IDM pour le passage objet/relationnel

Georges Louis, Marie Christine Lafaye, Antoine Wiedemann

IDM pour le passage objet/relationnel

1 Introduction

Notre travail porte sur la production semi-automatique d'un schéma relationnel à partir d'un diagramme de classes UML [1] représentant un modèle du domaine. Bien que cette question soit déjà largement abordée dans la littérature, l'automatisation du savoir faire des concepteurs de bases de données est loin d'être entièrement réalisée. Les travaux sur le sujet se focalisent sur la richesse des types de structures de données offerts par le modèle objet (donc celles du modèle source) et étudient les possibilités de leur implantation dans le modèle relationnel. A contrario, dans notre approche, nous nous intéressons à la qualité du modèle relationnel cible. Le processus de transformation que nous proposons assure l'exhaustivité de la collecte des contraintes d'unicité existantes, la qualité des contraintes d'unicité proposées par le concepteur, et celle des clés primaires et étrangères. Nous générons des schémas où il existe des valeurs nulles, y compris sur les clés étrangères, et nous traitons dans notre processus les diagrammes états-transitions représentant la dynamique des objets [2, 3, 4, 5].

2 Notre intérêt pour l'IDM

L'exemple du passage d'UML au relationnel est couramment utilisé pour présenter l'ingénierie dirigée par les modèles. Nous avons choisi cette technique pour mettre en œuvre notre proposition de transformation. Nous nous appuyons sur les méta modèles standard UML et CWM [6] comme méta modèles de départ et d'arrivée de notre transformation. Dans cette architecture, l'utilisation des méta modèles améliore la qualité des règles de transformation, et par là la qualité de la transformation elle-même en garantissant que le modèle résultat est conforme au méta modèle cible. Les éléments d'entrée et de sortie sont clairement typés par leurs métamodèles respectifs, et les règles de transformation peuvent être exprimées en termes de méta élément. C'est sans doute un des grands apports de l'IDM.

L'utilisation de méta modèles standards permet de tendre vers une compatibilité entre la transformation que nous développons et d'autres traitements (production, réutilisation. . .) de modèles. Nous acceptons en entrée des modèles UML qui peuvent provenir de nombreuses sources, et, de même, les modèles que nous produisons peuvent être réutilisés par les outils qui implantent CWM.

Dans l'IDM, l'intérêt de la méta modélisation nous paraît évident. Cette ingénierie propose aussi des techniques de transformation de modèles. Or, alors que les concepts de la méta modélisation sont stables et connaissent des normes relativement consensuelles depuis plusieurs années [7], les outils de transformations sont plus hétérogènes (ATL [8], KerMeta [9] ...). La normalisation récente d'un langage de transformation de modèles par l'OMG [10] n'a pas masqué ces différences de point de vue.

2.1 Avec un langage déclaratif

Nous avons expérimenté le langage de transformation de modèles ATL, qui permet d'exprimer la transformation sous forme déclarative, par des régles de mapping entre les méta classes sources et les méta classes cibles. Chaque règle fait correspondre à un élément du modèle d'entrée (i.e. à chaque instance d'une méta classe du méta modèle source) un ou plusieurs éléments du modèle de sortie. Le moteur de transformation maintient nativement une trace de l'exécution de la transformation.

Notre processus de transformation se déroule en plusieurs étapes, entre lesquelles une interaction avec l'utilisateur est nécessaire. Il ne peut pas s'exprimer sous forme d'une seule transformation ATL. L'enchaînement de plusieurs transformations rend plus difficile la traçabilité, et les modèles intermédiaires sont incomplets et n'ont pas réellement de statut.

Dans notre application, les éléments du modèle d'entrée doivent être traités différemment en fonction de leur type (classe ou association par exemple), de leur propriétés (une association en fonction de ses multiplicités), mais aussi de leurs relations avec les autres éléments (une classe a-t-elle une généralisation ?). Certaines situations exigent même qu'un élément soit transformé par deux règles distinctes (cas d'une classe-association). La définition d'une règle par méta élément s'avère insuffisante pour prendre en compte

tous ces paramètres, et il faut alors en créer plusieurs (par exemple pour les associations de multiplicités différentes). Leur multiplication rend leur maintenance difficile.

2.2 Avec un langage impératif

Aujourd'hui, nous travaillons entièrement avec un langage impératif à objets (python), en utilisant un patron qui recrée un fonctionnement par règles de transformation. La trace de la transformation est maintenue à jour par le programme, tandis que l'ordre d'exécution de la transformation est totalement maîtrisé. Ce que nous devions décomposer en plusieurs transformations avec ATL est réuni en un seul programme qui assure l'interactivité avec le concepteur. Un avantage est une évidente maîtrise du déroulement de la transformation. Un inconvénient est que les règles sont incluses dans du code utilitaire de chargement du modèle source et d'écriture du modèle cible, rendant leur lecture moins aisée. L'utilisation d'un langage généraliste facilite en tout cas le développement de l'interface graphique permettant le dialogue utilisateur.

Pour bénéficier du meilleur des deux mondes (déclaratif et impératif), il serait intéressant de pouvoir décrire avec un langage impératif un processus comprenant plusieurs transformations exprimées sous forme déclarative. Notre prototype logiciel reprend cette idée dans son architecture.

3 Conclusion

Dans notre exploitation de l'IDM nous utilisons donc surtout les concepts de modèle et de méta modèle. En comparant les deux prototypes réalisés (l'un de type déclaratif, l'autre avec un langage impératif), nous préférons poursuivre le prototypage en impératif pour exploiter non seulement le modèle du domaine mais aussi le diagramme états/transitions associés aux objets du domaine. Dans cette nouvelle étape, nous aurons à maîtriser le maintien de la cohérence inter modèles et la composition de modèles.

Références

[1] OMG. Unified modeling language : Superstructure, 2007.

[2] M. C. Lafaye, J. Y. Lafaye, J. P. Paillard, and S. Téouri. Qualité de conception des bases de données relationnelles avec UML. *Ingénierie des Systèmes d'Information*, 9(5-6) :69–101, 2004.

[3] M. C. Lafaye, J. Y. Lafaye, G. Louis, and A. Wiedemann. L'approche MDA pour la conception des bases de données relationnelles, une illusoire simplicité. *Actes des 3èmes journées sur l'Ingénierie Dirigée par les Modèles*, 1 :167–181, 2007.

[4] A. Wiedemann. Approche MDA pour la transformaion d'un modèle UML en schéma relationnel. In *Actes du XXVème congrès INFORSID, forum jeunes chercheurs*, pages 587–588. Association INFORSID, 2007.

[5] Antoine Wiedemann. Relational database modeling. In *MODELS'08*, Doctoral Symposium, Toulouse, 2008. Springer, LNCS.

[6] OMG. Common Warehouse Metamodel (CWM) specification, 2001.

[7] OMG. Model driven architecture (MDA) guide version 1.0.1, 2003.

[8] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez. Atl : A qvt-like transformation language. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, volume 2006 of *21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, OOPSLA 2006*, pages 719–720, Portland, OR, 2006.

[9] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In *Model Driven Engineering Languages and Systems*, pages 264–278. LNCS, 2005.

[10] OMG. MOF Query / Views / Transformations, 2007.