L’ingénierie des modèles

Jean-Marc Jézéquel
professeur à l’université de Rennes 1
responsable du projet Triskell à l’INRIA/Irisa.

La dernière décennie a vu se poursuivre l'accroissement exponentiel de la complexité des systèmes informatiques, avec par exemple l'apparition de la problématique des lignes de produits logiciels, visant à minimiser simultanément les coûts de développement et le temps de mise sur le marché de nouveaux logiciels d'une même famille. Comme dans les autres sciences, on s'est de plus en plus appuyé sur la *modélisation* pour essayer de maîtriser cette complexité, tant pour produire le logiciel (conception) que pour le valider (test).

La modélisation, au sens large, est en effet l'utilisation efficace d'une représentation simplifiée d'un aspect de la réalité pour un objectif donné. Loin de se réduire à l’expression d’une solution à un niveau d’abstraction plus élevé que le code, la modélisation en informatique peut-être vue comme la séparation des différents besoins fonctionnels et préoccupations extra-fonctionnelles (telles que sécurité, fiabilité, efficacité, performance, ponctualité, flexibilité, etc.) issus des exigences. Réciproquement la conception du logiciel consiste à fusionner (ou tisser) des solutions à ces différentes préoccupations dans du code. Ce processus n’est en aucun cas nouveau : bien qu’on utilise de nouveaux noms, ces activités d’abstraction/modélisation et conception/tissage sont le pain quotidien des développeurs depuis toujours. Cependant dans la plupart des cas, tant les modèles que les solutions de conceptions restent implicites, ou tout au moins informels (par exemples si l’on fait usage de patrons de conception), et sont appliqués manuellement. Le véritable challenge dans un contexte de lignes de produits est de pouvoir décliner des gammes, c'est-à-dire pour chaque aspect changer facilement quelle version de quelle variante particulière est sélectionnée. Dans ce cas, refaire à chaque fois le complexe processus de tissage de la conception devient beaucoup trop cher, trop lent et trop sujet à erreurs.

Ce que propose l’approche de l’ingénierie des modèles (IDM, ou MDE en anglais pour Model Driven Engineering) est simplement de mécaniser le processus que les ingénieurs expérimentés suivent à la main. L’intérêt pour l’IDM a été fortement amplifié à la fin du 20^{ième} siècle lorsque l'organisme de standardisation OMG (Object Modeling Group) a rendu publique son initiative MDA (Model Driven Architecture), qui peut être vue comme une restriction de l’IDM à la gestion de l’aspect particulier de dépendance d’un logiciel à une plateforme d’exécution. De manière plus générale, et comme de nombreuses autres approches de génie logiciel comme la programmation générative, la programmation par aspects, les usines à logiciel (Software Factories) où encore le MIC (Model Integrated Computing), l’IDM est une forme d’ingénierie générative, qui se singularise par une démarche par laquelle tout ou partie d’une application informatique est générée à partir de modèles.

Pour cela il faut bien sûr que tant les modèles que les processus de tissage de la conception soient rendus explicites, et soient suffisamment précis pour être interprétés ou transformés par des machines. Le processus de conception peut alors être vu comme un ensemble de transformations de modèles partiellement ordonné, chaque transformation prenant des modèles en entrée et produisant des modèles en sortie, jusqu'à obtention d'artéfacts exécutables. Ainsi, quand on doit dériver un nouveau produit, qu'il soit simple évolution d'un produit existant ou nouvelle variante, on peut se contenter de « rejouer » automatiquement la plus grande partie du processus de conception, en changeant simplement quelques détails ici et là.

Du point de vue de la vérification et la validation, l'intérêt de l'IDM est d'offrir des solutions conceptuelles et technologiques pour homogénéiser les processus de spécification et modélisation du système d'une part et ceux liés à la validation d'autre part. Actuellement, il existe en effet une rupture entre ces deux processus car les modèles du système sont exprimés dans des langages différents de ceux utilisés par les techniques de validation. Du point de vue conceptuel l'IDM offre une solution en proposant une manière uniforme de décrire les domaines et leurs outils spécifiques : la métamodélisation. Les langages utilisés pour décrire les modèles du système et les modèles permettant de valider le système sont différents, cependant ces langages peuvent être décrits de la même manière: avec un métamodèle. Cette manière uniforme de décrire des langages (grâce à un métamodèle) facilite le passage d'un langage à un autre par des transformations de modèles. Du point de vue technologique, l'IDM permet de concevoir, implémenter et exécuter des transformations d'un langage vers un autre. La notion de transformation de modèles est donc un atout majeur de l'IDM pour intégrer les activités de vérification et validation dans le cycle de développement. Par exemple, ces transformations permettent de traduire automatiquement des modèles UML vers des automates ou d'autres langages intermédiaires qui sont requis en entrée des outils de vérification automatique ou de génération de test.

Dans le contexte de l'IDM, la notion de transformation de modèle joue un rôle fondamental ; aussi de nombreux outils, tant commerciaux que dans le monde de l'open source sont aujourd'hui disponibles pour faire la transformation de modèles. On peut grossièrement distinguer quatre catégories d'outils :

1. les outils de transformation génériques qui peuvent être utilisées pour faire de la transformation de modèles
2. les facilités de type langages de scripts intégrés à la plupart des ateliers de génie logiciel
3. les outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standards.
4. les outils de méta-modélisation " pur jus " dans lesquels la transformation de modèles revient à l'exécution d'un méta-programme.

Dans la première catégorie on trouve notamment d'une part les outils de la famille XML, comme XSLT ou Xquery, et d'autre part les outils de transformation de graphes (la plupart du temps issus du monde académique). Les premiers ont l'avantage d'être déjà

largement utilisés dans le monde XML, ce qui leur a permis d'atteindre un certain niveau de maturité. En revanche, l'expérience montre que ce type de langage est assez mal adapté pour des transformations de modèles complexes (c'est-à-dire allant au-delà des problématiques de transcodage syntaxique), car ils ne permettent pas de travailler au niveau de la sémantique des modèles manipulés mais simplement à celui d'un arbre couvrant le graphe de la syntaxe abstraite du modèle ce qui impose de nombreuses contorsions qui rendent rapidement ce type de transformation de modèles complexes à élaborer, à valider et surtout à maintenir sur de longues périodes.

Dans la seconde catégorie, on va trouver une famille d'outils de transformation de modèles proposés par des vendeurs d'ateliers de génie logiciel. Par exemple, l'outil Arcstyler de Interactive Objects propose la notion de MDA-Cartridge qui encapsule une transformation de modèles écrite en JPython (langage de script construit à l'aide de Python et de Java). Ces MDA-Cartridges peuvent être configurées et combinées avant d'être exécutées par un interpréteur dédié. L'outil propose aussi une interface graphique pour définir de nouvelles MDA-Cartridges. Dans cette catégorie on trouve aussi l'outil Objecteering de Objecteering Software (filiale de Softeam), qui propose un langage de script pour la transformation de modèles appelé J, ou encore OptimalJ de Compuware qui utilise le langage TPL, et bien d'autres encore, y compris dans le monde de l'open source avec des outils comme Fujaba (From UML to Java and Back Again),
L'intérêt de cette catégorie d'outils de transformation de modèles et d'une part leur relative maturité (car certains d'entre eux comme le J d'Objecteering sont développés depuis plus d'une décennie) et d'autres pas leur excellente intégration dans l'atelier de génie logiciel qui les héberge. Leur principal inconvénient est le revers de la médaille de cette intégration poussée : il s'agit la plupart du temps de langages de transformation de modèles propriétaires sur lesquels il peut être risqué de miser sur le long terme. De plus, historiquement ces langages de transformation de modèles ont été conçus comme des ajouts au départ marginaux à l'atelier de génie logiciel qui les héberge. Même s'ils ont aujourd'hui pris de l'importance, ils ne sont toujours pas vus comme les outils centraux qu'ils devraient être dans une véritable ingénierie dirigée par les modèles. De nouveau ces langages montrent leur limitation lorsque les transformations de modèles deviennent complexes et qu'il faut les gérer, les faire évoluer et les maintenir sur de longues périodes.

La troisième catégorie regroupe les outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standards. On va y trouver par exemple Mia-Transformation de Mia-Software qui est un outil qui exécute des transformations de modèles prenant en charge différents formats d'entrée et de sortie (XMI, tout autre format de fichiers, API, dépôt). Les transformations sont exprimées comme des règles d'inférence semi-déclaratives (dans le sens où il n'y a pas de mécanisme de type programmation logique), qui peuvent être enrichies en utilisant des scripts Java pour des services additionnels tels que la manipulation de chaînes. PathMATE de Pathfinder Solutions est un autre environnement configurable de transformation de modèles qui s'intègre particulièrement bien avec les ateliers de modélisation UML dominant le marché. Dans le monde académique on va trouver de nombreux projets open source s'inscrivant dans cette approche : les outils ATL de l'INRIA, AndroMDA, BOTL

(Bidirectional Object oriented Transformation Language),Coral (Toolkit to create/edit/transform new models/modeling languages at run-time), Mod-Transf (XML and ruled based transformation language), QVTEclipse (une implantation préliminaire de quelques unes des idées du standard QVT dans Eclipse) ou encore UMT-QVT (UML Model Transformation Tool). Ces langages spécifiquement conçus pour la transformation de modèle ont l'avantage de permettre d'exprimer très simplement les transformations de modèles simples (comme par exemple des transcodages syntaxiques), mais ils trouvent rapidement leurs limites lorsque les transformations de modèles deviennent complexes du point de vue algorithmique, ou bien lorsqu'il faut gérer de nombreuses variantes (contexte des lignes de produits) et les faire évoluer et les maintenir sur de longues périodes.

La dernière catégorie d'outils de transformation de modèles est celle des outils de méta-modélisation. La transformation de modèles revient alors à l'exécution d'un méta-programme orienté objet, pour lequel il est relativement aisé de construire un framework facilitant la gestion et la maintenance de nombreuses variantes de transformations nécessaires au contexte des lignes de produits. Le plus ancien et le plus connu de ces outils est certainement MetaEdit+ de MetaCase. Celui-ci permet de définir explicitement un métamodèle, et au même niveau de programmer tous les outils nécessaires, allant des éditeurs graphiques aux générateurs de code en passant par des transformations de modèles. Dans une veine similaire, XMF-Mosaic de Xactium est un environnement complet de définition de langage. Au coeur de XMF-Mosaic on trouve un noyau exécutable de définition de langage (qui est d'ailleurs auto-définissant). Tout est modélisé sur cette base, que ce soit les langages (e.g. UML), les outils, les GUI, parsers et autres analyseurs XML, mais à la différence de l'environnement MetaEdit+ obtenu par génération plutôt que par instantiaiton. Le langage de méta-modélisation de base est un langage orienté objet qui a toute la puissance d'un langage de programmation complet ce qui en fait un outil particulièrement puissant pour la transformation de modèles. Dans le monde de l'open source on trouvera dans cette catégorie l'environnement KerMeta (www.kermeta.org) développé à l'INRIA qui se présente comme l'ajout d'un aspect d'exécutabilité dans le MOF par un mécanisme astucieux inspiré du tissage d'aspects lui permettant de garder une compatibilité totale avec les environnements MOF standard (par exemple EMF).

Kermeta est aujourd'hui un des logiciels de l'INRIA les plus téléchargés sur la forge gforge.inria.fr. Kermeta est utilisé par l'INRIA et ses partenaires académiques ou industriels dans le monde entier, pour répondre à une variété de besoins, dont :

- Edition de méta-modèles EMOF décrits sous forme textuelle dans l'environnement Eclipse (un méta-modèle EMOF étant exactement équivalent à un programme Kermeta contenant uniquement des déclarations).
- Ajout de contraintes de « bonne formation » (exprimés en OCL) aux méta-modèles EMOF, contraintes qui seront vérifiées par l'interpréteur Kermeta lors du chargement d'un modèle.

- Ingénierie des langages, incluant la spécification homogène de la syntaxe abstraite, de la syntaxe concrète, de la sémantique statique et de la sémantique dynamique d'un langage.
- Simulation de modèles et de méta-modèles.
- Frameworks de transformation de modèle pour gérer des lignes de produits
- Tissage d'aspects au niveau modèle.
- Ingénierie des modèles en général

Même s'il existe une longue expérience de l'utilisation de l'ingénierie des modèles dans certains domaines comme les télécoms, sa généralisation à l'ensemble de l'industrie n'en est qu'à ses débuts. Visant à automatiser une partie du processus du développement, elle requiert un effort d'abstraction plus important de la part des développeurs. En contrepartie, elle permet de conserver le savoir faire de conception proche des centres de décision, grâce aux économies d'échelle dues à l'automatisation.