A faire en dernier :-)

Table des matières

Chapitre 1

État de l'art

L'Ingénierie Dirigée par les Modèles (IDM) est née du constat que le paradigme du « tout est objet », prôné dans les années 1980, a atteint ses limites avec ce début de siècle [Greenfield 2004]. En effet, face à la croissance de la complexité des systèmes logiciels, au coût de la main d'œuvre et de maintenance, une approche centrée sur le code, jugé alors seul représentant fiable du système, suscitait de moins en moins l'adhésion des industriels et du milieu académique.

Partant de ce constat, l'Object Management Group (OMG) a proposé en novembre 2000, l'approche MDA (Model Driven Architecture) qui s'inscrit dans le cadre plus général de l'IDM et se réalise autour d'un certain nombre de standards tels qu'UML, MOF, XML, QVT, etc. Le monde de la recherche s'y est aussitôt intéressé pour dégager les principes fondamentaux de l'IDM [Bézivin 2001][Kent 2002] [De Lara 2002] et déjouer le piège des définitions parfois trop floues qui prêtent à confusion entre les concepts liés aux paradigmes d'objet et de modèle [Bézivin 2004a]. Par ailleurs, des industriels comme IBM [Booch 2004] et Microsoft [Greenfield 2004] ont aussi rendu publiques leur vision de l'IDM. Ainsi, l'IDM prend son origine dans la convergence de toutes ces visions et des avancées techniques de chacun.

L'originalité de l'IDM ne réside pas dans le recours systématique aux modèles dans le développement logiciel comme le laisserait entendre sa terminologie[Bézivin 2004b]. Plusieurs méthodes de modélisation telles que Merise ou SSADM préconisent aussi l'utilisation de modèles dont le rôle s'achève aux phases amont du développement logiciel : l'analyse et la conception. Les modèles servent alors à faciliter la communication et compréhension entre les différents acteurs mais n'interviennent pas dans la phase de production, de maintien et d'évolution. Nous parlons dans ce cas de modèles « contemplatifs ».

L'IDM a pour objectif de rendre les modèles « productifs » sur tout le cycle de vie du système et à tout niveau d'abstraction. Pour y parvenir, les modèles doivent être décrits formellement pour être interprétés et exécutés par une machine. Dès lors, ces modèles permettent d'industrialiser la production logicielle, jusque-là centrée sur le code produit par l'informaticien [Bézivin 2005].

En mettant à profit des disciplines comme la modélisation par objets, l'ingénierie des langages, la compilation de langages, les méthodes formelles, la programmation par composants, etc., l'IDM offre un cadre intégrateur reposant sur quelques concepts fondamentaux : la notion de modèle et la relation *ReprésentationDe*, la notion de métamodèle et la relation *ConformeÀ*.

1.1 Concepts et relations de l'IDM

1.1.1 Modèle et ReprésentationDe

La notion de modèle est centrale dans l'IDM car, comme nous venons de voir, l'enjeu de cette approche est de rendre les modèles productifs sur tout le cycle de vie du système. Il n'existe pas de définition universelle de la notion de modèle. En nous appuyant sur les définitions données dans les travaux [Minsky 1967] [Bézivin 2001] et [Seidewitz 2003], nous adoptons la définition suivante du terme modèle :

Définition 1. Un modèle est une abstraction d'un système, selon le bon point de vue, qui permet de répondre à des questions prédéfinies sur ce système en lieu et place de celui-ci.

De cette définition découle la première relation fondamentale de l'IDM qui lie le modèle et le système qu'il représente. Celle-ci est nommée *ReprésentationDe* et notée (μ). Bien que la relation *ReprésentationDe* ne soit pas nouvelle dans l'ingénierie logicielle (Merise, UML), l'IDM a permis d'en définir les contours [Atkinson 2003] [Seidewitz 2003] [Bézivin 2004a].

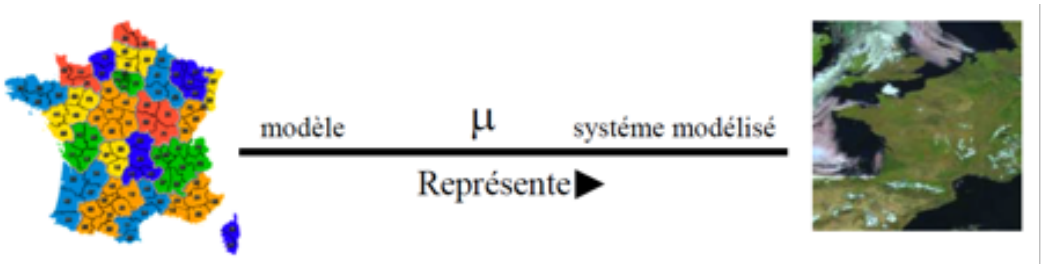

FIGURE 1.1 – Relation entre système et modèle [FAVRE 2006]

Cette définition n'est pas restreinte à l'informatique et pourrait s'appliquer à n'importe quel système. La figure 1.1 reprend l'exemple connu de la cartographie où une carte géographique joue le rôle de modèle pour une région donnée jouant alors le rôle de système modélisé.

L'intérêt de l'IDM est de produire des modèles exploitables informatiquement. Ceci n'est possible que si ces modèles sont décrits par des langages formels. Il devient alors important de bien définir ces langages à l'aide de métamodèles

1.1.2 Métamodèle et ConformeÀ

L'originalité de l'IDM ne réside pas dans la relation ReprésentationDe qui trouve plutôt son origine dans les méthodes de modélisation telles que Merise ou SSADM. L'apport de l'IDM est dans l'utilisation systématique de métamodèles pour la description des langages de modélisation.

Il existe plusieurs définitions de la notion de métamodèle dans la littérature. Cependant la définition suivante est communément admise [Bézivin 2004b].

Définition 2. Un métamodèle est un modèle du langage de modélisation qui sert à exprimer les modèles.

Une autre définition courante mais erronée de la notion de métamodèle suppose qu'un métamodèle est un modèle d'un modèle. La figure 1.2 reprend l'exemple de la cartographie évoquée plus haut. Nous appliquons récursivement la relation *ReprésentationDe* (μ) au territoire français. Ici une carte de la France joue le rôle de modèle du territoire français et un fichier XML joue le rôle de modèle de la carte. Dans ce contre exemple, le fichier XML n'est pas un métamodèle de la France. Un métamodèle n'est donc pas un modèle d'un modèle.

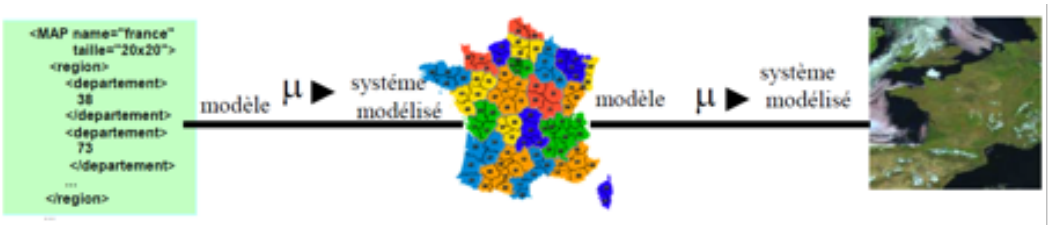

FIGURE 1.2 – Modèle de modèle selon l'exemple de la cartographie [FAVRE 2006]

Par ailleurs, le concept de métamodèle induit la deuxième relation fondamentale de l'IDM liant un modèle à son métamodèle. Cette relation est nommée *ConformeÀ* et notée χ [Bézivin 2004a] [Favre 2004]. La figure 1.3 reprend l'exemple de la cartographie où la légende de la carte joue le rôle de métamodèle (χ) pour une carte de la France. En effet, pour être lisible, la carte doit être conforme à la légende.

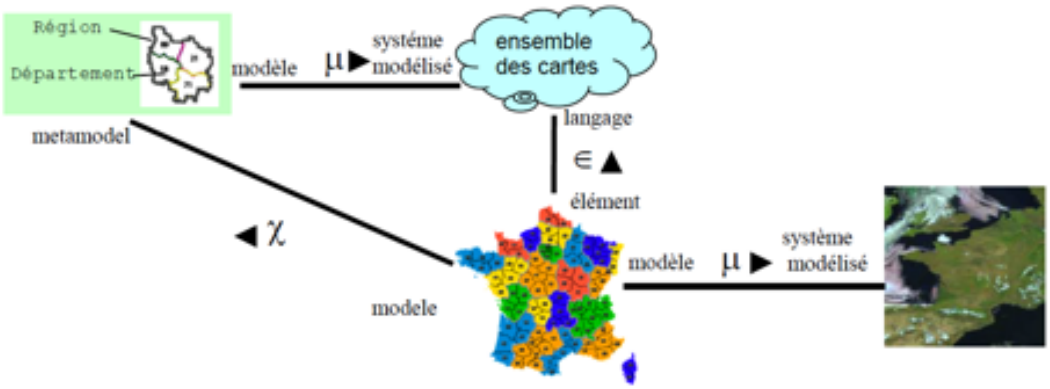

FIGURE 1.3 – Relations entre système, modèles, métamodèle et langage de modélisation [FAVRE 2006]

1.2 Transformation de modèle

Dans la partie précédente, nous avons introduit les concepts fondamentaux de l'IDM que représentent la notion de modèle et la relation *ReprésentationDe* ainsi que la notion de métamodèle et la relation *ConformeÀ*. Comme expliqué, la préoccupation majeure de l'IDM est de rendre les modèles opérationnels sur tout le cycle de vie des systèmes logiciels, depuis l'analyse et la conception jusqu'à la maintenance et l'évolution. Ainsi, la transformation de modèle se retrouve au cœur de l'IDM car c'est à travers elle que se fait l'automatisation des traitements apportés aux modèles. Nous allons d'abord donner une définition de la notion de transformation de modèle puis en présenter les types et les usages.

1.2.1 Définition de la transformation de modèle

L'OMG définit une transformation de modèle comme « le processus consistant à convertir un modèle en un autre modèle d'un même système » [OMG 2011].

[Kleppe 2003] proposent une définition moins générique en insistant sur l'aspect automatique de ce processus, ainsi, « une transformation de modèle consiste en la génération automatique d'un modèle source en un modèle cible, selon une description établie de cette transformation ». Cette définition implique aussi qu'une transformation est décrite à un plus haut niveau d'abstraction : au niveau d'un métamodèle auquel elle doit se conformer.

[Mens 2006] étendent cette définition en considérant qu'une transformation est une opération qui peut avoir en entrée un ou plusieurs modèles source et en sortie un ou plusieurs modèles cible :

Définition 3. Une transformation génère automatiquement un ou plusieurs modèles cible à partir d'un ou plusieurs modèles source, selon une description établie de la transformation.

C'est cette dernière définition que nous allons adopter dans ce document. Par ailleurs, notons que, si les métamodèles source et cible sont différents, la transformation est dite exogène. Si les métamodèles source et cible correspondent au même métamodèle, la transformation est dite endogène. Ces termes ont été introduits par [Mens 2006].

1.2.2 Composants d'une transformation de modèle

La figure 1.4 illustre les composants d'une transformation de modèle : les modèles source, les modèles cible, la définition de la transformation et le moteur qui va opérer la transformation selon sa définition.

La description de la transformation spécifie comment un ou plusieurs modèles source sont transformés en un ou plusieurs modèles cible. Elle est écrite dans un langage de transformation de modèle. Par exemple, si c'est un langage à base de règles, la description de la transformation consiste en un ensemble de règles de transformation à opérer [Kleppe 2003].

Un moteur de transformation exécute ou interprète la description. Il applique donc la description aux modèles source pour produire les modèles cible en suivant les étapes ci-dessous [Tratt 2005] :

- Identifier l'élément du ou des modèles source à transformer.
- Pour chaque élément identifié, produire l'élément cible qui lui est associé dans le ou les modèles cible.
- Produire une trace de la transformation qui lie les éléments du ou des modèles cibles aux éléments du ou des modèles source.

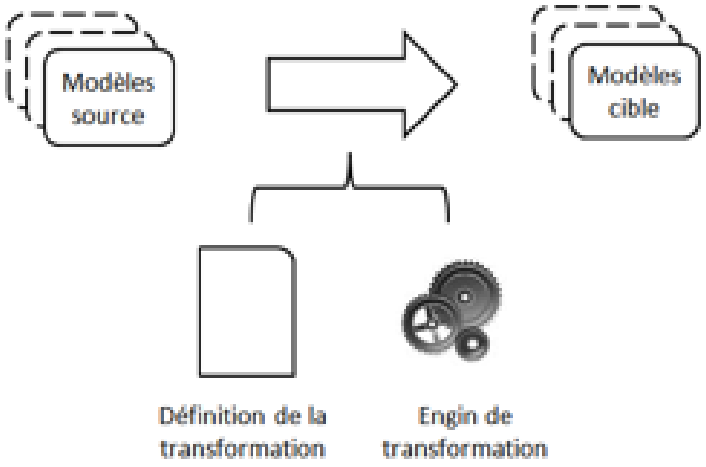

FIGURE 1.4 – Composants d'une transformation de modèle

1.2.3 Usages de la transformation de modèles

Les transformations de modèles sont au cœur d'une démarche dirigée par les modèles : elles permettent d'automatiser les manipulations subies par les modèles telles que la modification, la création, l'adaptation, la composition ou encore le filtrage de modèles, à travers la réutilisation systématique d'informations contenues dans les modèles existants.

Il est possible de recourir aux transformations de modèles sur tout le cycle de vie d'un système. Les usages les plus répondus sont le raffinement, l'intégration d'outils, la composition, l'analyse, la simulation et l'optimisation que nous présentons dans la suite de ce document.

1.2.3.1 Raffinement

Le raffinement consiste à rajouter plus de détails au modèle initial. Ce type de transformation peut aussi bien être endogène (métamodèles source et cible identique) ou exogène (métamodèle source et cible différents). Le raffinement se prête

parfaitement à toute la partie descendante du cycle en V où les modèles passent à des niveaux d'abstraction plus bas. Ceci revient à faire des transformations successives de type modèle-à-modèle et une transformation de type modèle-à-texte pour aboutir au code final.

Raffiner un modèle revient à décomposer des concepts de haut niveau, à choisir un algorithme particulier, à spécialiser un concept pour un contexte donné ou encore à le concrétiser sous forme d'une solution exécutable par une machine en générant le code à partir de modèles de plus haut niveau d'abstraction [Czarnecki 2000].

1.2.3.2 Intégration d'outil

Il existe une panoplie d'outils disponibles pour créer, manipuler, analyser ou encore simuler des modèles. Souvent ces outils utilisent des métamodèles internes et des espaces techniques qui leurs sont propres. Ainsi, l'échange de modèle entre ces outils est compromis et l'interopérabilité est fortement entravée. L'utilisateur se trouve obligé d'utiliser un seul et même outil sur tout le cycle de vie du système et ne peut donc pas tirer avantage des possibilités offertes par d'autres outils plus adaptés à ses besoins à certaines étapes.

L'intégration d'outil est une solution pour palier la divergence syntaxique et sémantique des outils et des langages de modélisation par le biais la transformation de modèle [Tratt 2005]. Ce type de transformation permet de naviguer entre deux métamodèles, de synchroniser des modèles qui évoluent séparément sur des outils distincts, de faire des mapping entre métamodèles pour maintenir la cohérence des modèles conformes à ces métamodèles. Il sera donc possible de faire appel à des outils mieux adaptés à chaque étape du cycle de vie.

1.2.3.3 Composition

Pour réduire la complexité inhérente à la modélisation et à l'analyse de grands systèmes, tels que les Smart Grids par exemple, il est possible d'adopter une approche par points de vue qui permet de séparer les préoccupations. Les modèles produits correspondent donc à ces différents points de vue qu'on peut ainsi valider séparément dans un premier temps. A l'issue de cette approche modulaire, on pourra composer ces modèles, c'est-à-dire les assembler, pour aboutir un modèle global du système.

Dans le cas le plus simple, les deux modèles à composer sont conformes à un même métamodèle. Cependant, il est aussi possible de composer deux modèles conformes à deux métamodèles différents.

Les deux modèles à composer peuvent aussi présenter des concepts en commun. Deux techniques existent pour composer des modèles, que nous illustrons dans la figure 1.5 :

- La première technique consiste à les fusionner. Dans ce cas, le modèle final résultant de la composition doit contenir toutes les informations issues des modèles initiaux, sans duplication des informations communes [Bézivin 2006a].

[Fleurey 2008] présente un framework générique capable de composer des modèles indépendamment de leurs langages de modélisation. L'approche consiste à identifier les éléments qui représentent le même concept dans les deux modèles à composer et à les fusionner dans un nouveau modèle qui représente une vue intégrée de ces concepts. Il est aussi possible de spécialiser le framework pour un métamodèle particulier mais qui reste conforme au MOF.

- La deuxième technique consiste à les tisser. Dans ce cas, on crée des correspondances entre les éléments qui représentent un même concept. Un métamodèle générique est créé pour définir les correspondances qui sont donc modélisées dans le modèle final. On y retrouve donc les éléments en commun dupliqués mais liés par un lien de correspondance.

Il est à noter que le modèle issu du tissage de deux modèles M_A et M_B peut être utilisé comme modèle intermédiaire que l'on note M_T pour la fusion de M_A et M_B. Dans ce cas L'opération de fusion consiste à produire un modèle M_{AB} en prenant comme entrée M_A, M_B et M_T. Cette technique est notamment utilisée par [Del Fabro 2007] pour la composition semi-automatique de modèles.

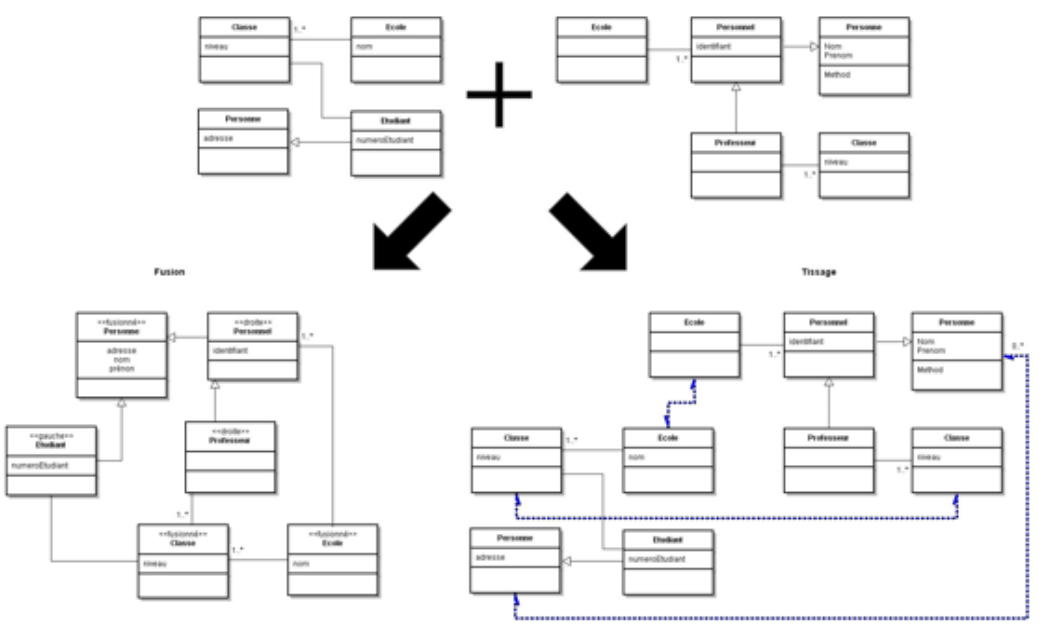

FIGURE 1.5 – Exemple de composition de deux modèles

1.2.3.4 Simulation

La transformation de modèle peut être utilisée pour simuler des modèles. En effet, une transformation de modèle peut mettre à jour le système modélisé. Dans ce cas, le modèle cible est une mise à jour du modèle source et la transformation est de type sur-place (modèles source et cible confondus).

Par exemple, [Syriani 2011] simule un comportement simple d'un jeu de Pacman en utilisant la transformation de modèle. La transformation spécifie les règles de transition qu'une instance du jeu peut prendre (Pacman et fantôme se trouvant dans la même case, Pacman et pomme se trouvant dans la même case, etc.). En ingénierie des langages, ceci revient à définir la sémantique opérationnelle d'un langage de modélisation. L'exécution de la transformation anime le modèle en fonction du comportement qu'on lui confère.

La transformation peut aussi être utilisée comme intermédiaire dans la simulation de modèle. Des modèles en entrée d'un outil de simulation externe sont produits par une transformation des modèles qu'on souhaite simuler. Cette technique permet de tirer profit d'outils de simulation existant sur le marché en utilisant l'intégration d'outils.

1.2.3.5 Analyse et optimisation

La transformation de modèle peut être utilisée pour les activités d'analyse de modèle. Une analyse simple telle que le calcul de métrique de similarité entre deux modèles via la transformation de modèle est donnée dans [Del Fabro 2007] avec un modèle de transformation écrit en ATL [Jouault 2006b]. Des analyses plus complexes sont possibles grâce à l'intégration d'outils d'analyse externes vers lesquels les modèles source sont transformés. [Biehl 2010] propose d'utiliser la transformation de modèle pour l'analyse de sûreté de fonctionnement dans le domaine de l'automobile. Les modèles source sont transformés en modèles conformes au métamodèle de l'outil d'analyse de sûreté de fonctionnement retenu.

L'optimisation vise à améliorer les propriétés non fonctionnelles des modèles telle que l'évolutivité, la fiabilité, la modularité, etc. L'optimisation est typiquement utilisée sur les modèles d'architecture. Les transformations utilisées pour l'optimisation sont de types endogènes car on cherche à affiner la conception de modèles existants. La réingénierie est un exemple de transformation utilisée pour optimiser les modèles : on cherche à améliorer la maintenabilité, la lisibilité et l'évolutivité des modèles.

1.2.4 Approches existantes pour la transformation de modèle

Le recours à la transformation de modèle est l'objet de recherches informatiques antérieures à l'apparition de l'approche IDM. Par exemple, les compilateurs utilisent la transformation pour passer du code source au fichier binaire [Aho 1985]. Ce type de transformation est restreint au domaine de la programmation informatique. La transformation de modèle embrasse un domaine plus large encore.

Nous trouvons dans la littérature plus d'une trentaine d'approches différentes de transformation de modèle [Syriani 2011]. Czarnecki et Helsen proposent une classification de ces approches selon plusieurs critères tels que le paradigme retenu pour définir la transformation, la relation entre les modèles sources et cibles, la directivité de la transformation, le nombre de modèles cible et source, l'orchestration et

l'ordonnancement des règles de transformation, etc. [Czarnecki 2006].

[Blanc 2011] retient trois grandes catégories d'approches :

- Par programmation

  Les modèles offrent une interface qui permet d'écrire les transformations dans un langage de programmation. Mais cette technique relève plus de la programmation que de la modélisation. Ce sont en fait des applications informatiques qui ont la particularité de manipuler des modèles. L'avantage de cette approche est que l'on utilise un langage de programmation généraliste tel que Java ou C++ pour écrire les transformations. Ainsi le programmeur n'a pas besoin d'apprendre un nouveau langage. Cependant ces applications ont tendance à devenir difficilement maintenables.

- Par template

  Dans cette approche on définit des canevas des modèles cibles. Ces modèles contiennent des paramètres qui seront remplacés par les informations contenues dans les modèles source. Ce type de transformation est souvent utilisé pour les transformations modèle-à-texte et est associé au visitor-pattern qui va traverser la structure interne du modèle source. Cette approche est utilisée par l'outil Enterprise Architect par exemple.

- Par modélisation

  Cette approche vise à appliquer les principes de l'IDM aux transformations de modèles elles-mêmes. Ainsi, on produit des modèles de transformation prennes, réutilisables et indépendants des plates-formes d'exécution [Bézivin 2006b]. Pour cela, on utilise des langages de modélisation dédiés à l'activité de transformation de modèles. Cette approche considère donc la transformation comme un modèle à part entière conforme à un métamodèle de transformation. La figure 1.6 , illustre cette approche en positionnant la transformation sur les différents niveaux d'abstraction de l'IDM. Elle corrobore ainsi la vision unificatrice de l'IDM à travers le paradigme du « tout est modèle » [Bézivin 2005]. Le langage de transformation ATL, que nous présentons dans la section 1.2.5.1, a été développé dans ce sens.

1.2.5 Langages et outils pour la transformation de modèle

Dans cette section, nous introduisons succinctement quelques langages et outils dédiés à la transformation de modèles, sans viser à l'exhaustivité.

1.2.5.1 ATL

Atlas Transformation Language (ATL) [Jouault 2006b] [Jouault 2008] est né de la volonté de proposer des langages de modélisation dédiés à la transformation de modèle en définissant un métamodèle et des outils pour l'exécution des transformations. Il permet de réaliser des transformations de type modèle-à-modèle et de type modèle-à-texte.

FIGURE 1.6 – Méta niveaux d'une transformation de modèle

ATL est un langage hybride (déclaratif et impératif) à base de règles OCL (OMG 2014). Une règle déclarative, appelée Matched rule, permet de décrire l'implémentation de mapping simples entre les modèles source et cible en utilisant des patrons source (*InPattern*) mappés avec les éléments source et des patrons cibles (*outPattern*) mappés avec les éléments cible.

L'approche impérative explicite les étapes d'exécution de la transformation à travers les Helpers. Ce mécanisme de Helpers permet en outre d'éviter la redondance de code et la création de longues règles écrites en OCL, ce qui confère une meilleure lisibilité aux programmes ATL.

Une transformation écrite en ATL est composée d'un ensemble de règles qui spécifient comment créer et initialiser les éléments des modèles cible. Il n'est pas possible de spécifier l'ordre d'exécution des règles de transformation. Cet ordre est établi automatiquement, exception faite pour les *lazy rules* qui ont besoin qu'on fasse spécifiquement appel à elles. ATL est conforme au méta-métamodèle MOF et est doté d'une syntaxe concrète textuelle. Il est intégré à l'environnement Eclipse. Une transformation prend en entrée un ensemble de modèles conformes à Ecore (EMF 2014) ou KM3 [Jouault 2006a].

ATL ne prend pas en charge les transformations incrémentales. Il commence par lire entièrement les modèles source et génère des modèles cible complet. Les modifications manuelles dans les modèles cible ne sont donc pas préservées si l'on opère une nouvelle transformation.

ATL peut réaliser des transformations sur place, c'est-à-dire, une transformation où le modèle source et le modèle source sont confondus en utilisant le mode raffinement de modèle. Cependant ce mode présente quelques limitations avec certaines fonctionnalités comme celle des *lazy rules*.

1.2.5.2 QVT

Le framework Query View Transformation (QVT) [Kurtev 2008] [OMG 2011] a rejoint la batterie de standards de l'OMG. Le métamodèle de QVT est conforme au MOF. Comme ATL, QVT se base sur OCL pour accéder aux éléments des modèles. QVT définit trois langages de transformation de type modèle-à-modèle. QVT-Relations (QVT-R) et QVT-Core (QVT-C) sont des langages déclaratifs qui adressent deux niveaux d'abstraction différents. QVT-Operational Mappings (QVT-OM) est un langage impératif qui étend QVT-R et QVT-C.

QVT-R est un langage de transformation de haut niveau d'abstraction doté de syntaxes concrètes textuelle et graphique. Les transformations, bidirectionnelles, sont spécifiées sous forme de relations entre les modèles source et cible. Une transformation a pour but de vérifier la cohérence entre deux modèles, renforcer la cohérence en modifiant le modèle cible, synchroniser deux modèles ou encore pour raffiner un modèle par une transformation sur-place. La sémantique de QVT-R est définie par une transformation vers QVT-C.

QVT-C est un langage de transformation de bas niveau qui sert de base pour QVT-R. Les deux ont le même niveau d'expressivité. Une transformation consiste en la déclaration de mapping entre les métamodèles source et cible en utilisant des patterns. Contrairement à QVT-R, la traçabilité est explicitement définie à travers les liens entre les métamodèles.

QVT-OM est un langage de transformation impératif qui étend QVT-R avec des constructions impératives basée sur une extension impérative de OCL. Les transformations sont unidirectionnelles mais établissent explicitement des modèles de traçabilité.

QVT est aussi doté d'un mécanisme de *blackbox* qui permet de faire appel à des algorithmes complexes écrits dans n'importe quel langage de programmation et d'utiliser des librairies existantes. Mais ce mécanisme rend la transformation opaque puisqu'il n'est pas contrôlé par le moteur d'exécution. Nous pouvons citer SmartQVT ou encore ModelMorf comme machines d'exécution de transformation écrite en QVT.

1.2.5.3 Kermeta

Kermeta est un langage généraliste de méta-modélisation exécutable et de méta-programmation orientée objet qui peut aussi décrire des transformations de modèle. Intégré à EMF, il est doté d'un métamodèle conforme au MOF qu'il étend avec un langage d'action impératif utilisé pour écrire le corps des opérations définies sur les concepts d'une syntaxe abstraite (ce qui revient à doter une syntaxe abstraite d'une sémantique opérationnelle). On peut ainsi décrire n'importe quel traitement sur un modèle ce qui est assimilé à une transformation de modèle.

Le langage d'action de Kermeta permet d'écrire des expressions impératives qui spécifient explicitement la construction des éléments des modèles cible. A l'inverse de QVT-OM, Kermeta n'est pas un langage à base de règles. Kermeta est capable de gérer les exceptions mais les transformations multidirectionnelles ne sont pas

supportées par les outils d'exécution. Il en est de même pour la transformation incrémentale. Les modèles source sont lus en une seule fois et les modèles cible sont produits complets lors de l'exécution de la transformation.

Annexe A

Exemple d'annexe

A.1 Exemple d'annexe

.

Bibliographie

[Aho 1985] Alfred V Aho, Ravi Sethi et Jeffry D Ullman. *Compilers Principles, Techniques, and Tools Addison-Wesley, 1986*. QA76, vol. 76, page C65A37, 1985. (Cité en page 8.)

[Atkinson 2003] Colin Atkinson et Thomas Kühne. *Model-driven development : a metamodeling foundation*. Software, IEEE, vol. 20, no. 5, pages 36–41, 2003. (Cité en page 2.)

[Bézivin 2001] Jean Bézivin et Olivier Gerbé. *Towards a precise definition of the OMG/MDA framework*. In Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on, pages 273–280. IEEE, 2001. (Cité en pages 1 et 2.)

[Bézivin 2004a] Jean Bézivin. *In search of a basic principle for model driven engineering*. Novatica Journal, Special Issue, vol. 5, no. 2, pages 21–24, 2004. (Cité en pages 1, 2 et 3.)

[Bézivin 2004b] Jean Bézivin, Mireille Blay, Mokrane Bouzhegoub, Jacky Estublier, Jean-Marie Favre, Sébastien Gérard et Jean Marc Jézéquel. *Rapport de Synthèse de l ?AS CNRS sur le MDA (Model Driven Architecture)*. CNRS, novembre, 2004. (Cité en pages 1 et 2.)

[Bézivin 2005] Jean Bézivin. *On the unification power of models*. Software & Systems Modeling, vol. 4, no. 2, pages 171–188, 2005. (Cité en pages 1 et 9.)

[Bézivin 2006a] Jean Bézivin, Salim Bouzitouna, Marcos Didonet Del Fabro, Marie-Pierre Gervais, Frédéric Jouault, Dimitrios Kolovos, Ivan Kurtev et Richard F Paige. *A canonical scheme for model composition*. In Model Driven Architecture–Foundations and Applications, pages 346–360. Springer, 2006. (Cité en page 6.)

[Bézivin 2006b] Jean Bézivin, Fabian Büttner, Martin Gogolla, Frédéric Jouault, Ivan Kurtev et Arne Lindow. *Model transformations ? transformation models !* In Model driven engineering languages and systems, pages 440–453. Springer, 2006. (Cité en page 9.)

[Biehl 2010] Matthias Biehl, Chen DeJiu et Martin Törngren. *Integrating safety analysis into the model-based development toolchain of automotive embedded systems*. In ACM Sigplan Notices, volume 45, pages 125–132. ACM, 2010. (Cité en page 8.)

[Blanc 2011] Xavier Blanc et Olivier Salvatori. Mda en action : Ingénierie logicielle guidée par les modèles. Editions Eyrolles, 2011. (Cité en page 9.)

[Booch 2004] Grady Booch, Alan W Brown, Sridhar Iyengar, James Rumbaugh et Bran Selic. *An MDA manifesto*. Business Process Trends/MDA Journal, 2004. (Cité en page 1.)

[Czarnecki 2000] Krzysztof Czarnecki et Ulrich W Eisenecker. *Intentional programming.* Generative Programming. Methods, Tools, and Applications, 2000. (Cité en page 6.)

[Czarnecki 2006] Krzysztof Czarnecki et Simon Helsen. *Feature-based survey of model transformation approaches.* IBM Systems Journal, vol. 45, no. 3, pages 621–645, 2006. (Cité en page 9.)

[De Lara 2002] Juan De Lara et Hans Vangheluwe. *Using Meta-Modelling and Graph Grammars to Process GPSS Models.* In ESM, pages 100–107, 2002. (Cité en page 1.)

[Del Fabro 2007] Marcos Didonet Del Fabro et Patrick Valduriez. *Semi-automatic model integration using matching transformations and weaving models.* In Proceedings of the 2007 ACM symposium on Applied computing, pages 963–970. ACM, 2007. (Cité en pages 7 et 8.)

[Favre 2004] Jean-Marie Favre. *Towards a basic theory to model model driven engineering.* In 3rd Workshop in Software Model Engineering, WiSME, pages 262–271. Citeseer, 2004. (Cité en page 3.)

[FAVRE 2006] Jean-Marie FAVRE, Jacky ESTUBLIER et Mireille BLAY-FORNARINO. *L'ingénierie dirigée par les modèles. Au-delà du MDA (Traité IC2, série Informatique et Systèmes d'Information)*, 2006. (Cité en pages 2 et 3.)

[Fleurey 2008] Franck Fleurey, Benoit Baudry, Robert France et Sudipto Ghosh. *A generic approach for automatic model composition.* In Models in Software Engineering, pages 7–15. Springer, 2008. (Cité en page 7.)

[Greenfield 2004] Jack Greenfield, Keith Short, Steve Cook, Stuart Kent et John Crupi. *Software factories : assembling applications with patterns, models, frameworks, and tools.* 2004. (Cité en page 1.)

[Jouault 2006a] Frédéric Jouault et Jean Bézivin. *KM3 : a DSL for Metamodel Specification.* In Formal Methods for Open Object-Based Distributed Systems, pages 171–185. Springer, 2006. (Cité en page 10.)

[Jouault 2006b] Frédéric Jouault et Ivan Kurtev. *Transforming models with ATL.* In satellite events at the MoDELS 2005 Conference, pages 128–138. Springer, 2006. (Cité en pages 8 et 9.)

[Jouault 2008] Frédéric Jouault, Freddy Allilaire, Jean Bézivin et Ivan Kurtev. *ATL : A model transformation tool.* Science of computer programming, vol. 72, no. 1, pages 31–39, 2008. (Cité en page 9.)

[Kent 2002] Stuart Kent. *Model driven engineering.* In Integrated formal methods, pages 286–298. Springer, 2002. (Cité en page 1.)

[Kleppe 2003] Anneke G Kleppe, Jos B Warmer et Wim Bast. Mda explained : the model driven architecture : practice and promise. Addison-Wesley Professional, 2003. (Cité en page 4.)

[Kurtev 2008] Ivan Kurtev. *State of the art of QVT : A model transformation language standard*. In Applications of graph transformations with industrial relevance, pages 377–393. Springer, 2008. (Cité en page 11.)

[Mens 2006] Tom Mens et Pieter Van Gorp. *A taxonomy of model transformation*. Electronic Notes in Theoretical Computer Science, vol. 152, pages 125–142, 2006. (Cité en page 4.)

[Minsky 1967] Marvin L Minsky. Computation : finite and infinite machines. Prentice-Hall, Inc., 1967. (Cité en page 2.)

[OMG 2011] QVT OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1. 1*, 2011. (Cité en pages 4 et 11.)

[Seidewitz 2003] Ed Seidewitz. *What models mean*. IEEE software, no. 5, pages 26–32, 2003. (Cité en page 2.)

[Syriani 2011] Eugene Syriani. *A multi-paradigm foundation for model transformation language engineering*. PhD thesis, McGill University, 2011. (Cité en page 8.)

[Tratt 2005] Laurence Tratt. *Model transformations and tool integration*. Software & Systems Modeling, vol. 4, no. 2, pages 112–122, 2005. (Cité en pages 5 et 6.)