Refactoring des modèles : concepts et défis

Khalid Allem[†‡] — **Tom Mens**[†*]

[†] *Service de Génie Logiciel, Université de Mons-Hainaut*
Avenue du Champ de Mars 6, 7000 Mons, Belgique

[‡] *SmalS-MvM/Egov, Rue du Prince Royal 102, 1050 Bruxelles, Belgique*

[*] *LIFL (UMR 8022), Université Lille 1 - Projet INRIA Jacquard*
Cité Scientifique, 59655 Villeneuve d'Ascq Cedex, France

RÉSUMÉ. *Dans le cadre de l'ingénierie dirigée par les modèles (IDM), l'activité de refactoring des modèles est peu étudiée. Cette activité consiste à restructurer un modèle pour en améliorer ses facteurs de qualité, tels que l'adaptabilité, la compréhensibilité et l'efficacité, tout en préservant le comportement de l'application logicielle liée à ce modèle. Avec cet article court, nous structurons le domaine de recherches de refactorings des modèles, et nous proposons quelques pistes de travail. Notamment, nous discuterons de la préservation du comportement et de la cohérence intra-modèle et inter-modèles. Nous examinerons également les technologies mises en oeuvre pour la réalisation des outils de refactoring des modèles.*

ABSTRACT. *In model-driven engineering (MDE), the activity of model refactoring is not sufficiently addressed. Model refactoring aims to improve the structure of a model while preserving the external behaviour and other quality characteristics such as adaptability, understandability and performance. In this short article we shed more light on the current state-of-the-art in this important research domain, and we suggest important avenues of further research.*

MOTS-CLÉS : *évolution logicielle, qualité logicielle, transformation de modèles, refactoring, ingénierie dirigée par les modèles.*

KEYWORDS: *software evolution, software quality, refactoring, model transformation, model-driven engineering.*

1. Introduction

Le refactoring[1] est une activité d'ingénierie logicielle qui consiste à modifier le code source d'une application de manière à améliorer sa qualité tout en préservant son comportement extérieur vis-à-vis de ses utilisateurs. La plupart des environnements de développement contemporains supportent l'application automatique des refactorings au niveau du code source. La première apparition des refactorings est due à (Opdyke, 1992). (Fowler, 1999) a repris certains de ces refactorings et créé un catalogue de 72 refactorings du code Java. (Kerievsky, 2004) a proposé un catalogue de refactorings qui permettent de restructurer du code en introduisant des *design patterns*.

Dans l'ingénierie logicielle dirigée par les modèles (IDM), les modèles jouent un rôle prépondérant. Il est donc logique et utile d'étudier les techniques de refactoring au niveau des modèles. Dans cette perspective nous parlerons du *refactoring des modèles*. Dans cet article, nous ferons le point sur les mécanismes et techniques liées au processus de refactoring des modèles. Nous passerons en revue les défis les plus importants et les futures orientations de recherche liées à la mise en oeuvre du refactoring des modèles, et nous discuterons de la pertinence de son utilisation. Pour pallier le manque de support de refactoring dans les outils contemporains de modélisation logicielle et de transformation de modèles, nous discuterons des difficultés de gestion de refactoring des modèles, et nous proposerons quelques possibilités pour contourner ces difficultés. Ceci permettrait dans un avenir proche d'automatiser la technique de refactoring des modèles.

2. Le processus du refactoring des modèles

Selon (Mens *et al.*, 2004), le processus de refactoring se divise en plusieurs activités qui peuvent être résumées de la façon suivante : (1) Identifier quelles parties du modèle devraient être refactorisées ; (2) Déterminer quels refactorings devraient être appliqués à ces endroits ; (3) Garantir qu'une fois appliqué, le refactoring des modèles préserve le comportement et la cohérence ; (4) Automatiser l'application du refactoring ; (5) Evaluer l'impact du refactoring sur des critères de qualité logicielle (complexité, lisibilité, adaptabilité) ou du processus (productivité, coût, effort) ; (6) Synchroniser le modèle refactorisé et les autres artefacts tels que le code source, la documentation, les spécifications, les tests, etc.

L'un des défis que les environnements de modélisation doivent relever consiste à prendre en charge ces différentes activités d'une manière automatique. Cette automatisation est liée à plusieurs problèmes qui devraient être pris en compte :

1) *Le rapport gain / risque*. Il est nécessaire de comparer les gains avec les risques encourus. Selon le contexte, il peut être préférable de ne pas appliquer un refactoring si le gain attendu est faible en comparaison des risques pris. Afin de mieux comprendre

1. "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure" (Fowler, 1999).

les gains et risques liés au refactoring des modèles, nous devons nous baser sur des études empiriques des modèles réels utilisés par l'industrie.

2) *Tests de non-regression.* Le refactoring d'un modèle modifie par définition la structure du modèle existant. Les tests de non-regression doivent donc être adaptés afin de garantir un refactoring réussi. Autrement, la batterie de tests risque de ne plus être applicable après le refactoring (van Deursen *et al.*, 2002).

3. Comment spécifier les refactorings de modèles ?

Afin de pouvoir automatiser les refactorings de modèles, nous devons d'abord les spécifier. Une implémentation *ad hoc* dans un outil de modélisation n'est pas une solution idéale parce que les refactorings deviennent dépendants de l'outil de modélisation. Le défi consiste donc à trouver une spécification plus générique de refactoring des modèles qui est à la fois indépendant de l'outil de modélisation utilisé, et indépendant du langage de modélisation (Tichelaar *et al.*, 2000; Lämmel, 2002).

Une question ouverte concerne quel type de langage est le mieux adapté pour exprimer des transformations de modèles. Les langages impératifs sont plus proches de ce qui est déjà connu par la plupart des développeurs, alors que les langages déclaratifs ont souvent des propriétés intéressantes pour faciliter la manipulation des transformations (Mens *et al.*, 2006).

Dans l'ingénierie dirigée par les modèles, il semble utile d'utiliser des outils existants de transformation de modèles pour spécifier et automatiser le refactoring des modèles. Une alternative consiste à donner une spécification formelle. Par exemple, (Mens, 2006; Mens *et al.*, 2007) propose d'utiliser la théorie de transformation de graphes pour la spécification de refactorings de modèles.

4. Détection et amélioration de la qualité des modèles

L'objectif du refactoring des modèles est d'améliorer les caractéristiques de qualité des modèles logiciels (p.e., compréhensibilité, portabilité, adaptabilité, réutilisabilité, modularité, complexité). En améliorant la qualité du modèle, nous atteindrons un gain maximal au niveau de sa maintenabilité et de son évolutivité, ce qui facilitera les extensions futures.

Lorsqu'on applique un refactoring, on peut essayer d'évaluer cette amélioration en mesurant les attributs internes de qualité du modèle avant et après le refactoring (Du Bois, 2006). Ces attributs peuvent être mesurés grâce à des métriques logicielles (Fenton *et al.*, 1997). Au niveau du code source, les métriques logicielles sont fort étudiées et souvent utilisées. Au niveau des modèles, ce n'est pas le cas. Bien sûr, pour un diagramme de classes, on peut utiliser des métriques similaires à celles utilisées au niveau du code source. Mais comment mesure-t-on, par exemple, la qualité d'une machine à états ou d'un diagramme de composants ? Pire encore, comment mesure-t-

on la qualité d'un modèle qui est composé de différentes sortes de diagrammes ? Le défi est donc de faire avancer l'état de l'art au niveau des métriques de modèles, afin de mesurer la qualité logicielle au niveau des modèles, et afin de connaître l'effet d'un refactoring sur cette qualité. Un problème important à résoudre dans ce contexte est la présence des objectifs de qualité contradictoires. Il est donc essentiel d'étudier la relation entre les transformations de modèles et les attributs de qualité (Tahvildari *et al.*, 2004).

Une technique moins formelle, mais néanmoins très utilisée pour identifier les endroits du logiciel susceptibles d'être refactorisés est la détection de *bad smells* (Fowler, 1999). Il s'agit de structures dans le logiciel qui indiquent un appauvrissement de la qualité logicielle. Au niveau du code source, les bad smells sont bien connus. Le défi est d'identifier et d'automatiser la détection de *design smells* (c.à.d. de bad smells au niveau de modèles), ainsi que leur relation avec le refactoring des modèles.

5. Préservation du comportement de modèle

Par définition, un refactoring ne devrait pas modifier le comportement ou les fonctionnalités d'un logiciel. La première définition de la conservation du comportement a été donnée par (Opdyke, 1992). Il stipule que, pour les mêmes valeurs d'entrée, l'ensemble des valeurs de sortie résultant doit être le même avant et après l'application du refactoring. Mais ce n'est pas une contrainte assez forte dans certains domaines d'application. Par exemple, elle ne tient pas compte d'autres critères comportementaux importants tels que les contraintes de temps, de mémoire, de consommation d'énergie, de synchronisation entre processus distribués, etc. Dans les applications où le traitement de données joue un rôle prépondérant, une autre contrainte essentielle est la préservation de la cohérence de ces données.

En UML, certaines sortes de diagrammes décrivent le comportement (p.e., les machines à états, les diagrammes d'interaction), alors que d'autres décrivent la structure (p.e., les diagrammes de classes et les diagrammes de composants). Le défi est donc de trouver une bonne définition du comportement d'un modèle, afin de pouvoir garantir la conservation de ce comportement lors d'un refactoring. Une définition formelle et mathématique peut s'avérer nécessaire (Van Kempen *et al.*, 2005).

Au niveau de code source, la programmation par contrat (Meyer, 1992) est souvent proposée pour préserver certains aspects du comportement, en spécifiant des préconditions, postconditions et invariants sur les transformations. Au niveau des modèles, la même technique peut être exploitée, par exemple en utilisant le langage de contraintes OCL pour "contractualiser" certaines propriétés comportementales des modèles (Correa *et al.*, 2004; Markovic *et al.*, 2005).

6. Préservation de cohérence de modèle

Le design d'un logiciel est souvent modélisé en utilisant différentes sortes de diagrammes. Par exemple, un modèle UML est typiquement constitué de diagrammes de classes, d'interaction, de cas d'utilisation, d'états, etc. A cause de cette variété de modèles utilisés lors de la modélisation, un problème apparaît lorsqu'on veut appliquer un refactoring. La refactorisation d'un diagramme peut engendrer des incohérences dans les autres diagrammes. Le défi est donc d'adapter tous les diagrammes associés afin de préserver leur cohérence avec le diagramme refactorisé. Ce principe est schématisé dans la Figure 1 (première figure). Une approche formelle (p.e., la logique de descriptions) peut faciliter le processus de gestion de cohérence (Simmonds *et al.*, 2004).

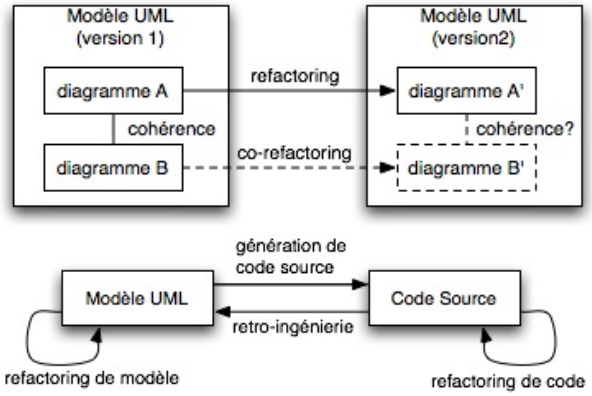

Figure 1. *Préservation de cohérence entre modèles et code source.*

Le même problème peut se produire lors de la phase de développement, quand on utilise des outils automatisés pour générer du code source à partir d'un modèle quelconque. Dans l'état actuel des choses, le code source généré doit souvent être adapté par le développeur. Dans cette perspective, des refactorings peuvent être appliqués au modèle ou au code source. Ceci est schématisé dans la Figure 1 (deuxième figure). Lors d'un refactoring du modèle ou du code source associé, le défi consiste à préserver la cohérence entre le modèle et son code source, ou de les synchroniser après le refactoring (Van Gorp *et al.*, 2003). De ce fait, avant et après toute opération de refactoring, il est nécessaire de mettre au point un mécanisme de traçabilité et de vérification de contraintes entre le modèle et le code source généré à partir de ce modèle. Cette vérification peut être spécifiée de manière *ad hoc* (mais néanmoins exécutable), en utilisant une batterie de tests, ou de manière précise, p.e., se basant sur la théorie de vérification formelle.

7. Refactoring des modèles génériques

Les refactorings peuvent être employés pour rendre un modèle plus réutilisable et plus générique. L'introduction des *design patterns* entre dans cette perspective, en offrant des solutions génériques éprouvées à des problèmes récurrents rencontrés dans la conception des modèles (Kerievsky, 2004). En assurant un respect des meilleures pratiques orientées objet, l'utilisation des design patterns dans un modèle est très importante pour améliorer sa qualité. L'un des défis à relever est de comprendre comment on peut améliorer la réutilisabilité et la généricité d'un modèle afin de pouvoir réutiliser ce modèle dans différents domaines, que ce soit dans un domaine métier ou dans un domaine technique.

8. Résumé

Cet article visait à montrer la nécessité d'introduire les concepts de refactoring au niveau des modèles. D'un point de vue historique, les outils de développement logiciel se sont tout d'abord concentrés sur le refactoring de code source. Cependant, la recherche actuelle devrait s'intéresser à l'implémentation de refactoring des modè–les dans les outils de modélisation logicielle et dans les outils de transformation de modèles. Les développeurs de ces outils doivent prendre conscience de divers aspects liés à l'amélioration de la qualité de modèle, la conservation du comportement des modèles, la traçabilité et la préservation de la cohérence des modèles. Ensuite, ils doivent utiliser cette connaissance pour adapter les processus et outils de refactorings contemporains afin d'automatiser le refactoring des modèles.

Des premiers travaux de recherche dans ce domaine intéressant ont été effectués mais beaucoup reste à faire. Avec cet article, nous espérons avoir dévoilé des pistes intéressantes dans ce domaine de recherche qui devient de plus en plus important.

9. Bibliographie

Correa A., Werner C., « Applying Refactoring Techniques to UML/OCL Models », *UML 2004 - The Unified Modeling Language*, vol. 3273 of *Lecture Notes in Computer Science*, Springer, p. 173-187, 2004.

Du Bois B., A Study of Quality Improvements by Refactoring, PhD thesis, Universiteit Antwerpen, Belgium, September, 2006.

Fenton N., Pfleeger S. L., *Software Metrics : A Rigorous and Practical Approach*, second edn, International Thomson Computer Press, London, UK, 1997.

Fowler M., *Refactoring : Improving the Design of Existing Code*, Addison-Wesley, 1999.

Kerievsky J., *Refactoring to patterns*, Addison-Wesley, 2004.

Lämmel R., « Towards Generic Refactoring », *Third ACM SIGPLAN Workshop on Rule-Based Programming*, ACM, p. 15-28, 2002.

Markovic S., Baar T., « Refactoring OCL Annotated UML Class Diagrams », *Proc. Int'l Conf. Model Driven Engineering Languages and Systems (MoDELS 2005)*, vol. 3713 of *Lecture Notes in Computer Science*, Springer, p. 280-294, 2005.

Mens T., « On the use of graph transformations for model refactoring », *Generative and transformational techniques in software engineering*, vol. 4143 of *Lecture Notes in Computer Science*, Springer, p. 219-257, 2006.

Mens T., Taentzer G., Runge O., « Analyzing Refactoring Dependencies Using Graph Transformation », *Software and Systems Modeling*, 2007.

Mens T., Tourwe T., « A Survey of Software Refactoring », *Transactions on Software Engineering*, vol. 30, n° 2, p. 126-162, February, 2004.

Mens T., Van Gorp P., « A Taxonomy of Model Transformation », *Electronic Notes in Theoretical Computer Science*, vol. 152, p. 125-142, 2006.

Meyer B., « Applying "Design by Contract" », *Computer*, vol. 25, n° 10, p. 40-51, 1992.

Opdyke W. F., Refactoring : A Program Restructuring Aid in Designing Object-Oriented Application Frameworks, PhD thesis, University of Illinois at Urbana-Champaign, 1992.

Simmonds J., Van Der Straeten R., Jonckers V., Mens T., « Maintaining Consistency between UML Models Using Description Logic », *Série L'objet - logiciel, base de données, réseaux*, vol. 10, n° 2-3, p. 231-244, 2004.

Tahvildari L., Kontogiannis K., « Improving Design Quality Using Meta-Pattern Transformations : A Metric-Based Approach », *Journal of Software Maintenance and Evolution : Research and Practice*, vol. 16, n° 4-5, p. 331-361, 2004.

Tichelaar S., Ducasse S., Demeyer S., Nierstrasz O., « A Meta-Model for Language-Independent Refactoring », *Proc. Int'l Symp. Principles of Software Evolution*, IEEE Computer Society, p. 157-169, 2000.

van Deursen A., Moonen L., « The Video Store Revisited – Thoughts on Refactoring and Testing », *Proc. 3rd Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering*, p. 71-76, 2002. Alghero, Sardinia, Italy.

Van Gorp P., Stenten H., Mens T., Demeyer S., « Towards automating source-consistent UML refactorings », *UML 2003 - The Unified Modeling Language*, vol. 2863 of *Lecture Notes in Computer Science*, Springer, p. 144-158, 2003.

Van Kempen M., Chaudron M., Koudrie D., Boake A., « Towards Proving Preservation of Behaviour of Refactoring of UML Models », *Proc. SAICSIT 2005*, p. 111-118, 2005.

ANNEXE POUR LE SERVICE FABRICATION

A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LES ACTES :
   IDM 2007

2. AUTEURS :
   Khalid Allem[†‡] *— Tom Mens*[†*]

3. TITRE DE L'ARTICLE :
   Refactoring des modèles : concepts et défis

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Refactoring des modèles

5. DATE DE CETTE VERSION :
   4 janvier 2007

6. COORDONNÉES DES AUTEURS :

   – adresse postale :

   [†] Service de Génie Logiciel, Université de Mons-Hainaut
   Avenue du Champ de Mars 6, 7000 Mons, Belgique

   [‡] SmalS-MvM/Egov, Rue du Prince Royal 102, 1050 Bruxelles, Belgique

   [*] LIFL (UMR 8022), Université Lille 1 - Projet INRIA Jacquard
   Cité Scientifique, 59655 Villeneuve d'Ascq Cedex, France

   – téléphone :

   – télécopie :

   – e-mail :

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.22 du 04/10/2005.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com