MDSD, MDD ou MDA : quel outil choisir ?

Les enjeux des chaînes de Développement Logiciel Guidé par des Modèles (Model Driven [Software] Development – MDSD / MDD) résident dans une réduction des charges de codage autant que dans la garantie de la qualité du code produit.

Les gains, réels et pragmatiques, constatés sur les projets bénéficiant de ces approches sont de 10 à 20% en réalisation initiale et de 30 à 50% en maintenance. Encore faut-il pour cela avoir retenu un ensemble d'outils adaptés à son contexte et à ses besoins. Cinq questions clefs permettent de bien cadrer les offres du marché en la matière.

COMPRENDRE LES OFFRES MDSD EN 5 QUESTIONS

Quelles transformations ?

La chaîne MDSD a pour rôle de créer des codes sources à partir de modèles qui lui sont fournis en entrée. Ces modèles sont autant que possible indépendants de la solution technique cible retenue. Ce sont des Modèles Indépendants de la Plate-forme (PIM) dans la terminologie MDA (Model Driven Architecture), l'approche MDSD formalisée par l'OMG (1).La capacité de la chaîne MDSD à accepter en entrée différents types de modèles répondant à différents formalismes graphiques ou textuels est essentielle. Il s'agit, d'une part, de modèles exprimés dans un formalisme normalisé ou standardisé, comme UML, d'autre part, de modèles exprimés dans un formalisme spécifique à un domaine (Domain Specific Model - DSM).Pour exploiter ces modèles, la chaîne MDSD s'appuie sur deux catégories de transformations : les transformations modèles à texte (M2T) et les transformations modèles à modèles (M2M).Les transformations M2T sont dédiées à la production du code source (Java, C#, HTML, XML,...) à partir de modèles. Elles produisent les fichiers qui, en l'état ou moyennant compléments des développeurs, sont compilés ou assemblés pour construire l'application cible.

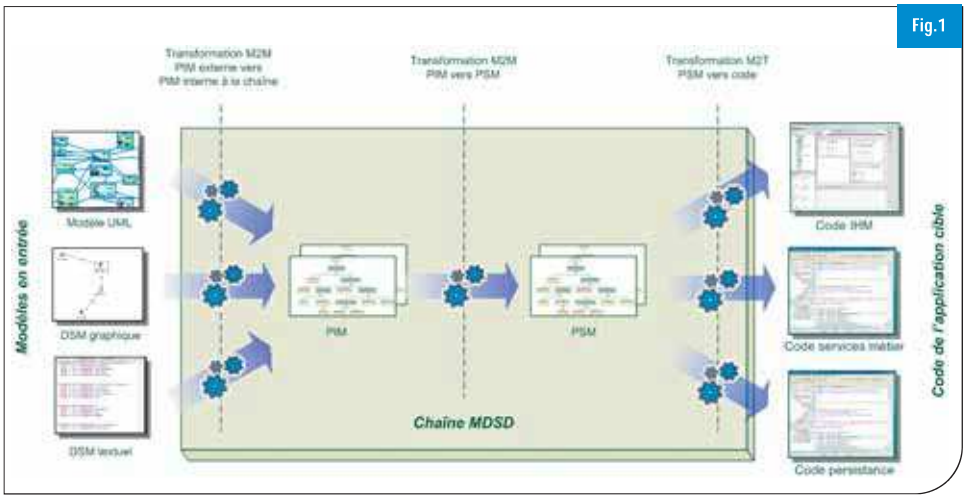
Les transformations M2M permettent de produire un ou plusieurs modèles à partir d'un ou plusieurs autres modèles. Ces transformations peuvent être chaînées. Une première transformation M2M peut par exemple être en charge des spécificités de l'architecture logique (services métiers, couche d'accès aux données,...). La transformation suivante pourra quant à elle, par exemple, introduire dans les modèles les spécificités techniques de la cible (JEE, .Net, PHP, Spring, JPA, ...). Les modèles obtenus à l'issue de cette transformation sont, dans la terminologie MDA, des Modèles Spécifiques à la Plate-forme (PSM).

Cette décomposition de la chaîne MDSD en plusieurs transformations apporte à la chaîne une grande flexibilité et une grande adaptabilité aux évolutions de la plate-forme cible (évolution de normes, changement de solution technique,...).

La figure 1 illustre les liens entre les différents types de modèles introduits ci-dessus et les transformations qui les lient.

Illustration d'une chaîne MDSD

Fig.1



Quel langage de transformation ?

Les outils de transformations M2T sont des générateurs de code. Ils exploitent des moteurs de templates. Le rôle de ces moteurs est de générer du code à partir, d'un côté, d'informations issues des modèles en entrée, de l'autre, de squelettes de code (" templates ") paramétrés. La partie dynamique des squelettes de code est décrite à l'aide d'un langage ad hoc, tel que le VTL (Velocity Template Language) pour le moteur Velocity.

Les outils du marché s'appuient soit sur des moteurs de templates propriétaires, open source ou non, soit sur des moteurs de templates standardisés par le marché, notamment Velocity, de la communauté Apache, ou JET (2) de la communauté Eclipse.

Les transformations M2M sont quant à elles assurées par des technologies

qui restent encore jeunes. Les moteurs de transformations que l'on retrouve dans les outils sont tous propriétaires. Ils exploitent des langages de transformations qui peuvent être propriétaires, ouverts tel ATL (3) ou normalisés tel QVT (4). Le choix des technologies de transformation supportées par l'outil MDSD a un impact sur la gestion des compétences des architectes responsables de la chaîne MDSD et sur la portabilité et la pérennité des transformations réalisées.

Quelle approche de génération ?

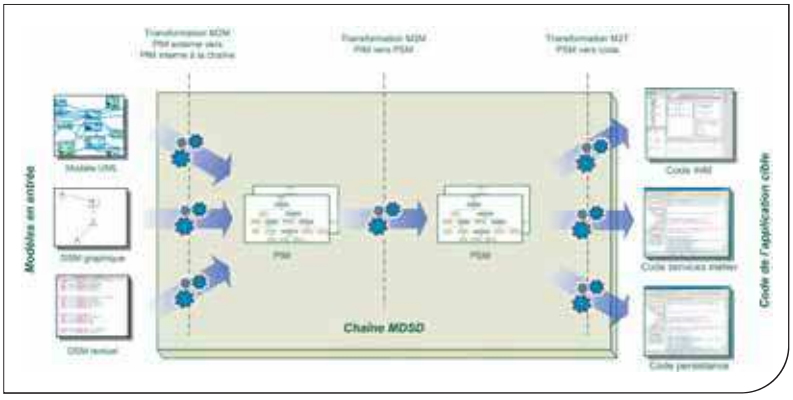
De façon générale, générer du texte, donc du code, est particulièrement aisé avec les technologies actuelles. Les difficultés résident dans le système de templating et dans la gestion de la génération incrémentale du code source. La qualité du système de templating détermine la facilité que les architectes ont à créer et à paramétrer les squelettes de code utilisés par le moteur de génération de code. La gestion incrémentale du code permet de générer à volonté différentes parties de la cible sans écraser les modifications effectuées manuellement par les utilisateurs.

La prise en compte du code des utilisateurs reste un des grands enjeux techniques des offres de génération de code. Face à ce constat, deux approches s'offrent à l'architecte. La première s'appuie sur la sélection d'une solution de génération ne sachant pas gérer le code utilisateur (solutions basées sur le moteur de templates Velocity par exemple) complété par une stratégie de génération de code intrusive. Cette stratégie nécessite la production de code respectant des motifs de conception (design pattern) qui permettent de protéger le code utilisateur. Elle est limitée aux technologies cibles supportant une forme de surcharge (redéfinition de méthodes héritées en langages objets) afin de permettre la génération d'une partie abstraite qui contient le code généré et d'une partie concrète, dite utilisateur, dans laquelle le développeur peut insérer son propre code. Cela exclut de facto les cibles telles que les fichiers HTML par exemple. Cette approche impacte la structure du code cible et peut en rendre la lecture plus complexe. Elle réduit cependant très fortement l'adhérence du code produit avec l'outil producteur et permet donc de faire évoluer ce dernier tout en maintenant la séparation code généré / code utilisateur.

La seconde s'appuie sur la sélection d'un outil gérant lui-même le contrôle du code utilisateur. Deux stratégies sont retenues par ces outils. La première consiste à placer des marqueurs dans le code. Ceux ci définissent des zones dans lesquelles l'utilisateur peut placer son code (JET, Acceleo, Xpand2). La seconde revient à maintenir un référentiel au sein de l'outil qui, pour chaque fichier, mémorise les modifications de l'utilisateur (Acceleo). Cette approche libère le code cible de toute considération liée à la manière dont il est produit (hors les marqueurs). Elle crée cependant une très forte adhérence du code produit avec l'outil producteur, ce dernier étant le seul à savoir quelles parties du code sont générées et quelles parties viennent de l'utilisateur.

Quelle intégration dans l'environnement de réalisation ?

Un outil de génération de code ne fonctionne jamais seul. Il est associé en amont avec un modeleur, en aval avec un environnement de développement. En la matière vous trouverez sur le marché des offres centrées sur un modeleur et intégrant les fonctions nécessaires de génération de code et des offres pure player MDSD centrées sur le générateur. La plupart de ces outils sont nativement intégrés ou intégrables à un environnement de développement du marché. En termes d'ingénierie logicielle, il est couramment attendu des concepteurs et ou des développeurs qu'ils pilotent les générateurs de code depuis des interfaces graphiques dédiées et produisent de fait automatiquement du code immédiatement disponible dans leur environnement de développement. Certains outils de génération ne disposent pas d'interface graphique. Leur déclenchement passe par la ligne de commande et donc, en pratique, par des scripts d'exécution. L'exécution par ligne de commande est certes moins ergonomique ; elle offre cependant l'opportunité d'insérer la chaîne MDSD dans une chaîne de construction et de packaging de l'application cible (chaîne Maven par exemple). De fait, la régé-



(1) OMG : Object Management Group, l'organisme international de normalisation des technologies objet.
(2) JET et Xpand : moteurs de templates du projet Eclipse M2T : http://www.eclipse.orsg/modeling/m2t/
(3) ATL : ATLAS Transformation Language, http://www.eclipse.org/m2m/atl/
(4) QVT : Query/View/Transformation, normalisé par l'OMG, http://www.omg.org/spec/QVT/1.0/

Le choix d'Auchan

Programmez ! : M. Lipka, pouvez vous nous dire deux mots sur votre société, sa DSI et votre fonction au sein de celle-ci ?

M. Lipka : Je suis Responsable de l'équipe Expertise & Outils dans le Département Ingénierie de développement. Ce Département est rattaché à la Direction Urbanisation & Service Transverse au sein de la Direction des Systèmes d'Information et de l'Organisation (DSIO). Nos missions, au sein de ce Département, sont de définir et standardiser des technologies de développement

P! : Dans quel contexte vous êtes vous lancé dans la sélection d'un outil MDD ?

M. Lipka : Sur le plan technique, nous travaillons depuis 2 ans environ sur l'évolution de nos socles techniques JEE : l'atelier de développement et de modélisation, le Framework se basant essentiellement sur des standards. Côté organisationnel, nous travaillons sur la mise en place d'un Centre de Développement pour nous aider à absorber le volume d'activité de nos développements.

P! : Quels étaient vos objectifs ?

M. Lipka : Nos objectifs principaux concernant la mise en place d'une solution MDD étaient de diminuer le temps en prise en main du Framework, d'automatiser la génération des couches techniques, de fiabiliser le développement des applications et enfin d'optimiser les abaques de développement.

P! : Quelle a été votre démarche de sélection d'une solution ?

M. Lipka : Nous avons fait un état de l'art des différentes solutions du marché et avons envoyé un appel d'offres à trois éditeurs nous paraissant représentatifs. Nous avons souhaité nous faire accompagner par un partenaire ayant une expérience réussie sur un environnement technique similaire au nôtre.

P! : Quels ont été vos critères de sélection d'une solution ?

M. Lipka : Nos critères étaient les suivants : facilité de maîtrise de la solution par nos équipes internes, pérennité, niveau de maturité des templates, activité de la communauté, sociétés partenaires maîtrisant la solution et enfin le retour sur investissement.

P! : Quels sont les principaux écueils que vous avez rencontrés ?

M. Lipka : Un marché en cours de stabilisation avec des normes et standards qui commencent à émerger Des éditeurs qui commencent à se positionner sur un outillage, mais peu de solutions packagées.

P! : Où en êtes vous dans votre démarche, quelles sont les prochaines étapes ?

M. Lipka : Nous avons fait le choix d'un éditeur et d'un partenaire nous aidant à mettre en place un prototype représentatif de nos besoins techniques. Le projet d'implémentation dépendra du résultat du prototype (gain en productivité, couverture technologique, ...)

nération systématique du code cible lors de la construction de l'application est l'une des solutions les plus directes et efficaces pour garantir que le code compilé, assemblé, déployé et qualifié est bien celui exprimé par les modèles.

Cartouches clefs en main ou à façon ?

Les éditeurs et les communautés, à la base des outils de génération dont nous parlons ici, ont développé une grande expertise dans l'art et la manière de produire du code automatiquement. Ils ne se contentent cependant pas de fournir le moteur de génération mais ils fournissent la plupart du temps des chaînes prêtes à l'usage. Elles offrent l'opportunité de démarrer très rapidement des projets de réalisation sans passer par une phase initiale de réalisation des templates et des transformations nécessaires. Elles présentent cependant deux risques significatifs. Le premier réside dans la divergence probable entre les choix d'architecture et d'implémentation retenus par l'éditeur pour les applications cibles et vos propres choix.

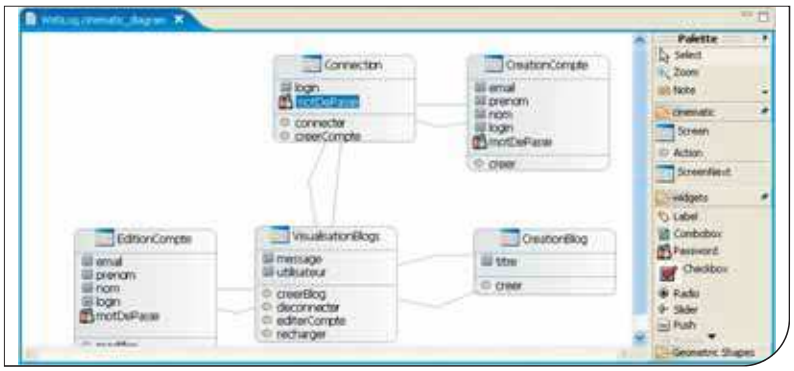
Le second risque réside dans la qualité des templates fournis. De fait, et nonobstant les compétences intrinsèques des équipes techniques des éditeurs ou des communautés, ces compétences ne sont pas nécessairement centrées sur les technologies cibles des générateurs, mais bien sur les technologies de génération ou de modélisation. En conséquence, en évitant soigneusement de verser dans le syndrome NIH (Not Invented here) il convient de se poser la question de l'exploitabilité et de l'exploitation des templates et transformations livrées avec les outils. Quelle que soit la stratégie retenue, cartouches clefs en main ou cartouches spécifiques, les facilités offertes par l'environnement de développement pour créer ou modifier ces cartouches restent primordiales pour les ajustements qui apparaîtront inévitables. Ces facilités peuvent s'appuyer sur des langages de transformations intrinsèquement flexibles (oAW) et/ou sur des outils et assistants efficaces. La capacité des outils à accepter des ajustements " locaux "aux projets, indépendamment des cartouches communes (Acceleo, oAW), représente également un réel facteur d'adaptabilité et de réactivité.

Conclusion

L'offre MDSD s'est fortement industrialisée, simplifiée et démocratisée ces quelques dernières années. Elle a entamé une structuration sur la base de normes et standards, appuyée en cela par les efforts croissants du marché et les communautés open source très actives. Ceux-ci se cristallisent tout particulièrement autour des travaux de la fondation Eclipse. La masse critique est désormais atteinte. Les approches MDSD ne sont plus réservées à une minorité très technique mais sont maintenant abordables et rentables pour tous.

■ Arnaud **Buisine**,
Directeur Technique, ProxiAD Nord, a.buisine@proxiad.com

■ Cédric **Vidal**,
Architecte Technique, expert JEE et MDSD, ProxiAD Ile de France, c.vidal@proxiad.com



Panorama des outils

Sur le marché de la modélisation il existe de nombreux outils pour l'UML et toutes les mouvances Model Driven. Nous vous proposons une sélection de solutions de modélisation commerciales et open source. L'offre est particulièrement importante aussi bien dans le monde open source que commercial. Les modeleurs UML sont les plus nombreux, même si aujourd'hui beaucoup d'outils mêlent UML, approche MD, génération de code, etc. Bref, le choix ne manque pas.

A vous de tester et de trouver l'outil idéal correspondant au mieux à vos besoins, vos contraintes projets. Si vos équipes ne sont pas " **Model Ready** ", une formation sera à envisager. Il s'agit d'un marché mouvant et les solutions naissent et disparaissent.

C'est notamment le cas d'OptimalJ, un des pionniers du MDA, qui a été arrêté par Compuware courant 2008. Mieux vaut opter pour des versions récentes.

Les solutions et outils UML

Nom de la solution	Licence	Commentaire
PowerAMC (Sybase)	commerciale	Outil historique, PowerAMC permet de modéliser et de désigner des processus d'entreprise, les données. Très complet et reconnu par le marché, il supporte UML et Merise. PowerAMC 15 est disponible depuis l'automne.
MagicDraw (No Magic Inc.)	Commerciale	Idéal pour le travail en équipe, les process métiers, le design objet, les systèmes, les données, autorise le round trip et supporte de nombreux langages. Très complet. Version 16 sortie en automne._
Win'Design (Cecima)	commerciale	Doté de 4 modules (database, object, business process et user interface), Win'Design est un modeleur multi usage (bpm, données, IHM, Merise, UML). Version 9 prévue courant janvier 2009.
UModel (Altova)	commerciale	Modeleur UML2 générant du Java, C#, VB.Net, avec reverse ingeniering, support modèle XML, XMI 2.1, intégration Eclipse, Visual Studio.
JViews (Ilog)	Commerciale	L'éditeur français intègre dans Jviews des éditeurs UML / EMF sous Eclipse pour modéliser les activités de l'entreprise plus facilement.
Visual Paradigm for UML (Visual Paradigm)	gratuite	Dans la version communautaire, l'outil propose les diagrammes de base, les flux de données, le process métiers et quelques possibilités de styles et de formatage des diagrammes.
Poseidon (gentleware)	Open source / commerciale	Modeleur UML comportant 9 diagrammes, la navigation, la génération de code Java, l'export des modèles. Installation simplifiée.
ArgoUML	Open source	Outil de modélisation permet de démarrer en douceur dans UML même si le projet est encore en développement !
Enterprise Architect (Sparx)	Commerciale	Modeleur UML 2.1 complet avec génération de code (java, .Net, Delphi, etc.). Supporte le modelage en équipe, XMI et dispose d'une API pour étendre l'outil.
Papyrus 4 UML (CEA)	Open source	Basé sur Eclipse, Papyrus est un modeleur UML 2 respectant DI2, extensible et acceptant les profils UML.
Umbrello	Open source	Modeleur pour KDE. Supporte différents langages dont PHP5, C#, Java, C++.
StarUML	Open source	Plate-forme UML et MDA. Extensible et fonctionnant sous Windows. Disponibilité de Template Designer.
JDeveloper (Oracle)	gratuite	Environnement d'Oracle. Propose un puissant modeleur UML réécrit dans en 11g. Supporte Java. Idéal pour concevoir des modèles de données.

Outils Model-Driven

Offre	Licence	Editeur	Type	Meta meta modèle	Transformation modèle à texte	Transformation modèle à modèle	Intégration IDE(1)	Modeleur
Acceleo	Open source	Obeo	Pure player MDSD	Ecore (2)	Propriétaire	non	Eclipse	ouvert
Acceleo Pro	Commerciale	Obeo	Pure player MDSD	Ecore	Propriétaire	Propriétaire (ATL annoncé)	Eclipse	ouvert
AndroMDA 3.x	Open source	-	Pure player MDSD	MOF (3)	Velocity	non	ouvert	MagicDraw Poseidon
ArcStyler	Commerciale	Interactive Objects	Pure player MDSD	MOF	Propriétaire	QVT	IBM RSM, MagicDraw	ouvert
Blu Age	Commerciale	Netfective Technology	Pure player MDSD	MOF & Ecore	JET	QVT (ATL annoncé)	Eclipse	ouvert
Eclipse M2T & M2M	Open source	Eclipse	Pure player MDSD	Ecore	JET	ATL, QVT	Eclipse	ouvert
MDWorkbench	Commerciale	Sodius	Pure player MDSD	Ecore	Propriétaire	Propriétaire & ATL	Eclipse	ouvert
Mia-Studio	Commerciale	Mia Software	Pure player MDSD	Ecore & MOF	Propriétaire	Propriétaire	Eclipse	ouvert
oAW	Open source	-	Pure player MDSD	Ecore	Xpand	Propriétaire (Xtend)	Eclipse	ouvert
RSM et RSA	Commerciale	IBM	Modeleur et IDE	Ecore	JET	Propriétaire	Eclipse/RSA	ouvert
Together Visual Studio	Commerciale	Borland	Modeleur et IDE	Ecore	Xpand , JET	QVT	Eclipse	ouvert
DSL Tools	Commerciale	Microsoft	Modeleur et IDE	Microsoft DSL	Propriétaire	non	Visual Studio	Visual Studio
Objecteering	Commerciale	Objecteering Software	Modeleur	n.c.	Propriétaire	Propriétaire	Eclipse Visual Studio	Objecteering

Ce tableau a été réalisé par Arnaud Buisine (ProxiAD)

(1) IDE : Integrated Development Environment – Environnement de Développement Intégré
(2) Ecore : MMM du Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/?project=emf
(3) MOF : Meta-Object Facility, MMM normalisé par l'OMG, http://www.omg.org/spec/MOF/2.0/

Quelques autres outils

Nom de la solution	Licence	Commentaire
CodeFluent (Softfluent)	commerciale	Génération des applications .Net par modèles qui sont des schémas structurés XML, avec choix des briques techniques. Un outil particulièrement puissant !
Leonardi	gratuite	Générer automatiquement les IHM de vos applications grâce à l'approche modèle MMI.
(W4 Lyria)		Supporte de nombreuses interfaces Java et Web.
Windev (PCSoft)	commerciale	L'atelier clé en main de PC Soft intègre des modules de modélisation Merise et UML pour faciliter le développement des applications !
ER/Studio (Embarcadero)	Commerciale	Environnement complet de modélisation de données. Supporte l'approche MD, le cycle de vie complet des données ou encore un module de qualité du design des bases.