareActvity* spécifie qu'une activité peut être réutilisée *par une ou plusieurs activités*. Le type de réutilisation est spécifié par l'attribut *useKind.* Par manque de place, nous ne donnons pas la description des autres concepts de ce paquetage.

3.4. Typologie des activités de transformation

La figure 4 ci-dessous montre les concepts qui décrivent une taxonomie de transformations de modèles. Une transformation de modèles est basée soit sur un *programme*, des *règles*, un *template* ou un *pattern*.

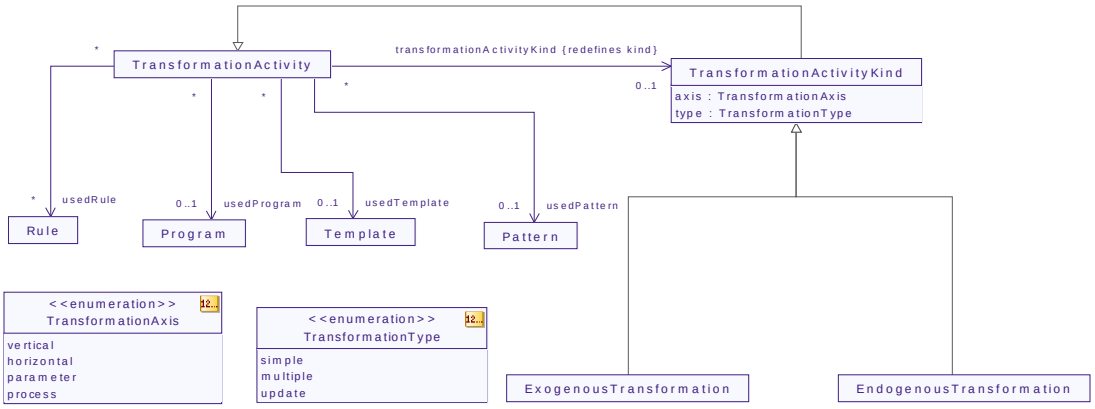

Figure 4. - Typologie des activités d'une transformation

- *TransformationActivityKind* : C'est un concept abstrait pour représenter une généralisation des différents types de transformations de modèles. L'attribut *axis* spécifie l'axe de la transformation qui peut être horizontal, vertical, paramétré ou de type processus. Quant à l'attribut *type,* il spécifie le type de la transformation qui peut être simple (1 modèle vers 1 modèle), multiple (N modèles vers M modèles) ou une transformation sur place (update). Nous rappelons que la composition de modèles et la fusion de modèles sont des cas particuliers de transformation multiple.
- *EndogenousTranformation* : Ce concept représente toute transformation endogène dans un processus de développement orienté modèle. Une transformation est dite endogène si les métamodèles cible et source impliqués dans la transformation sont identiques.
- *ExogenousTranformation* : Ce concept représente toute transformation exogène dans un processus de développement orienté modèle. Une transformation est dite exogène si les métamodèles cible et source impliqués dans la transformation sont différents.

4. Exemple : Procédé de composition VUML

Dans cette section nous validons l'approche de SPEM4MDE en décrivant le procédé de composition de VUML dans une *vision IDM*. VUML [2, 11] est une modélisation UML centrée sur les points de vue. Il propose un processus de composition de modèles pour obtenir le modèle final à partir des modèles par point de vue. La démarche de VUML dans une vision IDM comporte 4 phases :

- *phase 1* : Analyse Globale: C'est la phase centralisée et la modélisation des exigences ;
- *phase 2* : Analyse / Conception par point de vue. Dans cette phase, plusieurs équipes de concepteurs peuvent travailler séparément pour réaliser des modèles de conception par point de vue appelés modèles structurels et des machines-vues (machines à états représentant le cycle de vie des objets-vues) qui spécifient le comportement des objets-vues attendus par l'acteur ;
- *phase 3* : Composition / Fusion. On compose les modèles structurels partiels afin d'aboutir à un diagramme de classes VUML final et les machines-vues pour arriver au comportement global de

l'objet multi-vue. Le mécanisme de composition structurelle comporte deux étapes. La première étape consiste à éliminer les différents conflits (homonymies, etc.). La deuxième étape, automatique, est réalisée par l'intermédiaire de trois règles de transformation :

o règles de correspondance (définition des correspondances entre les modèles de conception par point de vue) ;

o règles de fusion (fusion des modèles par le biais des règles de correspondance pour obtenir le modèle VUML final) ;

o règles de translation (copier les éléments des modèles qui n'ont pas été mis en correspondance dans le modèle VUML final) ;

- *phase 4* : Application d'un patron de conception sur le modèle VUML final pour générer le modèle d'implémentation objet qui générera ensuite le code objet.

La figure 5 ci-dessous décrit le procédé de composition de VUML en utilisant les concepts du métamodèle SPEM4MDE. Nous avons d'abord traduit une partie du métamodèle SPEM4MDE en un profil UML pour pouvoir faire la description du procédé de composition VUML. Le procédé est décrit par une séquence de transformations de modèles. A partir du modèle des exigences en UML de type CIM (Computational-Independent Model), le concepteur en UML réalise une transformation endogène multiple pour produire les modèles de conception en UML de type PIM (Platform-Independent Model) pour chaque point de vue. Par la suite les modèles de conception en UML sont composés par le concepteur VUML pour produire un modèle VUML final de type PIM. A partir du modèle VUML final, le concepteur objet applique un patron relatif à la plate-forme objet pour produire le modèle d'implémentation objet de type PSM (Paltform-Specific Model). Enfin, le programmeur objet génère du code objet à partir du modèle d'implémentation objet.

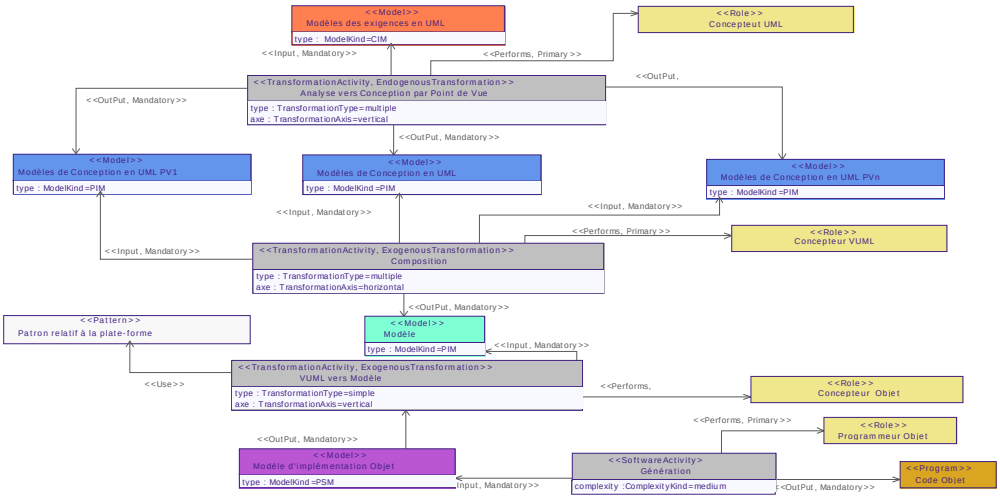

Figure 5.- Représentation du procédé de composition VUML avec SPEM4MDE

5. Conclusion

Nous avons présenté dans cet article une nouvelle approche pour la spécification des procédés à base de transformations de modèles. Cette approche est basée sur un métamodèle dénommé SPEM4MDE : un LDP dédié à la modélisation et à la mise en œuvre des processus à base de modèles. Comme nous l'avons dit dans l'introduction, nous avons plus particulièrement mis l'accent sur les concepts qui permettent de décrire statiquement un procédé IDM afin d'assister les développeurs dans l'élaboration et la génération des modèles.

Le développement logiciel dirigé par les modèles a pour principal objectif de concevoir des applications en séparant les préoccupations et en plaçant les notions de modèles, métamodèles et

transformations de modèles au centre du processus de développement. Les transformations nécessitent des langages dédiés (ATL, QVT,…), des outils de métamodélisation flexibles (Kermeta, EMF/Ecore, TOPCASED,…) qui pourront assister les concepteurs de systèmes logiciels complexes. L'outillage permettant de formaliser toutes ces actions dans un développement logiciel devient plus qu'une nécessité. C'est dans cette perspective que nous travaillons actuellement à l'élaboration d'un prototype d'environnement permettant de modéliser un procédé IDM en utilisant les concepts de SPEM4MDE.

Bibliographie

1. AndroMDA (2008). AndroMDA.org available at: http://www.andromda.org/
2. Anwar A, Ebersold S., Coulette B., Nassar M., Kriouile A. «Vers une approche à base de règles pour la composition de modèles. Application au profil VUML.» *Revue RTSI série L'OBJET*, numéro spécial Ingénierie Dirigées par les Modèles. Vol 13, n°4/2007.
3. ANR-DOMINO, Définition des activités, composants et modèles MDA pour le projet DOMINO, Rapport interne ANR, 2007
4. ATLAS, KM3: Kernel MetaMetaModel, Technical Report, LINA&INRIA, NANTES, aug. 2005.
5. Bendraou R., *UML4SPM : Un langage de modélisation de procédé de développement de logiciel exécutable et orienté modèle*, Thèse de doctorat, Université de Paris VI, 06 Septembre 2007.
6. Bézivin J., « Sur les principes de base de l'ingénierie des modèles » *RTSI-L'Objet*, 10(4) : Page 145-157, 2004.
7. Compuware-website, OptimalJ at: http://www.compuware.com/products/optimalj/,2003
8. LeBlanc H., Millan T., Ober I., «Démarche de développement orientée modèles : de la vérification de modèles à l'outillage de la démarche », *IDM'05*, – Premières Journées sur l'Ingénierie Dirigée par les Modèles, Paris, 30 Juin au 1er Juillet 2005.
9. Kabbaj M.,. Lbath R., Coulette B., « A Deviation Management System for Handling Software Process Enactment Evolution», Proc. International Conference ICSP, co-located with ICSE. Springer-Verlag ? LNCS, 2008.
10. Kabbaj M.,. Lbath R., Coulette B., « A Deviation-tolerant Approach to Software Process Evolution». In: Proc. 9th IWPSE 2007, in conjunction with ACM 6th ESEC/FSE Joint Meeting, Dubrovnik, Croatia, pp. 75-78, 2007.
11. Nassar M. *Analyse/conception par objets et points de vue : le profil VUML*. Thèse INPT, Toulouse, 28 septembre 2005.
12. NEPTUNE-website, available at: http://www.neptune.irit.fr
13. OAW. available at : http://www.openarchitectureware.org/
14. OMG, MOF2.0, http://www.omg.org/cgi-bin/doc?formal/06-01-01, 2006
15. OMG, MDA, http://www.omg.org/mda/specs.htm, 2001
16. OMG, QVT 2.0 Transformation Spec., http://www.omg.org/spec/QVT/1.0/PDF/, Avril 2008
17. OMG, UML 2.0 Superstructure, http://www.omg.org/cgi-bin/doc?formal/05-07-04.pdf, 2005a
18. OMG, UML 2.0 Infrastructure, http://www.omg.org/cgi-bin/doc?formal/05-07-05.pdf, 2005b
19. OMG, SPEM 2, http://www.omg.org/spec/SPEM/2.0, Avril 2008
20. Ribo Josep M., Franch Xavier, « A Precedence-based Approach for Proactive Control in Software Process Modeling. » *SEKE'05,* Park, Seunghun 22 Juin- 05, 2005.
21. Softeam, support de formation Objecteering 6/MDA Modeler version 2.0, Janvier 2008.
22. TOPCASED-WP5, staff, Guide méthodologique pour les transformations de modèles. Rapport de recherche, version 0.1, 18 Novembre 2008, IRIT/MACAO
23. TOPCASED-website, available at: http://www.topcased.org
24. Triskell, Kermeta, IRISA, Rennes, http://www.irisa.fr/triskell, http://www.kermeta.org, 2005
25. Feiler, P. H., Humphrey, W. S., "Software process development and enactment: Concepts and definitions". In Proceedings of the *International Conference on Software Process ICSP 93*, Berlin, Germany, 1993.