



Maîtriser un développement guidé par le modèle dans une démarche projet

Les activités de gestion de configuration, de gestion de version, d'intégration, de validation ainsi que l'organisation des équipes et de la coopération sont souvent peu appréciées des équipes de développement mais restent cruciales pour la maîtrise d'un projet. C'est particulièrement sur ces points durs que sont expérimentées les limites des approches guidées par le modèle si les outils et méthodes ne sont pas adaptés.

Sil est relativement facile de présenter une solution produisant une grande quantité de code, impressionnante sur des cas d'étude, la confrontation avec la réalité des exigences des projets en termes de travail en équipe, de gestion de configuration et version nécessite des procédures et techniques opérationnelles et pragmatiques rarement effectives.

Nous allons prendre l'exemple du fonctionnement interne de l'équipe de développement du produit Modelio (société SOFTEAM) en Java et C++, utilisant le produit Modelio lui-même pour la modélisation et génération de code, pour illustrer un exemple de solution.

Comment gérer le travail coopératif d'une grande équipe sur des modèles de très grande taille ? Cette question est essentielle pour bénéficier des avantages de la modélisation et du développement guidé par le modèle. Outre des points techniques, liés aux capacités de l'atelier de modélisation

et aux outillages divers employés, le point organisationnel est fondamental, avec en particulier la nécessaire adhésion de l'ensemble de l'équipe de développement pour une bonne discipline collective.

Le succès ne peut être obtenu que s'il y a des avantages clairement perçus :

- Le développeur doit percevoir le gain, notamment au niveau de sa productivité, de son confort de travail et d'une coopération facilitée avec ses partenaires.
- Le projet doit obtenir un gain qualitatif clairement mesuré et perçu, sensible au niveau des responsables du projet, mais aussi des niveaux supérieurs de management.

Développement guidé par le modèle : les outils et techniques de productivité

La génération de code pratiquée selon deux modes : « model driven » ou « round trip »

L'approche guidée par le modèle doit être équilibrée par les avantages et inconvénients d'une orientation modèle ou d'une orientation code. Tout modéliser avant de coder peut s'avérer excessif dans le cas de petites classes techniques Java (par exemple), cependant qu'une approche centrée code aboutit à une production volumineuse de code, très productive avec des outils tels que Eclipse, mais souvent mal architecturée si on n'y prend garde.

Modelio permet ainsi deux modes de développement guidés par le modèle :

- « Model driven », où le modèle est élaboré, puis le code généré en

autorisant l'écriture du code des méthodes entre des balises dans le code Java. Modelio est donc maître, Eclipse (dans le cas Java) étant utilisé pour compléter le squelette de code produit. C'est en théorie le mode préconisé, car il garantit un respect de la conception faite au niveau du modèle.

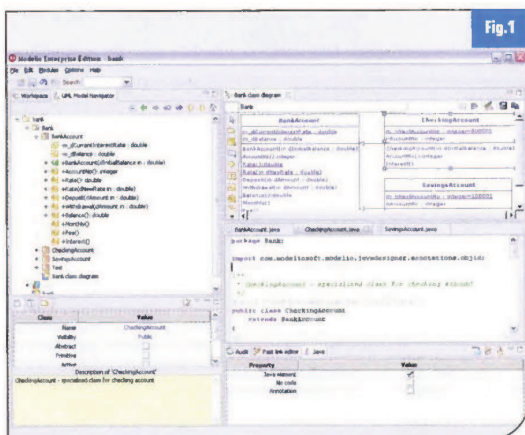
- « Roundtrip », où le code est retouché librement, et où Modelio resynchronise le modèle avec le code. Dans ce cas, Eclipse/Java est maître, et Modelio adapte le modèle au code. C'est un mode apprécié des développeurs, car il leur laisse toute la puissance de l'environnement Java [Fig.1].

Modelio conserve l'ensemble des informations - modèle et compléments code - dans son référentiel. A tout moment, l'ensemble du code peut être régénéré depuis Modelio. Ainsi, que l'approche soit « model driven » ou « roundtrip », la référence reste toujours le modèle, ce qui permet de se contenter de gérer le modèle en version, et non pas le code.

La modélisation, séparée en trois niveaux

Selon la phase de réalisation (analyse, conception, conception détaillée/codage), les parties de modèle mises en œuvre ne sont pas identiques. Dans les phases d'analyse, les modèles permettent de bâtir le dictionnaire de l'application, de définir ses exigences, de réaliser les Use Case, de construire des modèles conceptuels, et d'utiliser la totalité des capacités UML de modélisation.

Atelier Modelio, module Java designer - Modèle et code sont constamment synchronisés





La conception définit l'architecture générale en construisant des modèles productifs, essentiellement des modèles de classe, qui sont concentrés sur les grands principes et patterns architecturaux, et les classes essentielles du système. Le modèle de conception produit le code applicatif, soit en génération intégrale via les technologies MDA de Modelio, soit en génération de code en mode « model driven ».

La phase de codage/conception détaillée vient ensuite compléter le modèle de conception générale en mode « roundtrip », c'est-à-dire en utilisant massivement les capacités de l'environnement Eclipse/Java et en resynchronisant constamment le modèle.

La conception ne rentre pas dans le détail de classes Java trop spécifiques, typiquement dans le cas de construction IHM. Les classes principales de dialogue sont modélisées en conception, ainsi que les classes génériques IHM, puis générées, et le reste est fait sous Eclipse en réalisation puis reversé sous Modelio.

L'implémentation utilise le mode « roundtrip » plus flexible pour le programmeur, en préservant les classes de conception en mode model driven. Ainsi, l'analyste réalise des modèles pour une production essentiellement documentaire, ou pour reprise ultérieure en conception. Le concepteur réalise des modèles pour générer du code, qui doivent être pérennes et non remis en cause par le codage plus détaillé. Lors du codage plus détaillé sous Eclipse, le mode « model driven » n'autorise alors de retoucher et compléter que le corps des méthodes, balisé dans le source. Les modifications plus importantes doivent être faites sous l'atelier Modelio au niveau du modèle. Le programmeur intervient soit dans les modifications du code des opérations entre balises pour le mode « model driven », soit peut directement créer des classes techniques ou retoucher le code des classes sans restriction dans le mode « round trip ».

Un même intervenant peut jouer les différents rôles (analyste, concepteur, programmeur). En première itération

(Analyse/Conception/Codage), la phase de conception est clairement identifiée. Mais en phase de codage et dans les itérations suivantes, si la phase d'analyse reste bien séparée, il est cependant de la responsabilité de l'intervenant de choisir les modes « model driven » ou « round trip », selon son appréciation du niveau d'intervention (conception ou codage).

Tous les développeurs ont deux écrans sur leur station de travail : un écran est dédié au développement (Eclipse-Java ou Visual Studio-C++), et un écran est dédié à la modélisation (Modelio).

Les capacités MDA de génération dédiée à une architecture

Comme tout système informatique, l'application Modelio est appuyée sur une architecture dédiée. Les aspects systématiques et répétitifs de celle-ci donnent lieu à automatisation depuis une modélisation spécifique vers le code, afin qu'un principe architectural jugé optimal après de premières études et mises en œuvre soit systématiquement appliqué et automatisé. Outre une qualité garantie, une performance de développement démultipliée, cette approche permet lorsqu'un principe architectural évolue, de le réappliquer sur l'ensemble des cas concernés, et donc de moderniser toute l'application avec peu d'efforts.

Pour notre cible applicative, qui est l'atelier de modélisation Modelio lui-même, les transformateurs et générateurs MDA dédiés permettent à partir du métamodèle (définition par un diagramme de classes spécialisé du modèle supporté : UML, BPMN, modélisation des exigences, ...) de produire automatiquement : la gestion de la persistance de ce modèle, la construction de la vue « explorateur » du modèle, l'élaboration des boîtes de saisie des différents éléments du modèle, le contrôle de cohérence du modèle, la production de son interface de manipulation programmatique (API Java), et l'implémentation de différents mécanismes fonctionnels comme la gestion des verrous, les mécanismes de comparaison/fusion de modèles, etc.

Les éditeurs graphiques ne sont pas produits automatiquement, du fait de particularités importantes de chacun des cas, mais un « pattern » de modèle permet d'assister grandement leur programmation en permettant au développeur de se concentrer exclusivement sur les parties spécifiques à coder.

Parfois, de simples macros de manipulation de modèle, faciles à écrire en Python dans Modelio, permettent de traiter des « refactoring » massifs au niveau modèle avant régénération du code.

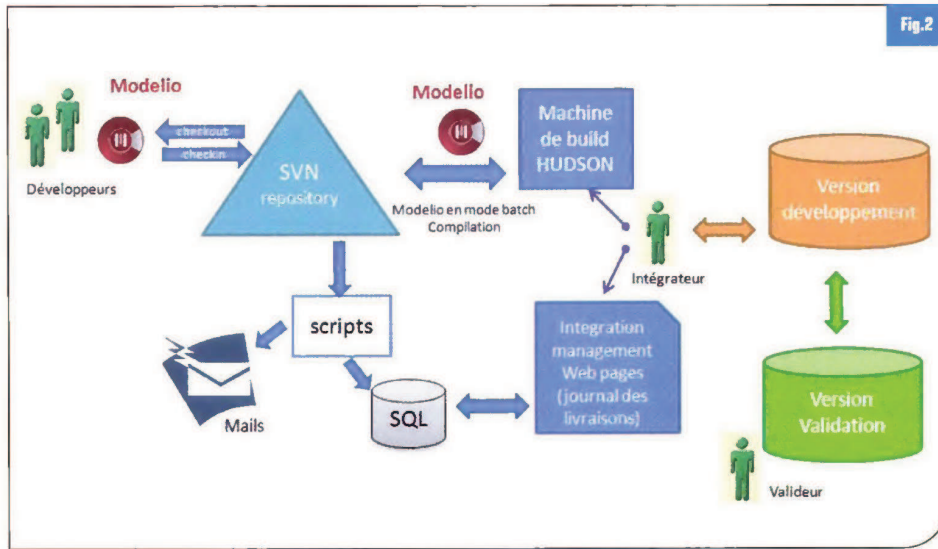
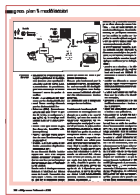
L'application de MDA permet soit de générer totalement le code à partir du modèle (ou du modèle plus compléments de code) via des transformations de modèle intermédiaires et des générations finales de code, soit de générer un modèle augmenté (par exemple via un pattern partiel) nécessitant un travail complémentaire de modélisation et codage. Pour une maîtrise des artéfacts gérés en configuration en phase de conception/codage, seul le modèle produisant du code est géré en configuration. Ainsi dans le deuxième cas, le seul modèle retenu et géré en configuration est le modèle augmenté, les étapes intermédiaires ne sont pas mémorisées, au même titre que l'on ne mémorise pas chaque action de modélisation ou codage en configuration.

Organisation du travail d'équipe

Plusieurs équipes contribuent au développement Modelio, réparties en plusieurs projets. Les développements utilisent des technologies variables selon les projets (C++ pour le cœur de l'atelier, Java pour l'interface homme/machine, projets MDA pour les modules Modelio). Une personne est dédiée à l'intégration, et une équipe séparée gère la validation [Fig.2].

Les outils utilisés ont été configurés et couplés à Modelio :

- L'atelier Modelio : Modélisation, génération de code [www.modelio.fr]
- Les modules Modelio Java Designer, C++ Designer, MDA Designer, et Teamwork manager



Organisation d'un projet

- Des extensions Modelio dédiées au support du processus de développement (modules Java spécifiques) : intégration avec les autres outils, automatisation de procédures, génération automatique vers notre framework interne de gestion de modèles
- Eclipse : (www.eclipse.org) outil open source pour les développements Java
- Visual Studio : outillage Microsoft pour les développements C++
- Subversion (SVN) : (www.subversion.tigris.org) outil open source de gestion de version et configuration. Il est utilisé via le couplage natif Modelio fourni par le module teamwork.
- Mantis : outil open source de gestion d'anomalie. Application web. (www.mantisbt.org)
- Testlink : outil open source de gestion des tests (<http://sourceforge.net/projects/testlink/>)
- Hudson : outil open source de production automatisée d'applications. (<http://hudson-ci.org/>)

Au sein d'un projet Modelio, les développements sont très couplés (chaque développeur effectuant plusieurs « commit » par jour) et nécessitent une gestion des conflits d'accès sur les modèles.

Tous les outils de développement sont déployés à l'identique sur chaque poste de travail. L'ensemble des outils, appelé « Toolkit » est versionné, géré par le responsable intégration sous la supervision du chef

produit qui décide des mises à jour chez les développeurs.

C'est un point fondamental pour le succès de l'ensemble. De cette façon, les environnements de développement sont homogènes, et les développeurs peuvent facilement intervenir dans les développements d'autres personnes.

Les procédures liées à l'intégration sont grandement facilitées.

Procédures de travail coopératif

L'ensemble du développement est découpé en « projets » (au sens de Modelio), qui sont des unités de coopération homogènes, sur lesquelles coopère une équipe d'au maximum 6 personnes. Le projet contient le référentiel du modèle et du code, et fait l'objet d'une gestion de version et configuration dédiée. Les « composants de modèle » supportés par l'atelier Modelio sont utilisés pour gérer les livraisons entre projets différents. Seul le modèle est géré en configuration, le code pouvant sans cesse être reconstitué à partir du modèle. Un référentiel SVN, directement alimenté par Modelio, est créé par projet. Ceci évite tout problème de taille, simplifie la gestion des branches et évolutions, permet des cycles de vie indépendants pour chaque projet, et permet de configurer chaque projet individuellement (modules, versions, composants de modèles, autorisations...).

La gestion de version et configuration

s'effectue à la granularité d'un package ou d'une classe (ou acteur, Use Case, ...) qui est l'unité minimale de travail d'un développeur. Avec l'atelier Modelio, les participants réservent sur les modèles les parties sur lesquelles ils travaillent (check out), puis livrent et libèrent celles-ci selon leur avancement (commit, check-in).

Le travail coopératif nécessite, outre les outillages Modelio Teamwork et Subversion, une discipline des participants. Un concepteur/développeur a les responsabilités suivantes lorsqu'il réalise ou modifie une partie de l'application :

- prendre en « checkout » les éléments du modèle qu'il doit modifier (ceci est imposé par Modelio), en minimisant le nombre d'éléments verrouillés,
- modifier ces éléments en respectant les règles méthodologiques citées précédemment (model driven versus roundtrip),
- « livrer » avec Modelio ses modifications pour les mettre à la disposition de l'équipe,
- garantir la « compilabilité » de ses modifications avant leur « livraison » ou « publication »,
- tester ses modifications avant leur « livraison » ou « publication »,
- documenter sa livraison (ce qui a été modifié, pourquoi, comment, impact sur la validation etc.)

Lorsqu'il livre ou publie des modifications, l'intervenant doit renseigner ce qu'il a fait. Ceci est assisté par l'outillage (extension MDA de Modelio Teamwork Manager), qui lui fournit un formulaire à remplir obligatoirement lors de chaque « check in » ou « commit » de ses modifications, avec des indications courtes du type : « Correction de l'anomalie 3765 » ou « Ajout de la fonction -move()- sur les objets graphiques ». Il doit aussi renseigner l'impact sur la validation : en quoi le comportement fonctionnel a été modifié (changement de commande, nouveau menu, ...). Les personnes concernées (développeurs impliqués, équipe d'intégration, de validation) sont averties par mail de l'évolution, qu'elles peuvent alors prendre en compte. Cette responsabilisation des acteurs dans le processus est essen-



tielle pour susciter leur adhésion au processus mis en œuvre. Elle leur permet d'ailleurs d'intervenir dans l'amélioration du processus.

En pratique, pour six intervenants simultanés sur un projet de 2800 classes, il y a peu de gêne occasionnée par le mécanisme de verrou. Le mécanisme de « add » de SVN, supporté par Modelio Teamwork Manager, permet en outre de créer de nouvelles parties de modèles au sein du modèle général, sans prendre de verrou tant que l'intervenant n'a pas livré ses premiers travaux.

Toutes les informations saisies dans le formulaire sont stockées dans une base de données relationnelle (MySQL). Ceci permet ensuite de faire des requêtes très utiles sur l'historique des travaux, typiquement pour savoir qui a changé quoi, pourquoi, et quelles sont les évolutions.

Lors d'une correction d'anomalie, le développeur doit accéder à l'outil de gestion des anomalies (Mantis), et mettre à jour le statut de celle-ci.

Cette correction fait également l'objet d'une notification aux autres intervenants concernés.

L'intégrateur joue un rôle clé d'orchestration dans le processus : Il voit la synthèse des livraisons. Les livraisons des concepteurs et développeurs sont automatiquement assemblées grâce à l'outillage Hudson, qui reconstruit en permanence l'application dans son état de développement/livraison le plus récent (intégration continue).

L'intégrateur supervise la construction permanente d'applications, et peut intervenir, notamment auprès des développeurs si le « build » n'est pas correct.

Il décide, avec l'assentiment du responsable de produit, de basculer des applications vers l'espace de livraison/validation. A ce stade, les livraisons sont versionnées et fournies à la validation. L'équipe de validation connaît, grâce à Mantis et à la base renseignant les livraisons, ce qu'il faut (re) valider sur cette nouvelle version.

Industrialisation du développement guidé par le modèle

Dans cet exemple, l'ensemble du développement guidé par le modèle a été industrialisé. Au-delà de la génération de code, la coopération de l'équipe en mode projet est outillée par des extensions à l'atelier Modelio, des outils additionnels, et est supportée par des procédures spécifiques.

Il est indispensable d'avoir le triptyque « processus/outillage adapté/adhésion et participation des développeurs ». C'est à cette condition que le développement guidé par le modèle devient opérationnel en pratique, pour des développements professionnels. Pour obtenir une description plus détaillée de l'organisation, un « livre blanc » est publié sur www.modelio.fr, où des démonstrations sont disponibles ainsi qu'une version gratuite de Modelio.

■ Philippe Desfray

SOFTEAM

philippe.desfray@softteam.fr