IDM 2009
Actes des $5^{èmes}$ journées sur l'Ingénierie Dirigée par les Modèles

Nancy, 25-26 mars 2009

17 mars 2009

Préface

La cinquième édition des journées sur l'Ingénierie Dirigée par les Modèles (IDM) rejoint cette année les conférences CAL (Conférence sur les Architectures Logicielles) et LMO (Langages et Modèles à Objets) à Nancy. Ce rapprochement est l'occasion de nouvelles interactions entre nos communautés pour répondre aux objectifs fondamentaux de l'IDM : maitriser la complexité du passage du problème au logiciel[1]. Depuis l'adoption par l'OMG de l'expression "Model Driven Architecture" en 2000, le développement dirigé par les modèles a été appliqué dans différents domaines (les systèmes critiques, les grands systèmes d'information, la production de portails web, etc) selon différents points de vue (la validation, le test, la production de code, la traçabilité, la sécurité, l'interopérabilité, etc). Avec la maturité des travaux en IDM vient le temps d'évaluer les solutions proposées. Les journées IDM qui visent à permettre aux chercheurs de présenter des travaux émergents, de discuter de nouvelles approches ou de retours d'expérience dans ce vaste domaine sont un lieu privilégié pour échanger sur ce thème. Dans le très riche programme des journées IDM 2009 nous lui avons accordé une part importante à la fois au travers des articles présentés, deux ateliers et la mise en place d'une table ronde sur le thème "difficultés industrielles en IDM".

Nous avons reçu 17 propositions d'articles. Parmi ceux-ci, nous avons retenu 9 articles en format long et 6 articles en format court qui présentent des travaux prospectifs. Les articles présentés lors de cette édition accordent une part importante à l'utilisation de l'IDM, en particulier dans le domaine des interactions hommes-machines où sont soulevés de nouveaux défis pour l'IDM. Nous abordons également le mariage entre l'IDM et les méthodologies de développement, la prise en compte du test par une démarche de développement dirigé par les modèles et les outils pour l'IDM. Un poster présente une solution à l'évolution logicielle des systèmes de recherche d'information utilisés pour la réalisation d'essais cliniques.
Deux ateliers partagés avec CAL et LMO nous permettront d'approfondir nos réflexions. Dans l'atelier Safe-Model dirigé par Daniel Deveaux et Isabelle Borne sera étudiée la combinaison entre l'IDM et les méthodes formelles pour la mise en place d'un processus de développement outillé intégrant les aspects spécification, vérification, implantation et la certification sur des architectures logicielles spécifiques. Dans l'atelier "Applications de l'IDM" dirigé par Xavier Lepallec nous nous intéresserons à trois études de cas sélectionnées pour les problèmes qu'elles soulèvent en IDM.
La table ronde dirigée par Joël Champeau réunira des industriels qui présenteront les difficultés rencontrées dans leur usage de l'IDM.

Nous remercions les membres du comité de programme et du comité d'organisation pour leur engagement dans la mise en place de ces journées, ainsi que les responsables d'ateliers et de table ronde. Plus largement, nous tenons à souligner l'implication du comité de pilotage des journées et les différents membres de l'action IDM qui par leur inventivité et leur dynamisme nous ont permis de proposer cette année une nouvelle approche des actes pour ces journées.

Nous souhaitons à toutes et tous un très bon IDM 2009 à Nancy,

Les éditeurs,

Mireille Blay-Fornarino et Olivier Zendra

[1] Robert France and Bernhard Rumpe. – Model-driven development of complex software : A research roadmap. – *In FOSE '07 : 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society

Ces journées ont été gérées en utilisant l'outil easychair.

Comité d'organisation

Président

Olivier Zendra, , INRIA Nancy - Grand Est / LORIA, France

Nacer Boudjlida, UHP
Anne-Lise Charbonnier, Service Colloques INRIA
Daniel Deveaux, VALORIA
Maha Idrissi Aouad, INRIA/UHP (Webmaster)
Amedeo Napoli, CNRS

Comité de programme

Présidente

Mireille Blay-Fornarino, I3S, Université de Nice-Sophia Antipolis, France

Olivier Barais, IRISA-INRIA, Rennes, France
Antoine Beugnard, TELECOM Bretagne, Brest, France
Isabelle Borne, Valoria, Vannes, France
Jean-Michel Bruel, IRIT, Université de Toulouse, France
Eric Cariou, LIUPPA, Université de Pau, France
Bernard Carré, LIFL, Lille, France
Joël Champeau, ENSIETA, Brest, France
Charles Consel, LABRI, ENSEIRB Bordeaux, France
Hubert Dubois, CEA LIST, Saclay, France
Sophie Dupuy-Chessa, LIG, Univeristé de Grenoble, France
Marie-Pierre Gervais, LIP6,Université de Paris 10, Nanterre, France
Regis Fleurquin, IRISA-INRIA, Rennes, France
Frederic Jouault, INRIA-EMN, Nantes, France
Xavier LePallec, LIFL, Lille, France
Yves Le Traon, ENST Bretagne - GET, France
Kim Mens, Université catholique de Louvain, Louvain-la-neuve, Belgique
Clémentine Nebut, LIRMM, Université de Montpellier, France
Marc Pantel, IRIT, Toulouse, France
Lukas Renggli, University of Bern, Suisse
Jean-Paul Rigault, I3S-INRIA, Université de Nice-Sophia Antipolis, France
Nicolas Rivierre, France Telecom R&D, MAPS/AMS Laboratory, France
Lionel Seinturier, LIFL-INRIA, Lille, France
Julie Vachon, DIRO, University of Montreal, Canada
Olivier Zendra, INRIA Nancy - Grand Est / LORIA, France

Autres relecteurs

Michel Dao, France Telecom R&D, MAPS/AMS Laboratory, France
Jean-Rémy Falleri, LIRMM, Université de Montpellier, France
Clémentine Nemo, I3S, Université de Nice-Sophia Antipolis, France
Anne-Marie Pinna-Dery, I3S, Université de Nice-Sophia Antipolis, France

Table des matières

Outils pour l'IDM **1**

Contrats de transformation pour la validation de raffinement de modèles,
Eric Cariou, Nicolas Belloir, Franck Barbier **1**

From UML Actions to Java,
Anis Charfi, Heiko Mueller, Andreas Roth, Axel Spriestersbach **17**

A Domain Specific Language for Expressing Model Matching,
Kelly Garcés, Frédéric Jouault, Pierre Cointe, Jean Bézivin **33**

Tests et Validation **49**

From Business Processes to Integration Testing,
Stéphane Debricon, Fabrice Bouquet, Bruno Legeard **49**

Démarche de développement à base de composant d'applications embarquées tolérantes aux fautes,
Mohamed Lamine Boukhenoufa, Brahim Hamid, Agnès Lanusse, A. Radermacher **65**

Sûreté de Fonctionnement dans un processus basé sur l'Ingénierie Dirigée par les Modèles,
Daniela Cancila, Hubert Dubois, Morayo Adedjouma **73**

Méthodologie **79**

Ingénierie des exigences par l'IDM, Assistance à l'ingénierie des systèmes complexes,
Eric Le Pors, Olivier Grisvard, Yvon Kermarrec **79**

Formalisation de bonnes pratiques dans les procédés de développement logiciels, ***Vincent Le Gloahec, Régis Fleurquin, Salah Sadou*** **95**

Ingénierie des modèles pendant la phase de spécification du besoin,
Benjamin Chevallereau, Alain Bernard, Pierre Mévellec **101**

Un processus d'imitation de patrons d'ingénierie supporté par l'approche IDM,
Nicolas Arnaud, Agnès Front, Dominique Rieu, Sophie Dupuy-Chessa **109**

Applications de l'IDM **115**

Correspondances et transformations actives dédiées aux IHM, ***Olivier Beaudoux, Arnaud Blouin, Slimane Hammoudi*** **115**

L'Ingénierie Dirigée par les Modèles au coeur d'un Framework d'aide à la composition d'interfaces utilisateurs,
Anne-Marie Dery, Audrey Occello, Philippe Renevier, Cedric Joffroy **131**

L'IDM pour la construction d'un éditeur de scénarios pédagogiques : vers un environnement intégré support au cycle de vie des scénarios,
Christian Martel, Laurence Vignollet, Salim Ouari , Christine Ferraris **147**

Industrial-strength Rule Interoperability using Model Driven Engineering,
Marcos Didonet Del Fabro, Patrick Albert, Jean Bézivin, Frédéric Jouault **163**

Model-based DSL Frameworks : A Simple Graphical Telecommunications Specific Modeling Language,
Vanea Chiprianov, Yvon Kermarrec, Patrick Alff, Martin Woods **179**

Contrôle guidé par l'IDM des évolutions d'un système de recherche d'informations clinique,
Valery Lopes, Eric Leclercq, Marie-Noëlle Terrasse **187**

Index des auteurs **189**

Contrats de transformation pour la validation de raffinement de modèles

Eric Cariou — Nicolas Belloir — Franck Barbier

Université de Pau et des Pays de l'Adour
LIUPPA
B.P. 1155, 64013 Pau Cedex France
{Eric.Cariou, Nicolas.Belloir, Franck.Barbier}@univ-pau.fr

RÉSUMÉ. *Un processus logiciel basé sur l'ingénierie des modèles est construit à partir d'un ensemble de transformations qui sont exécutées en séquence pour partir d'un niveau de modélisation abstrait et arriver au code final ou à une spécification d'implémentation détaillée. Ces transformations correspondent pour beaucoup à des raffinements, c'est-à-dire à de l'ajout de détails ou d'information sur un modèle. Ces raffinements peuvent être entièrement automatisés ou nécessiter l'intervention manuelle du concepteur. Nous proposons ici une méthode pour valider que le résultat d'une transformation, y compris lorsque le concepteur intervient manuellement sur les modèles, respecte la spécification d'un raffinement. Nous nous basons pour cela sur des contrats de transformations de modèles écrits en OCL afin de rendre notre méthode indépendante des outils de modélisation et de transformations de modèles.*

ABSTRACT. *A model-driven engineering process relies on a set of transformations which are sequentially executed, starting from an abstract level to produce code or a detailed implementation specification. These transformations are mostly refinements, that is to say detail or data added to models. These refinements may be entirely automated or may require manual intervention by designers. In this paper, we propose a method to demonstrate that a transformation result is correct with respect to the specification of the refinement. This method both includes automated transformations and manual interventions. For that, we focus on transformation contracts written in OCL. This leads to make the proposed method independent of modeling and transformation tools.*

MOTS-CLÉS : *IDM, contrats de transformation, raffinement, OCL*

KEYWORDS: *MDE, transformation contracts, refinement, OCL*

1. Introduction

Un processus logiciel basé sur l'ingénierie des modèles est construit à partir d'un ensemble de transformations qui sont exécutées en séquence pour partir d'un niveau de modélisation abstrait et arriver au code final ou à une spécification d'implémentation détaillée. Ces transformations correspondent pour beaucoup à des raffinements, c'est-à-dire à de l'ajout de détails ou d'information sur un modèle. Ces raffinements peuvent être entièrement automatisés ou nécessiter l'intervention manuelle du concepteur. En effet, même si le but principal de l'ingénierie des modèles est d'automatiser un processus logiciel complet, le concepteur doit souvent intervenir sur les modèles et faire des choix sur certaines actions à réaliser.

Il est important de pouvoir garantir que les transformations effectuées sont valides, c'est-à-dire qu'elles respectent la spécification du raffinement. Cela est d'autant plus important quand le concepteur intervient manuellement sur le modèle. Dans ce cas là, le modèle peut être grandement modifié sans contraintes particulières et il faut s'assurer que la modification reste dans le cadre de ce qui est attendu.

Nous proposons ici une méthode pour valider qu'une transformation, y compris quand le concepteur intervient manuellement, respecte la spécification d'un raffinement. Nous nous basons pour cela sur des contrats de transformations de modèles écrits en OCL (*Object Constraint Language* (OMG, 2006)) afin de rendre notre méthode indépendante des outils de modélisation et de transformation de modèles. La validation du contrat doit en effet pouvoir se faire à partir de deux modèles – l'un représentant le modèle source et l'autre le modèle cible de la transformation – générés ou obtenus à partir de n'importe quel outil. Notre méthode peut s'appliquer sur n'importe quelle transformation endogène et donc sur un raffinement car c'est un type particulier de transformation endogène. Dans cet article, nous nous intéressons plus particulièrement aux raffinements pour illustrer ces transformations endogènes.

La section suivante résume le concept de contrat de transformations de modèles. Ensuite, un exemple de raffinement de modèle et de son contrat est donné. La section 4 présente la méthode de définition et de validation d'un contrat, appliqué à notre exemple de raffinement. Enfin, avant la conclusion, nous parlons des travaux connexes.

2. Contrats de transformations de modèles

La programmation et conception par contrat (Meyer, 1992, Beugnard *et al.,* 1999) consiste à spécifier ce que fait un élément logiciel, programme ou modèle, afin de savoir comment l'utiliser et valider que l'on a bien obtenu le résultat attendu. En termes d'opérations, qu'il s'agisse d'une méthode d'un programme ou d'une classe d'un diagramme de classes, l'approche par contrat consiste à spécifier une pré et une post-condition. La pré-condition définit l'état du système à respecter pour que l'opération puisse être appelée et la post-condition, l'état que le système s'engage à respecter après l'appel. On peut également préciser un ensemble d'invariants que l'élément logiciel doit respecter en permanence.

Dans le contexte des transformations de modèles, nous avons proposé dans (Cariou *et al.,* 2004b, Cariou *et al.,* 2004a) d'appliquer ces principes à une opération de transformation de modèles et donc de définir des contrats de transformations. Ces contrats spécifient alors des transformations de modèles en établissant des contraintes sur l'état du modèle avant la transformation (modèle source) et l'état du modèle après la transformation (modèle cible). Ils servent à garantir qu'un modèle cible est bien le résultat valide d'une transformation par rapport à un modèle source, ou bien encore qu'un modèle source peut être transformé.

Un contrat de transformation est défini par trois ensembles de contraintes :

Contraintes sur le modèle source : contraintes à respecter par un modèle pour pouvoir être transformé

Contraintes sur le modèle cible : contraintes générales (indépendamment du modèle source) à respecter par un modèle pour qu'il soit le résultat valide de la transformation

Contraintes d'évolution d'éléments : contraintes à respecter sur l'évolution de certains éléments entre le modèle source et le modèle cible, pour que le modèle cible soit le résultat valide de la transformation par rapport au modèle source

Dans (Cariou *et al.,* 2004b, Cariou *et al.,* 2004a), nous avons établi les bases conceptuelles autour des contrats de transformations écrits en OCL. Dans cet article, nous validons en pratique cette approche via une méthode et une implémentation complète dans le contexte des transformations endogènes (c'est-à-dire à méta-modèle constant). Nous détaillons également des points que nous avions seulement abordés comme le problème essentiel de la correspondance entre éléments des modèles source et cible (voir la section 4.3.2).

Le choix d'OCL comme langage d'expression des contrats se justifie pour plusieurs raisons. La première est qu'OCL est par nature un langage d'expression de contraintes et qu'un contrat est un ensemble de contraintes. Ensuite, notre souci était de pouvoir définir un contrat le plus indépendamment possible des plates-formes et des outils. OCL est pour cela un bon candidat puisque c'est un standard utilisable sur des modèles de types variés, comme UML, le MOF ou bien encore pour la plate-forme Eclipse/EMF qui est une des principales plates-formes de développement d'outils d'ingénierie des modèles. On peut donc vérifier des contraintes OCL sur des modèles générés par une grande majorité des moteurs de transformation ou de manipulation de modèles. Enfin, OCL est un langage qui est relativement bien connu. Il est en effet difficile de définir un méta-modèle ou un modèle sans avoir besoin d'y ajouter des contraintes supplémentaires, écrites par exemple et bien souvent en OCL. OCL est également utilisé comme langage de requêtes ou est étendu dans un certain nombre de langages de transformations de modèles (par exemple ATL[1] ou le standard QVT (OMG, 2008)). (Pons *et al.,* 2006) argumente également dans le sens de l'utilisation d'OCL en expliquant qu'OCL offre l'avantage d'être plus accepté en pratique

1. Atlas Transformation Language : `http: //www.eclipse.org/m2m/atl/`

par les concepteurs que d'autres langages formels, à la syntaxe et à l'utilisation plus complexes ou moins connues.

3. Raffinement de modèles

Le raffinement de modèles consiste à modifier le contenu d'un modèle sans en changer le but ou la sémantique. Il s'agit de rajouter des détails à un modèle ou bien encore de le restructurer pour en améliorer la conception. Un exemple simple de raffinement est l'ajout de méthodes d'accès à un attribut. À un certain niveau d'abstraction, dans un diagramme de classes, on peut se contenter de spécifier la liste des attributs d'une classe. Quand on se rapproche de l'implémentation, on rajoutera pour chaque attribut une méthode de lecture et une d'écriture (un *getter* et un *setter*). Le sens du modèle reste le même, des détails ont juste été ajoutés.

La réalisation d'un raffinement consiste donc à prendre un modèle et à le modifier, ce qui correspond en fait à une transformation de modèle. Par principe, le raffinement implique que les deux modèles, celui d'avant le raffinement (le modèle source de la transformation) et celui d'après (le modèle cible), sont du même type, c'est-à-dire conformes au même méta-modèle. La transformation est donc endogène. De manière plus générale, le raffinement est un cas particulier de transformation endogène. Un raffinement étant une transformation, il peut donc être spécifié par un contrat de transformation.

À titre d'exemple, nous allons étudier dans cet article un raffinement d'ajout d'interfaces sur un diagramme de classes de type UML. Afin de simplifier nos explications et l'écriture du contrat associé, nous n'avons pas utilisé directement un diagramme de classes UML et donc le méta-modèle UML standard car ce dernier est relativement complexe en terme de nombre d'éléments et de navigation entre ces éléments. Nous avons défini un méta-modèle de diagramme de classes simplifié, comme détaillé dans la section suivante.

3.1. *Méta-modèle de diagramme de classes*

La figure 1 représente le méta-modèle de notre diagramme de classes. On y retrouve les mêmes concepts de base que dans un diagramme de classes UML : classe, interface, type, méthode, attribut, association ainsi que quatre types primitifs (Void, Boolean, Integer et String). Les concepts qui ne sont pas présents, par souci de simplification, sont par exemple les visibilités ou bien encore la spécialisation.

L'élément `ModelBase` est particulier et représente, comme son nom l'indique, la base du modèle. Il ne définit pas de concept du domaine mais est une aide à la manipulation du modèle dans certains outils en référençant l'ensemble des principaux éléments d'un modèle. Nous avons implémenté ce méta-modèle via la plate-forme Eclipse/EMF et son méta-méta-modèle Ecore. Dans ce contexte, il est recommandé, par exemple pour la génération automatique d'éditeurs de modèles, d'avoir un tel élé-

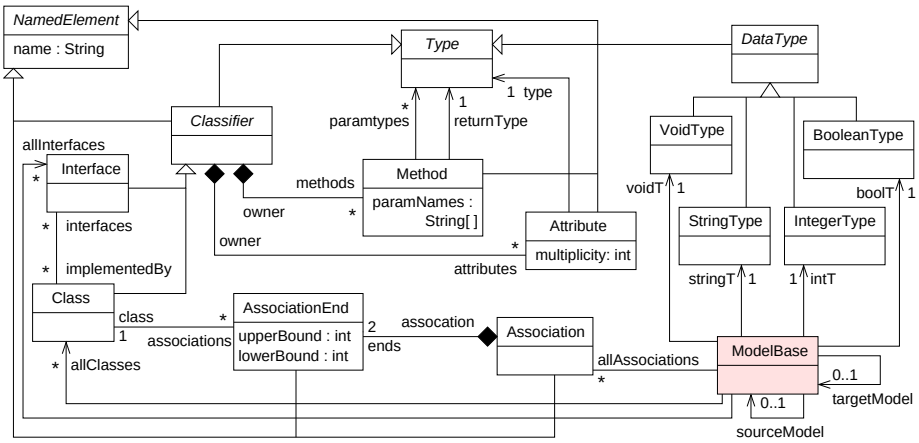


Figure 1. *Méta-modèle du diagramme de classes simplifié*

ment de base. De même, dans d'autres outils, tel que Kermeta[2], où la navigation sur le modèle se fait de manière explicite dans le code de la manipulation du modèle, une telle base de modèle est utile. Cette base référence donc l'ensemble des classes, interfaces et associations du modèle ainsi qu'une instance de chacun des quatre types primitifs. Les références `targetModel` et `sourceModel` serviront à la concaténation de deux modèles (voir la section 4.2).

Afin de compléter la définition du méta-modèle, au diagramme de classe de la figure 1 s'ajoute un ensemble de régles de bonne formation, exprimées sous la forme d'invariants OCL (ces règles ne sont pas présentées ici faute de place).

3.2. *Exemple de raffinement : ajout d'interfaces*

Le principe du raffinement que nous allons utiliser en exemple est de rajouter une ou plusieurs interfaces à chaque classe et de déplacer une partie ou toutes les méthodes implémentéees par cette classe dans ses interfaces. Ce raffinement se fait en deux étapes :

1) Ajout automatique d'une interface par défaut : pour chaque classe du diagramme, on crée (sauf si elle existait déjà) et lui associe une interface nommée `I`*`nomDeLaClasse`*`.` Toutes les méthodes implémentées directement par la classe sont ensuite déplacées dans cette interface. Cette opération est entièrement automatisable et est réalisée par un outil de transformation de modèles.

2. `http://www.kermeta.org/`

2) Réarrangement éventuel des méthodes et interfaces : l'ajout d'une interface par défaut ne convient pas forcément au concepteur. Il peut alors modifier manuellement pour chaque classe la liste des interfaces (renommage, suppression, ajout) ainsi que la localisation des méthodes (déplacement d'une méthode vers une autre interface ou la classe).

Au final, après ces deux étapes, on obtient un modèle cible qui est la transformation du modèle source et correspond à un raffinement d'ajout d'interfaces. La principale contrainte à vérifier pour que le raffinement soit correct est que chaque classe implémente toujours la même liste de méthodes, directement ou via ses interfaces. De manière détaillée, le contrat de transformation associé à ce raffinement est le suivant :

– Contraintes sur le modèle source : aucune, on peut raffiner un diagramme de classes quelconque

– Contraintes sur le modèle cible : chaque classe implémente au moins une interface (même si elle ne contient pas de méthodes)

– Contraintes d'évolution des éléments du modèle source vers le modèle cible : toutes les classes sont conservées[3] et chacune implémente, directement ou indirectement via ses interfaces, la même liste de méthodes qu'avant la transformation

La figure 2 montre un exemple d'un tel raffinement, avec ses deux étapes. On peut y noter que le concepteur a renommé les deux interfaces de la classe `Compte` et a replacé la méthode `getNom` directement dans la classe `Client`. On constate que le raffinement est correct puisque toutes les classes sont conservées, que chaque classe implémente toujours, directement ou indirectement, la même liste de méthodes et qu'elle implémente au moins une interface.

4. Méthode de validation de transformation par contrat

Dans cette section, nous allons détailler notre méthode de définition et de validation de contrats. Cette méthode est la plus générale possible et doit donc s'appliquer sur deux modèles, l'un supposé source et l'autre cible de la transformation ou du raffinement, définis ou obtenus à partir d'une grande variété d'outils.

4.1. *Discussion sur la forme du contrat*

Dans (Cariou *et al.,* 2004b), nous avons défini deux techniques pour écrire le contrat de transformation, et plus précisément les contraintes sur l'évolution des éléments entre le modèle source et le modèle cible. Ces contraintes sont particulières car elles nécessitent de référencer à la fois le modèle source et le modèle cible. Les

3. D'autres éléments du modèle doivent être également conservés sans modification, comme les associations entre les classes. Par souci de simplification, nous laisserons ces éléments de côté dans nos explications.

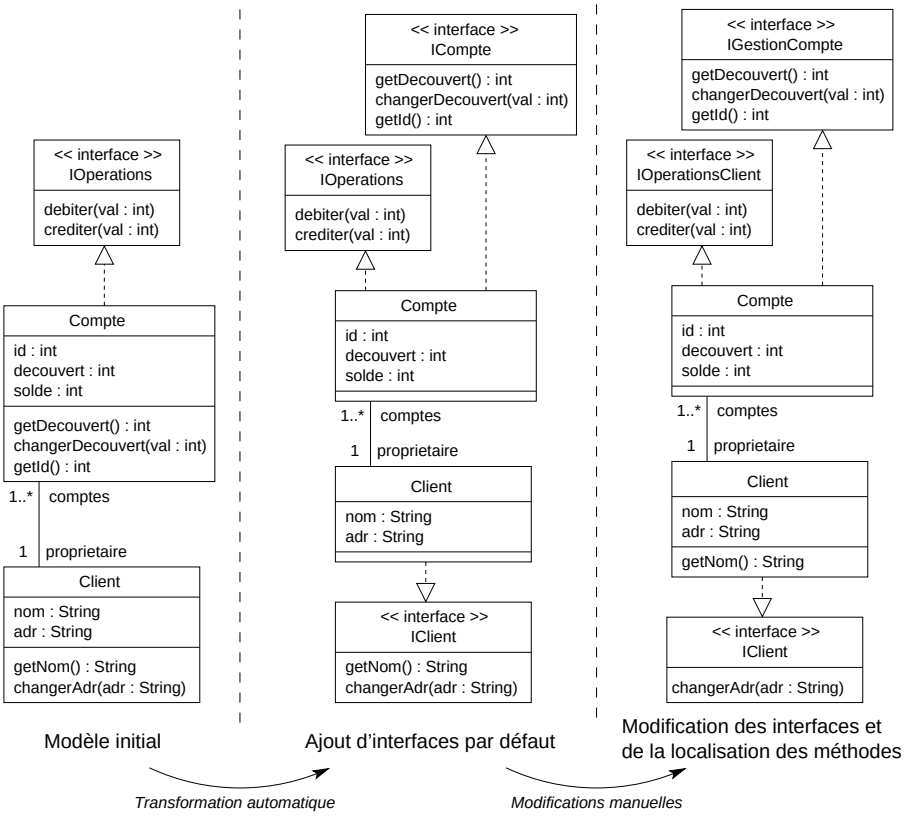


Figure 2. *Exemple de raffinement : ajout et modification d'interfaces*

contraintes OCL s'exprimant dans un contexte – et donc par rapport à un seul méta-modèle – il faut donc pouvoir manipuler les deux modèles à partir de ce seul contexte[4].

Pour réaliser cela, la première technique consiste à attacher l'opération de transformation à un élément du méta-modèle et à la spécifier en OCL : le modèle avant la transformation est le modèle source et une fois la transformation exécutée, le modèle modifié par l'opération est le modèle cible. Dans la post-condition, on peut alors référencer à la fois les éléments du modèle cible et ceux du modèle source via la construction `@pre` d'OCL. Au final, la pré-condition permet d'écrire la partie du contrat relative au modèle source et la post-condition contient les deux autres parties du contrat

4. Cela implique par défaut que le méta-modèle soit le même pour le modèle source et le modèle cible, d'où la restriction à des transformations endogènes. En pratique, les méta-modèles source et cible peuvent être relativement différents si l'on arrive à en déterminer un méta-modèle commun, comme expliqué dans (Cariou *et al.,* 2004a)

(sur le modèle cible et l'évolution des éléments). Pour notre exemple d'ajout d'interfaces, on pourrait avoir une écriture du contrat de la forme suivante :

```
context ModelBase : : addInterfaces()
pre : -- contraintes sur le modèle source
post :
    -- contraintes sur le modèle cible et
    -- contraintes d'évolution entre le modèle source et le modèle cible
    allClasses -> size() = allClasses@pre -> size() and ...
```

Ici, l'opération de transformation s'appelle `addInterfaces` et est attachée au méta-élément `ModelBase`. La post-condition de l'opération vérifie, entre autres, que le nombre de classes après la transformation est le même qu'avant. On voit que l'on manipule bien les éléments avant la transformation (modèle source) et les éléments après la transformation (modèle cible).

Si cette méthode est conceptuellement élégante car on associe un contrat à l'opération de transformation comme on associe un contrat en OCL à n'importe quelle opération d'une classe (via une pré et une post-condition) ou bien encore un contrat à une méthode d'un programme objet, elle n'est pas adaptée à notre problème, et cela pour deux raisons. La première est que ce contrat n'est vérifiable que lorsqu'on exécute la transformation. Cela implique qu'on utilise un outil qui permet à la fois d'exécuter une transformation et d'en vérifier les contraintes associées. Cela limite fortement les outils utilisables. On peut tout de même citer Kermeta qui permet d'associer des opérations aux méta-éléments et de leur associer une pré et une post-condition[5]. La seconde raison est la conséquence de la première : il faut exécuter une transformation pour valider son contrat, ce qui oblige à une exécution entièrement automatique. Si le concepteur modifie à la main le modèle généré, il n'est donc pas possible de valider le contrat.

4.2. *Concaténation des modèles source et cible*

Lors de la définition du contrat, pour éviter les problèmes que nous venons de décrire, nous allons utiliser la deuxième technique que nous avons définie. Le principe général est de concaténer ou de fusionner le modèle source et le modèle cible dans un modèle plus global. Ensuite, on définit en OCL un ensemble d'invariants s'appliquant sur le modèle global, et donc à la fois sur les éléments du modèle source et du modèle cible. Cette méthode est à adapter en fonction du type de modèle. Par exemple, dans le cadre d'un diagramme de classes UML, on définira un modèle simple contenant deux packages : le premier correspondra au diagramme de classes source et le second au diagramme de classes cible (on peut utiliser cette méthode du double package pour n'importe quel type de diagramme UML).

5. Kermeta utilise un langage dédié pour l'écriture des contraintes. On n'écrirait donc pas le contrat directement en OCL mais dans un langage à la sémantique et l'expressivité similaires.

Dans le cas de définition d'un méta-modèle dédié à un domaine, il est intéressant d'y ajouter directement les éléments nécessaires à la manipulation simultanée des deux modèles. Dans notre méta-modèle de diagramme de classes simplifié, nous avons par exemple rajouté dans la classe `ModelBase` deux références, nommées `sourceModel` et `targetModel` de type `ModelBase`. Ces deux références permettent de charger et référencer deux modèles de type diagramme de classes, à savoir un modèle source et un modèle cible. L'intérêt de rajouter ces références de cette manière permet de rester dans le contexte du même méta-modèle, ce qui simplifie l'écriture du contrat.

À partir de deux modèles, un source et un cible, il s'agit donc d'en créer un troisième. D'un point de vue pratique, pour l'implémentation en Ecore de notre méta-modèle, il suffit de créer un nouveau modèle et de positionner les références `sourceModel` et `targetModel` de la base de ce modèle avec les deux modèles à fusionner via les mécanismes standards d'instantiation et d'édition de modèles de la plateforme EMF. Une autre méthode consiste à passer par un petit programme dédié, par exemple écrit en Kermeta, qui prend en paramètre les noms des deux modèles et sauvegarde leur fusion dans un troisième modèle.

Ce type de programme dédié permet également d'effectuer le cas échéant des traitements supplémentaires. Dans notre exemple, les quatre types primitifs (Void, Boolean, Integer et String) sont gérés comme des singletons : la base du modèle référence une instance de chacun de ces types. Chaque élément du modèle devant utiliser un type primitif va utiliser cette référence unique. Le problème ici est, qu'après concaténation des modèles source et cible, on se retrouve avec trois bases de modèle (la base du modèle global, la base du modèle source et la base du modèle cible) et donc avec les types primitifs chacun présents en trois exemplaires. Pour revenir à une unique instance de chaque type, il faut parcourir le contenu des modèles cible et source et remplacer chaque référence vers un type primitif par la référence sur le type primitif de la base globale. Ainsi, on pourra facilement comparer que deux éléments de modèles différents utilisent le même type primitif via une simple égalité de référence.

4.3. *Écriture du contrat*

Comme nous l'avons vu, le contrat comporte trois parties : les contraintes que doit respecter le modèle source, les contraintes que doit respecter le modèle cible et les contraintes sur l'évolution des éléments du modèle source vers le cible. Les deux premières parties sont simples à exprimer, il s'agit de définir des invariants associés au méta-modèle, comme cela se fait de manière classique. Pour notre exemple d'ajout d'interface, le modèle source est un diagramme de classes quelconque donc il n'y a pas de contraintes particulières à appliquer. Pour le modèle cible, il s'agit d'un diagramme de classes dans lequel chaque classe implémente au moins une interface. Cela s'exprime de la manière suivante :

context Class **inv** hasAnInterface :
self.interfaces -> notEmpty()

Il suffit ensuite via un évaluateur OCL de vérifier que cette contrainte est validée par le modèle cible.

La troisième partie concernant les contraintes d'évolution entre les deux modèles est plus complexe à établir, parce qu'elle s'applique sur le modèle global qui concatène le modèle source et le modèle cible au sein d'un seul modèle. Il faut donc, pour commencer, déterminer explicitement la référence sur le modèle source et sur le modèle cible. Ensuite, il faut disposer d'un ensemble de fonctions utilitaires permettant de déterminer que tel ou tel élément du modèle cible a une correspondance dans le modèle source et de récupérer cet élément correspondant.

4.3.1. *Détermination des modèles source et cible dans le modèle global*

Selon la façon dont le modèle global est construit, récupérer les références sur le modèle source et le modèle cible peut être direct ou nécessiter une requête OCL. C'est ce que nous avons besoin de faire dans notre exemple et notre implémentation. En effet, notre modèle global contient trois bases de modèles (trois instances de `ModelBase`) et il faut déterminer laquelle correspond au modèle source et laquelle correspond au modèle cible. Cela peut être réalisé par la définition de variables OCL comme suit :

```
context ModelBase
def: base : ModelBase = ModelBase.allInstances() -> any ( base |
    not(base.sourceModel.oclIsUndefined()) and
    not(base.targetModel.oclIsUndefined()) )

def: sourceModel : ModelBase = base.sourceModel
def: targetModel : ModelBase = base.targetModel
```

L'idée appliquée ici est de récupérer la base globale et via ses références `sourceModel` et `targetModel` de récupérer alors les modèles source et cible. La base globale est en fait la seule qui a ses deux références `sourceModel` et `targetModel` positionnées. On fait donc une requête sur l'ensemble des bases pour récupérer celle qui respecte cette caractéristique.

4.3.2. *Correspondance entre éléments du modèle source et du modèle cible*

Dans notre exemple de raffinement, le but principal du contrat, concernant l'évolution des éléments entre le modèle source et le modèle cible, consiste à vérifier que chaque classe du modèle cible implémente (directement ou via ses interfaces) les mêmes méthodes que sa classe équivalente du modèle source. Pour valider cela, il est donc nécessaire de pouvoir déterminer quelle est sa classe équivalente. De manière plus générale, il est nécessaire de pouvoir déterminer que tel élément dans un modèle possède ou pas un élément du même type qui lui correspond dans l'autre modèle ainsi que de récupérer cet élément correspondant. Pour l'exemple de la figure 2, la classe `Compte` du modèle cible (modèle final) correspond dans le modèle source (modèle initial) à la classe `Compte`. Cette correspondance se vérifie en comparant les noms des classes et en vérifiant, entre autres, que leurs attributs (ici `id`, `decouvert` et

`solde`) sont les mêmes. Il est donc nécessaire de vérifier également la correspondance des attributs.

La correspondance d'un élément d'un modèle à un élément du même type d'un autre modèle peut être de trois natures :

Correspondance totale : un élément correspond à un élément identique de l'autre modèle, dans le sens où tous ses attributs et toutes ses références ont également une correspondance[6] dans l'autre modèle.

Par exemple, dans notre méta-modèle, un méta-élement attribut possède un nom, une référence sur un type et une référence sur son propriétaire. Deux instances d'attribut seront en correspondance totale si elles ont le même nom, si leurs types sont en correspondance et si leurs propriétaires sont également en correspondance.

Correspondance partielle : un élément possède un élément correspondant dans l'autre modèle dans le sens où une sous-partie de ses atttributs et de ses références ont également une correspondance[6] dans l'autre modèle.

Dans notre exemple, une classe est en correspondance partielle avec une classe de l'autre modèle. En effet, si le nom doit être le même et si la liste d'attributs doit être en correspondance pour les deux classes, il ne faut en revanche pas vérifier que les listes des méthodes ou des interfaces implémentées sont en correspondance. Ces dernières concernent les parties modifiées entre le modèle source et le modèle cible et leur validité est contrôlée à part.

Pas de correspondance : il n'est pas nécessaire de vérifier qu'un élément possède une correspondance dans l'autre modèle.

Pour notre exemple, il ne faut pas vérifier qu'une interface est en correspondance avec une interface de l'autre modèle. En effet, pendant le raffinement, la liste des interfaces change (renommage, création, suppression d'interface, modification de la liste des méthodes d'une interface) en fonction des choix du concepteur.

Ces correspondances entre éléments sont implémentées en OCL par une série de fonctions utilitaires (via le constructeur `def`). Selon la complexité du méta-modèle et le nombre de correspondances requises entre les éléments, le nombre de ces fonctions peut être important et au final, fastidieux à écrire à la main. Ce problème est facilement contournable via un outillage approprié en générant automatiquement ces fonctions pour un méta-modèle donné. En effet, ces fonctions sont toujours basées sur le même schéma, en vérifiant la correspondance des attributs et des références d'un élément de manière transitive. Pour une liste de références ou d'attributs, on vérifie la

6. Les correspondances sont donc vérifiées de manière transitive (la correspondance des classes implique la correspondance des attributs de ces classes qui implique à son tour la correspondance des types de chacun des attributs) mais il n'est pas nécessaire de rester dans le même mode de correspondance (totale partout ou partielle partout). Par exemple, la correspondance des classes pourra être totale mais la correspondance de ses attributs sera partielle.

correspondance des éléments de la liste un par un. L'outil pourrait analyser un méta-modèle quelconque et proposer au concepteur de choisir, pour chaque type d'élément de ce méta-modèle, le type de correspondance souhaitée (pour une correspondance partielle, on choisira les attributs et références requis). À partir de là, l'outil générera toutes les fonctions OCL de correspondance qu'il suffira de compléter ou de modifier légèrement au besoin.

Lors de la définition de ces fonctions, il faut tout de même faire attention à un détail important : éviter les cycles de correspondance. Par exemple, un attribut possède un champ `owner` de type `Classifier` référençant le propriétaire de l'attribut. Quand on vérifie la correspondance de deux classes, on vérifie transitivement la correspondance des attributs ; mais si pour un attribut on vérifie la correspondance des champs `owner`, on reboucle dans la vérification des correspondances des classes initiales. Il ne faut donc pas vérifier la correspondance de ce champ. Si l'on veut avoir une correspondance en profondeur, c'est-à-dire s'appliquant transitivement sur un grand nombre de types d'éléments, ce problème devient d'autant plus important et complexe à gérer. On pourrait là encore imaginer que notre outil vérifierait, en fonction des choix de correspondances, qu'un tel cycle n'est pas présent.

4.3.3. *Contraintes sur l'évolution des éléments pour l'exemple d'ajout d'interfaces*

La troisième partie du contrat, concernant l'évolution des éléments entre le modèle source et le modèle cible est exprimée sur la figure 3 par l'invariant des lignes 10 et 11 qui s'applique sur les modèles source et cible.

L'invariant consiste à appeler la fonction `sameClasses` (1) qui, pour deux bases de modèle, commence par vérifier que le nombre de classes est le même (2). Ensuite, pour chacune des classes du premier modèle (3), on récupère sa liste de méthodes en concaténant les méthodes de la classe et de toutes ses interfaces (4). On vérifie via la fonction `hasMappingClass` que la classe courante à une correspondance dans l'autre modèle (5) et si ça n'est pas le cas, le contrat n'est pas validé (9). On récupère via la fonction `getMappedClass` la classe associée (6) ainsi que sa liste complète de méthodes (7). Puis on vérifie que les deux classes ont la même liste de méthodes via l'appel de `sameMethodSet` (8).

Faute de place, nous ne détaillerons pas l'ensemble des fonctions de correspondance utilisées par ce contrat. Celles qui concernent les classes sont néanmoins définies en figure 4. La fonction principale en est `classMapping` qui applique une correspondance partielle entre deux classes : comparaison du nom et de la liste des attributs (via l'appel de `sameAttributes`). Les fonctions de correspondance d'attribut auront une forme similaire.

La fonction `sameMethodSet` (figure 5) compare deux ensembles de méthodes. Cela se fait en vérifiant d'abord que les ensembles ont le même nombre d'éléments. Ensuite, on vérifie, via la fonction de correspondance de méthode `methodMapping`, que chaque élément du premier ensemble se trouve dans le second.

```
1   context ModelBase def: sameClasses(mb : ModelBase) : Boolean =
2       self.allClasses -> size() = mb.allClasses -> size() and
3       self.allClasses -> forAll( c |
4           let myMethods : Set(Method) = c.interfaces -> collect(i | i.methods)
                                              -> union(c.methods) -> flatten() in
5           if c.hasMappingClass(mb)
            then
6               let eqClass : Class = c.getMappedClass(mb) in
7               let eqClassMethods : Set(Method) = eqClass.interfaces -> collect(i |
                            i.methods) -> union(eqClass.methods) -> flatten() in
8               c.sameMethodSet(myMethods, eqClassMethods)
            else
9               false
            endif)

10  context ModelBase inv checkInterfaceContract:
11  targetModel.sameClasses(sourceModel)
```

Figure 3. *Contraintes d'évolution entre les modèles source et cible*

```
context Class def: classMapping(cl : Class) : Boolean =
    self.name = cl.name and
    self.sameAttributes(cl)

context Class def: hasMappingClass(mb : ModelBase) : Boolean =
    mb.allClasses -> exists( cl | self.classMapping(cl))

context Class def: getMappedClass(mb : ModelBase) : Class =
    mb.allClasses -> any ( cl | self.classMapping(cl))
```

Figure 4. *Fonctions de correspondance pour une classe*

```
context Classifier def: sameMethodSet(
                            mets1 : Set(Method), mets2 : Set(Method)) : Boolean =
    mets1 -> size() = mets2 -> size() and
    mets1 -> forAll ( m1 |
        mets2 -> exists ( m2 | m1.methodMapping(m2)) )
```

Figure 5. *Fonction de correspondance d'ensembles de méthodes*

Toutes ces fonctions et invariants formant le contrat sont écrits en OCL standard. Il est donc vérifiable par n'importe quel évaluateur OCL implémentant le standard et pouvant lire des modèles Ecore. D'un point de vue pratique, nous avons opté pour l'outil ATL pour valider un modèle global par rapport au contrat. Bien qu'étant à la base un outil de transformation de modèles, ATL s'est avéré pertinent pour plusieurs raisons. D'abord, il implémente entièrement le standard OCL et permet de manipuler des modèles Ecore ou UML. Ensuite, il est relativement simple de valider des contraintes OCL sur un modèle via une transformation, comme expliqué dans (Bézivin *et al.,* 2005). On commence par définir un méta-modèle cible qui définit simplement un élément d'information détaillant un résultat, avec deux spécialisations selon la validité du résultat : *Correct* et *Error*. On définit ensuite une règle de transformation qui, pour une instance de la base globale, génère dans le modèle cible soit une instance de *Correct* si la base respecte l'invariant du contrat (celui de la figure 3) soit une instance d'*Error* dans le cas contraire. On obtient ainsi, via ce modèle cible généré, le résultat global pour la validation du contrat. De plus, on peut être plus précis dans le détail du résultat en établissant des règles de validation de la même façon pour chacune des classes. Ainsi, on peut connaître facilement la liste des classes qui ne respectent pas leur part du contrat.

4.4. *Résumé de la méthode*

Notre méthode se résume par les étapes suivantes :

1) Définir un mécanisme permettant de concaténer deux modèles au sein d'un troisième modèle (les trois étants conformes au même méta-modèle).

2) Pour une transformation donnée, définir un contrat composé de trois parties, chacune étant un ensemble d'invariants OCL :

- Contraintes que doit respecter le modèle source
- Contraintes que doit respecter le modèle cible
- Contraintes d'évolution entre le modèle source et le modèle cible, définies à partir d'un ensemble de fonctions de correspondance entre types d'éléments

3) Pour un couple de modèles, valider le contrat en vérifiant que les contraintes sur le modèle source (respectivement cible) du contrat sont respectées par le modèle source (respectivement cible) et que la troisième partie du contrat est respectée par le modèle concaténant les modèles source et cible.

L'approche a été expérimentée en mettant notamment en œuvre le raffinement d'ajout d'interfaces et en utilisant un ensemble varié d'outils sans que cela ne pose de problème d'interopérabilité. Nous avons utilisé la plate-forme Eclipse/EMF pour définir le méta-modèle et créer ou modifier des modèles. Les transformations automatiques ont été réalisées en Kermeta et en ATL. ATL a également été utilisé pour vérifier le contrat sur un couple de modèles.

5. Travaux connexes

Notre approche a pour cœur l'utilisation d'OCL car, comme nous l'avons expliqué, c'est un langage relativement bien connu et suffisamment ouvert pour être le meilleur candidat pour la définition de contrats de transformations. Nous allons donc comparer ici notre méthode à d'autres approches se basant également sur OCL soit pour spécifier des raffinements de modèles, soit des contrats de transformations similaires aux notres. Par exemple, (Pons *et al.,* 2006) définit formellement en OCL des raffinements de modèles et (Song *et al.,* 2007) se base sur OCL pour effectuer des raffinements. (Van Grop, 2008) définit des contrats de transformations et du *refactoring* de modèles en OCL. (Baudry *et al.,* 2006, Mottu *et al.,* 2006) propose également d'utiliser des contrats de transformations de modèles en OCL pour spécifier l'oracle d'un test de transformation.

Dans quasiment toutes ces approches, il y a nécessité de définir des correspondances entre éléments du modèle source et éléments du modèle cible, comme pour la notre. Mais ces correspondances sont toujours définies de manière *ad-hoc* pour le contexte considéré, et parfois seulement de manière implicite. Nous avons proposé au contraire une méthode générale pour définir explicitement et en détail les correspondances entre éléments, dans le contexte des transformations endogènes.

L'autre différence est que la plupart de ces approches sont moins générales que la notre du point de vue de l'obtention et de la manipulation des modèles ou sont dédiées à des environnements logiciels particuliers. Par exemple, la spécification de raffinement dans (Pons *et al.,* 2006) se fait en utilisant la relation de dépendance UML stéréotypée par `«refine»` qui oblige à définir explicitement un diagramme UML dans lequel des éléments sont liés par cette dépendance. Mais il n'est pas proposé de méthode pour, à partir de deux modèles obtenus de manière quelconque, définir automatiquement un modèle conforme à cette représentation. (Song *et al.,* 2007) propose de guider le concepteur en exécutant automatiquement des raffinements pendant l'édition d'un modèle quand ce dernier est dans un certain état (cet état est spécifié en OCL). Le raffinement est donc effectué de manière instantanée par l'outil et il n'est pas possible de vérifier a posteriori que deux modèles respectent la spécification du raffinement comme nous le proposons.

6. Conclusion et perspectives

Nous avons présenté une méthode facilement outillable pour spécifier et valider un contrat de transformation de modèles dans le cadre de n'importe quel type de transformation endogène. Nous nous sommes ici plus particulièrement intéressés au cas des raffinements de modèle. L'idée générale est de considérer un couple de modèles, l'un étant le source et l'autre le cible d'une transformation ou d'un raffinement, et de vérifier que ce couple respecte bien le contrat de la transformation.

Cette méthode a été conçue pour être la plus générale possible : on peut prendre en compte des modèles obtenus à partir d'outils et de méta-modèles variés (UML, défini

à partir du MOF ou de Ecore). La transformation de modèle peut ainsi être réalisée automatiquement et/ou via une intervention manuelle d'un concepteur. Le contrat est écrit en OCL pour être là encore le plus indépendant possible des outils.

Nous avons mis en avant la nécessité de détailler les correspondances entre éléments des deux modèles ainsi que de pouvoir concaténer deux modèles au sein d'un modèle plus global afin d'élargir la possibilité en termes d'outils et de techniques pour obtenir les modèles source et cible. Au niveau des correspondances, le prochain travail à réaliser est de développer un outillage permettant de générer automatiquement les fonctions OCL de correspondance afin de grandement simplifier l'écriture du contrat.

Enfin, la suite naturelle de notre approche est de la généraliser à des transformations exogènes, c'est-à-dire pour des méta-modèles source et cible différents. La principale difficulté est de passer outre la limitation d'OCL qui est que les contraintes ne sont exprimables que par rapport à un seul méta-modèle. Pour cela, on peut par exemple imaginer de fusionner les deux méta-modèles en un troisième plus global pour se retrouver avec un contexte unique, comme proposé dans (Baudry *et al.,* 2006).

7. Bibliographie

Baudry B., Dinh-Trong T., Mottu J.-M., Simmonds D., France R., Ghosh S., Fleurey F., Le Traon Y., « Model Transformation Testing Challenges », *ECMDA workshop on Integration of Model Driven Development and Model Driven Testing.*, July, 2006.

Beugnard A., Jézéquel J.-M., Plouzeau N., Watkins D., « Making Components Contract Aware », *IEEE Computer*, vol. 32, n° 7, p. 38-45, 1999.

Bézivin J., Jouault F., « Using ATL for Checking Models », *Intl. Workshop on Graph and Model Transformation (GraMoT 2005)*, vol. 152 of *ENTCS*, p. 69 - 81, 2005.

Cariou E., Marvie R., Seinturier L., Duchien L., Model Tranformation Contracts and their Definition in UML and OCL, Technical Report n° 2004-08, LIFL, April, 2004a.

Cariou E., Marvie R., Seinturier L., Duchien L., « OCL for the Specification of Model Transformation Contracts », Workshop OCL and Model Driven Engineering of UML'04, 2004b.

Meyer B., « Applying "Design by Contract" », *IEEE Computer (Special Issue on Inheritance & Classification)*, vol. 25, n° 10, p. 40-52, 1992.

Mottu J.-M., Baudry B., Le Traon Y., « Reusable MDA Components : A Testing-for-Trust Approach », *MoDELS 2006*, vol. 4199 of *LNCS*, Springer Verlag, 2006.

OMG, « Object Constraint Language (OCL) Specification, version 2.0 », 2006. `http ://www.omg.org/spec/OCL/2.0/`.

OMG, « Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, version 1.0 », 2008. `http ://www.omg.org/spec/QVT/1.0/`.

Pons C., Garcia D., « An OCL-Based Technique for Specifying and Verifying Refinement-Oriented Transformations in MDE », *MoDELS 2006*, LNCS, Springer Verlag, 2006.

Song H., Sun Y., Zhou L., Huang G., « Towards Instant Automatic Model Refinement Based on OCL », *APSEC '07*, IEEE Computer Society, p. 167-174, 2007.

Van Grop P., Model-Driven Development of Model Transformations, PhD thesis, University of Antwerp, Dept. of Mathematics and Computer Science, 2008.

From UML Actions to Java

Anis Charfi[*] — Heiko Müller[**] — Andreas Roth[*] — Axel Spriestersbach[***]

[*] *SAP Research CEC Darmstadt*
Darmstadt, Germany

[**] *Darmstadt University of Technology*
Darmstadt, Germany

[***] *SAP Research CEC Karlsruhe*
Karlsruhe, Germany

RÉSUMÉ. *Le langage d'actions du standard UML 2 définit des primitives atomiques pour la spécification de comportement orienté-objet d´une façon indépendante de la plateforme. Bien que le langage d'actions d'UML 2 a été proposé il y a plus que trois ans il n´existe aucune projection de ce langage vers un langage de programmation et il n'y a également aucun outil permettant la génération de code a partir des modèles d'actions UML. Dans ce papier, nous définissons une projection des actions UML vers Java et nous présentons aussi un générateur de code qui réalise cette projection. La projection et le générateur de code sont configurables par rapport aux extensions du meta-modèle et de la plateforme cible.*

ABSTRACT. *UML actions is a part of the UML 2.0 standard that defines a set of atomic actions for object-oriented behavior specification at the PIM level. Although UML actions were proposed more than three years ago there exists no mapping from UML actions to a programming language and there is also no tool supporting code generation from behavioral UML actions models. In this paper, we define a mapping from UML actions to Java and present a code generator implementing that mapping. We also show that the defined mapping and its implementation are configurable with respect to extensions of the meta-model and the target platforms.*

MOTS-CLÉS : *IDM, modélisation comportementale, langage d'actions UML, génération de code*

KEYWORDS: *MDA, behavior modeling, UML action semantics, mapping, code generation*

1. Introduction

The aim of Model-Driven Software Development (MDSD) and the Model-Driven Architecture (MDA) is to make models the primary artifacts in software development. However, so far models are mainly focusing on structural aspects whereas behavioral aspects are addressed only partly. As a result, MDA-light [GRU 05] has emerged, which refers to the approach adopted by most existing MDA tools today. The tools generate code skeletons from the structural models and the programmers have to complete the behavior manually by writing code in a specific programming language and for a specific technology platform. This approach breaks the portability, interoperability, and reuse objectives of MDA. UML 2.0 addresses the shortcomings of the UML in behavioral modeling by introducing UML action semantics, which defines atomic behavioral units that allow behavioral modeling of methods at the Platform Independent Modeling (PIM) layer. Although UML 2.0 was released a few years ago, there is no mapping from UML actions to any programming language and there is also no tool for code generation from UML action models.

In the context of the EU Project VIDE[1] [VID 08a], which focuses on methodologies and tools for behavioral modeling, we defined textual and visual notations for UML actions[2] and built supporting editors. Further, we defined mappings from UML actions to Java and ABAP [FAR 07]. Another mapping to the ODRA platform was defined [FAL 08]. Based on those mappings, model compilers were built, which support the generation of complete and compile-ready applications including their behavioral parts. In this paper, we present the mapping of UML actions to Java and the supporting code generator. One important requirement was that the mapping and its implementation should be configurable with respect to changes/extensions of the source meta-model (e.g., when OCL is used as expression and query language in UML action models) as well as extensions of the target platform (i.e., it should not only support plain Java but also Java EE).

The work presented in this paper provides three contributions. First, we present the first and probably the only mapping of UML actions to Java. Second, we present a code generator, which incorporates that mapping. Our experiments with this tool confirm that UML actions are sufficient for behavior modeling at the PIM level and that 100% code generation from UML action models is possible. Third, we present a mapping from OCL to Java, which has been integrated with the action mapping and implemented as well.

The remainder of this paper is structured as follows : Section 2 gives background on UML actions and motivates the need for mappings from UML actions to programming languages as well as for code generators. Section 3 presents our mapping of UML actions to Java and also a mapping from OCL expressions to Java. Section 4 presents a model compiler tool that implements the presented mappings. Section 5 reports on related works and Section 6 concludes the paper.

1. This work is supported by the European Commission FP 6 Project VIDE - IST-033606-STP.
2. UML 2 defines only the abstract syntax of actions, i.e., it does not specify a concrete syntax.

2. Background and motivation

This section gives background on UML Actions and on the PIM-level language defined in VIDE project.

2.1. *UML 2 Action Semantics*

UML *actions* are atomic units for behavior modeling, which allow modeling procedural and object-oriented behavior independently of the target platform. UML integrates actions for the specification of method behavior by assigning a UML Behavior such as a UML Activity to the method. Activities can be created using activity diagrams, for instance. An action may take input data via so called *input pins* and produce some output data via *output pins*. The execution order of actions can be defined either using a Petri Net like flow concept that is based on control flow edges or using special nodes that structure the contained actions. UML defines several actions for various purposes, e.g., to create and destroy object instances, to perform operation calls, to read and set class properties or variables, etc. A *readStructuralFeatureValueAction* for instance accesses the object at hand using an input pin and places the value of the structural feature on an output pin. The pins of different actions may be connected by object flows to express the data flow between these actions.

Although UML 2.0 was released a couple of years ago there is no mapping from UML actions to a specific programming language and there is also no tool supporting code generation from UML 2.0 action models. Having such mappings and supporting code generators is necessary for an MDA development process, in which behavior is modeled using UML actions and then complete and compile-ready applications are generated to different programming languages and platforms.

2.2. *The VIDE PIM-level language*

The VIDE project makes use of a PIM level behavior modeling language, which is based on UML actions and OCL expressions [FAL 07] as specified in the MDT project [ECL 08]. This language covers most concepts of UML 2 structural modeling, such as packages, classes, operations, and properties. Regarding associations, the scope of VIDE is limited to binary associations ; qualified associations are not included. Dependencies - rather dedicated to modeling as to programming - are excluded, too. Only non-overlapping, multiple inheritance is allowed.

Behavioral modeling in VIDE is well supported by including the following UML 2 action types : invocation actions, object creation/destruction actions, link actions, structural feature actions, and variable actions. Some action types are not in the scope of VIDE namely actions for reading system state as all side-effect free queries are expressed in OCL, actions for signal processing, and actions for changing the classification of objects.

3. The Mapping

In this section, we first present a running example and show the output of our mapping for a behavior model, which is part of that example. Then, we explain the mapping of certain concepts of the UML meta-model, which is needed for understanding the action mapping. After that, we present the mapping of selected actions and the mapping of OCL expressions. Our mapping supports all concepts of the VIDE PIM-level language. It has been validated through several complex example models from the business application domain.

3.1. *Running example*

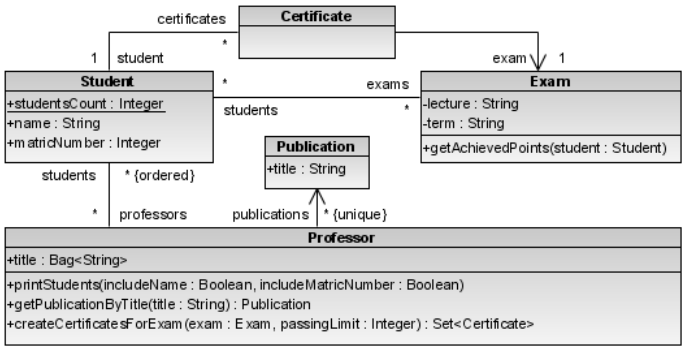


Figure 1. *Example model - class diagram*

For illustrating the mapping, we introduce the simple class diagram example shown in Fig. 1. Further, we present in Fig. 2 a behavioral model of the operation *createCertificatesForExams()* of the class Professor. This operation returns a set of certificates for those students that got enough points in an exam. The number of points needed to pass and the exam are both given as operation parameters. The activity diagram model in Fig. 2 uses UML actions and it was created with Visual Paradigm for UML [INT 08]. As all other UML tools, Visual Paradigm does not provide a concrete syntax for specific UML Actions. Thus, we used stereotypes and meaningful action names to make the modeled behavior more understandable.

Before going into the details of the mapping, we show in the listing below the output code for the behavior model shown in Fig. 2. The comments were added manually to interrelate the model elements with the corresponding lines of code.

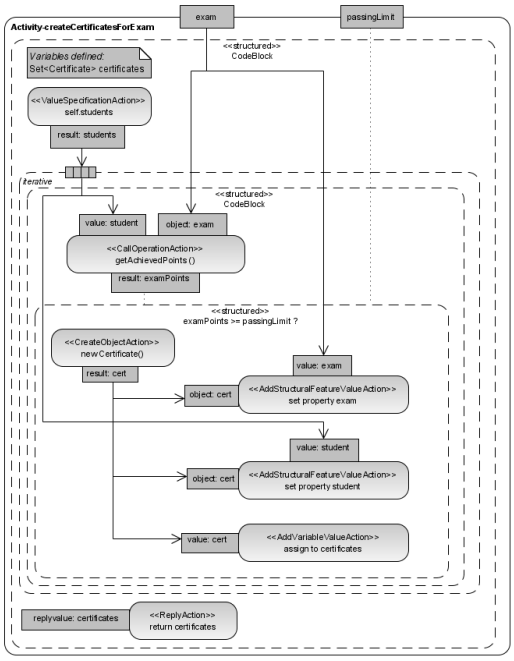


Figure 2. *Example model : Behavior model using an activity diagram*

```
public Set< Certificate > createCertificatesForExam (Exam exam, int passingLimit) {
  // Variable
  Set< Certificate > certificates = new HashSet< Certificate >();
  // ExpansionRegion; ExpansionNode and ValueSpecificationAction self.students
  // with OCL expression specifying the collection to be iterated
  for (Student student : students) {
    // CallOperationAction getAchievedPoints()
    int examPoints = exam.getAchievedPoints( student );
    // ConditionalNode with one Clause; ValueSpecificationAction with OCL expression
    // specifying the conditional statement
    if (examPoints >= passingLimit){
      // CreateObjectAction new Certificate()
       Certificate cert = new Certificate ();
      // AddstructuralFeatureValueAction set property exam
       cert .setExam(exam);
      // AddStructuralFeatureValueAction set property student
      // (Association Certificate-Student is synchronized)
       cert . setStudent ( student );
      // AddVariableValueAction assign to certificates
        certificates .add( cert );
    }
  }
  // ReplyAction
  return certificates ;
}
```

3.2. *Mapping structures, control flow, and object flow*

3.2.1. *Mapping structures and data types :*

Whilst the mapping of some structural parts of UML models such as packages and classes to Java is straightforward, several UML concepts have no direct counterpart in Java such as properties representing association ends and multiple inheritance. As association end properties are used by several UML actions the chosen mapping strategy for structural aspects has direct impact on the rules provided for the action mapping. Following a similar approach to the one implemented in [PAD 08] for bi-directional associations, we provide a set of mutator methods for each property ensuring the synchronization of the associated ends as listed in Table 1. We introduce mutator functionality with equivalent method signatures for both uni-directional association ends and class attributes. This results in a uniform interface of mutator methods for properties independently of whether they are attributes or association ends. The following methods are therefore widely used in the action mappings, e.g., in the context of the structural feature actions addressed in Section 3.3.4.

SHORT NAME	METHOD DESCRIPTION
ObjectSet	Sets the property's value.
ObjectAdd	Adds an object to the multi-valued property.
ObjectRemove	Removes an object from the multi-valued property.
RemoveAll	Clears a multi-valued property.
PositionAdd	Adds an object at a specified position to the multi-valued, ordered property.
PositionRemove	Removes the element at the specified index from the multi-valued, ordered property.
DoublePositionAdd	Adds objects at the specified positions on both association ends.
DoublePositionRemove	Removes the elements at the specified indices on both association ends.

Tableau 1. *Property mutator methods*

Another equally important aspect is the mapping of the data types used in the input model–including collections and simple types–to adequate Java concepts. In our work, we used the data types defined by the OCL standard library and we mapped them to equivalent Java primitive types and generic Java collections (from *java.util.collection*).

3.2.2. *Mapping control flow :*

UML activities provide two means to define the order in which the actions are executed : a Petri net like semantics based on *ControlNodes* and *ControlFlows* and sequencing by means of *StructuredActivityNodes*, which specify the order of their contained elements such as *ConditonalNode* and *ExpansionRegion*. In addition to a concept for traversing the actions graph when *ControlNodes* and *ControlFlows* are

used, we defined also mappings of the different subclasses of *StructuredActivityNode* to semantically equivalent Java statements. Details can be found in [MUE 08].

3.2.3. *Mapping object flow :*

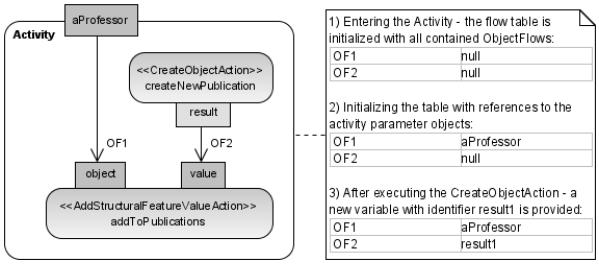


Figure 3. *ObjectFlow example*

Most actions can consume data via their *input pins* and produce other data via their *output pins*. The pins of different actions can be connected using object flows. In our mapping, the meta-classes involved in the object flow concept are not mapped into Java code directly. Instead, the objects passed via object flow are made available using a special table that keeps track of the accessible objects while traversing the action graph. For illustration a small example showing how the object flow table works is given in Fig. 3.

3.3. *Mapping UML actions to Java*

Due to space limitations we can only give some mapping examples for the different action categories. The complete mapping can be found at [MUE 08] and [VID 08b].

3.3.1. *Mapping invocation actions :*

As an example of invocation actions, we describe the mapping of *CallOperationAction*, which invokes the execution of the associated behavior of an operation. The operation is called on a dynamically resolved object, accessible by the *target* input pin. A sequence of *argument* input pins can be transmitted to the called operation and results of it (if there are any) are stored to a *result* output pin.

A *CallOperationAction* is mapped to a method call statement in Java. The name of the method is the value of the *name* property of the operation associated to the action. If the operation is static, then the Java method call statement includes the name of the Java class that holds the method. The information on whether or not an operation is static is available from the operation property *isStatic* and the Java class name can be obtained from the class association of the operation. If the operation is not static, the name of the *target* input pin (i.e., the reference to the target object) has to be included

in the generated Java method call statement. For each *argument* input pin targeting a parameter with direction *in*, the name of the respective object has to be included as parameter in the generated Java method call statement. If the operation has a return parameter then a new temporary variable is declared and the method call statement is assigned to it. A reference to this variable is stored into the *result* output pin. As an example, Fig. 4 shows an activity containing a *CallOperationAction* that calls the operation *printStudents* (cf. the class digram shown in Fig. 1). The target object of type *Professor* is provided at runtime via a *target* input pin ; additionally, there are two *argument* input pins that provide the parameters *includeName* and *includeMatricNumber* of the operation of direction *in*.

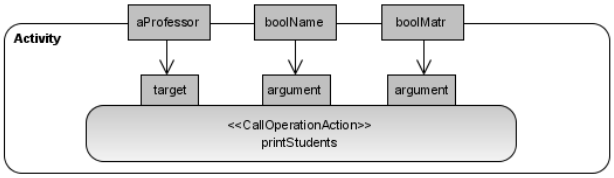


Figure 4. *CallOperationAction example*

The Java code generated from mapping this model is shown below.

```
aProfessor . printStudents (boolName, boolMatr);
```

3.3.2. *Mapping object creation/destruction actions :*

Object actions allow to create and destroy objects at runtime. As an example, we describe the mapping of *CreateObjectAction*, which dynamically creates an instance of a classifier. The newly created object is made accessible using the *result* output pin.

This action is mapped to a Java assignment statement. The right side of the assignment is a Java constructor call statement and its left side is a new variable of type classifier, which is used to store the *result* output pin of the *CreateObjectAction* action. As an example, a *CreateObjectAction* with its classifier set to class *Publication* will be mapped to the following Java statement. Note that the identifier *result_1* of the newly created variable must be checked to be unique in the current context.

```
Publication   result_1  = new Publication ();
```

3.3.3. *Mapping association and link actions :*

Actions of this category allow the manipulation of associations and links. These actions should preserve the synchronism of the involved association ends. There are two kinds of actions in this category : *ClearAssociationAction* is targeting the association itself, with direct impact on potentially several links. In contrast, a *LinkAction* operates on a single link, with detailed settings for each link end. For explaining the mapping of these actions recall property mutator methods introduced in Section 3.2.1.

As an example, we present the mapping of *CreateLinkAction*, which creates or mutates a link. Its configuration information is provided in form of two instances of *LinkEndCreationData* A and B :

– The *end* association defines the association end.

– The *value* input pin dynamically specifies the object that holds the end property. For synchronisation, *value* is added to the opposite end that is described by the other *LinkEndCreationData*.

– The property *isReplaceAll* specifies whether or not all existing elements of the end property should be replaced.

– For multi-valued ordered association ends a position *insertAt* can be specified.

This results in numerous possible combinations of *LinkEndCreationData* settings. Therefore, we focus in the following on collection-type association ends, omitting the simple case of single-valued properties. Additionally, we adopt the following two restrictions defined in [OMG 05] :

– If an end is not ordered, then *insertAt* is ignored.

– If *isReplaceAll* is set to true, the mapping disregards the corresponding *insertAt*.

The remaining cases are covered by the following rules :

1) $isReplaceAll_A = false$ and $isReplaceAll_B = false$:

a) $insertAt_A$ is unspecified and $insertAt_B$ is unspecified : Output is a Java method call statement with $value_A$ as target and $value_B$ as parameter. The method is the *ObjectAdd* mutator of end_A. Note that in case of unidirectional associations, only the *value* object that knows about the other association end has the functionality to change the link. The allocation of the target and parameter role is in this case not arbitrary. However, this differentiation is disregarded in the following, assuming that both ends are navigable.

b) $insertAt_A$ is specified, $insertAt_B$ is unspecified : Output is a Java method call statement with $value_A$ as target. The method is the *PositionAdd* mutator of end_A. Method parameters are $insertAt_A$ and $value_B$.

c) $insertAt_A$ is unspecified, $insertAt_B$ is specified : This situation is similar to the previous, if the roles *A* and *B* are switched.

d) $insertAt_A$ is specified and $insertAt_B$ is specified : Output is a Java method call statement with $value_A$ as target. The method is the *DoublePositionAdd* mutator of end_A. Method parameters are $insertAt_A$, $value_B$, and $insertAt_B$.

2) $isReplaceAll_A = true$, $isReplaceAll_B = false$:

a) $insertAt_B$ is unspecified : Output are two Java method call statements, both with $value_A$ as target. The first method is the *RemoveAll* mutator of end_A. The second method is the *ObjectAdd* mutator of end_A, with $value_B$ as parameter.

b) $insertAt_B$ is specified : Output are two Java method call statements. The first has $value_A$ as target. The method is the *RemoveAll* mutator of end_A. The second

has $value_B$ as target. The method is the *PositionAdd* mutator of end_B, with $value_A$ and $insertAt_B$ as parameter (shown in the example below).

3) $isReplaceAll_A = true$ and $isReplaceAll_B = true$: Output are three Java method call statements. The first is a call of the *RemoveAll* mutator of end_A on $value_A$, the second is a similar Java method call for end_B and $value_B$. The third method call adds $value_B$ to the end_A property of $value_A$, using the *ObjectAdd* mutator.

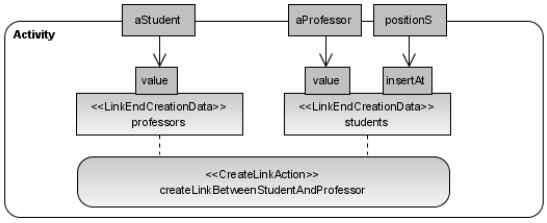


Figure 5. *CreateLinkAction example*

As an example, Fig. 5 shows a model that creates a link between an instance *aProfessor* of class *Professor* and an instance *aStudent* of class *Student*. As in this case $isReplaceAll_{professors} = true$, all associated professors of *aStudent* should be removed. On the other side, as $insertAt_{students}$ is specified, the newly associated student should be inserted in the *students* list of *aProfessor* at the specified index *positionS*. One has to ensure manually the correct ordering of the parameters and the correct connection of parameters with pins. The input model shown in Fig. 5 maps to the following Java code fragment :

```
aStudent.removeAllFromProfessors();
aProfessor.addToStudents( positionS , aStudent );
```

3.3.4. *Mapping structural feature actions :*

Structural feature actions allow to access and manipulate properties of objects (i.e., clear, add, or remove values). A property may be either an attribute of a class or an association end. Using the property mutator methods defined in Section 3.2.1 allows us to abstract from this differentiation.

As an example, we present the mapping of *RemoveStructuralFeatureValueAction*. This action has an input pin specifying the *object* to be manipulated, and may have an input pin providing the *value* to be removed. If the structural feature is multi-valued and ordered, then an input pin *removeAt* can be used to specify the element index to be removed. If the structural feature is multi-valued and unique, *isRemoveDuplicates* can be set to true with the effect that all duplicates of the specified element are removed. In the following, the mapping rules of this action for multi-valued properties (as this is the complex case) are listed :

1) $isRemoveDuplicates = false$:

a) $removeAt$ is unspecified : Output is a Java method call to the *ObjectRemove* mutator of the structural feature. Target of the method call is *object*, *value* is mapped as parameter.

b) $removeAt$ is specified : Output is a Java method call to the *PositionRemove* mutator of the structural feature. Target is *object*, *removeAt* is the method parameter.

2) $isRemoveDuplicates = true$:

a) $removeAt$ is unspecified : Output is an empty Java while loop with a method call as test statement. The test method is the *ObjectRemove* mutator of the structural feature, targeting *object*. The required method parameter is *value*.

b) $removeAt$ is specified : A new variable of the same type as the elements of the structural feature is declared and initialized with the element at index *removeAt* of the structural feature. After that, an *ObjectRemove* method call is generated, as a test statement in an empty while loop. The temporary variable serves as parameter of the remove method.

Fig. 6 shows a *RemoveStructuralFeatureValueAction*, removing the String *aTitle* from the *titles* collection of the *Professor* instance *aProfessor*. The action's property *isRemoveDuplicates* is set to true in the corresponding property view of the editor.

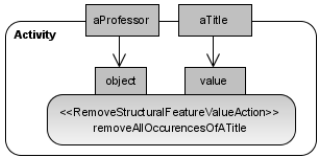


Figure 6. *RemoveStructuralFeatureValueAction example*

Based on the mapping rules for this action, which were described above, the following Java code is generated for the example shown in Fig. 6 :

```
while ( aProfessor .removeFromTitles( aTitle ){ }
```

3.3.5. *Mapping variable actions :*

Variable actions access and manipulate variables. Their functionality is similar to that provided by structural feature actions : variables may be read, cleared, values can be added or removed. Therefore the mapping of the variable actions is very similar to the mapping of the structural feature actions described above. The mapping of variable actions is even simpler because there is no need to call the generated mutator methods (which synchronize association ends in case of association end structural features) as variables can be manipulated directly (using functionality provided by Java *Collection* interfaces, in case of multi-valued variables). As an example, a *ClearVariableAction* on a variable *listOfStudents* maps to the following Java statement :

```
listOfStudents . clear ();
```

3.4. *Mapping OCL to Java*

The MDT UML2 meta-model which we use as basis offers various ways to specify expressions. One possibility is to use a set of "read" actions provided in the UML standard. Another one is to use opaque expressions, i.e., language expressions of a language like Java. As a third possibility OCL expressions can be used. As integral part of the UML standard, usually used to express invariant or pre- and post-condition properties on UML class models, a subset of OCL is naturally an expression language. It is integrated into the meta-model by (indirectly) subclassing the *ValueSpecification* metaclass with an *ExpressionInOCL* meta-class. The latter references the root of the OCL expression model tree.

While the use of read actions is too operational and complex in terms of the needed meta-model elements and the use of implementation language expressions is too target language dependent, the use of OCL expressions has several advantages. It unifies the way expressions are modeled in constraints and in the executable model. Moreover, it allows for an easy substitution of the OCL query language against other domain specific modeling languages which can be found in an enterprise modeling context.

A certain subset of OCL can very easily be mapped to Java. With regard to readability, it is desirable to have compact code. Ideally, the OCL expression should be directly integrated in the mapping product of the containing model element. This is possible for the leaf nodes and for some inner nodes of the abstract syntax tree produced by an OCL parser :

– *LiteralExp* : These types just return their specified value, e.g. an *IntegerLiteralExp* returns the value of its *integerSymbol*.

– *VariableExp* : A *VariableExp* returns the name of the referred variable.

– *PropertyCallExp* : The mapped source is followed by the Getter accessor method call of the referred property.

– *OperationCallExp* (basic type operations) : The operations of the basic types *Boolean*, *Integer*, *Real* and *String* can be directly transferred to Java primitive type functionality.

– *OperationCallExp* (parts of collection type operations) : Some operations have direct Java counterparts, e.g. *size*, *includes* or *isEmpty*.

– *IfExp* : If the *IfExp* is the value of a simple assignment, it can be translated into a Java conditional expression (map_cond ? map_then : map else).

More effort is needed to translate OCL's iterator expressions, such as *exists*, *collect*, *select*, *forAll*, *isUnique*, *sortedBy*, and *reject*, dealing with access to elements of collections. We introduce separate helper methods which capture all the needed functionality of the specific iterator expression. The method is parameterized with argu-

ments needed in subexpressions. The actual expression to translate then consists only of a method call to that method. This avoids to clutter surrounding expressions and statements with lengthy operational code. To give an example, the OCL expression

```
self . professors  -> exists( p | p. students . size ()<studentCount)
```

is translated to the simple Java expression

```
profExists2 ( this . getProfessors (), studentCount)
```

making use of the helper method *profExists2* with the following definition :

```
private boolean profExists2 (java . util .Set<Professor> profs , int studentCount) {
  for (Professor temp : profs) {
    if ((temp. getStudents (). size () < studentCount))
      return true ;
  }
  return false ;
}
```

Note, that our full mapping implementation [MUE 08] can – because its definition as a model transformation – easily be aligned with attempts to benchmark OCL tools [GOG 08].

4. Implementation

We implemented an Eclipse plugin research prototype incorporating the mapping described in Section 3. This tool, which is based on openArchitectureWare [OPE 08], supports the generation of Java and Java EE 5 code from Eclipse UML2 models. Moreover, it also supports code generation from Eclipse UML2 models, which may contain OCL expressions and OCL queries (as required in the context of the VIDE project) and which could also be profiled (to e.g., define web service related information or persistence). The generator components are shown in Fig. 7.

The package *plugin* encapsulates the functionality specific to the Eclipse plugin and contains the two entry points to the generator, which launch the generator by invoking oAW's *WorkflowRunner* with suitable arguments to produce either plain Java or Java EE 5 code. The package *java* contains a subpackage *templates*, in which the mapping rules defined in Section 3 expressed in the XPand language to control the output generation and in the Xtend language to non-invasively enrich the metamodel with additional functionality. The subpackage *workflow* contains oAW workflow files that control the generation process. The package *jee5* contains the mapping rules toward Java EE. As it is very likely that the *jee5* functionality is extended or modified in the future (e.g. to support other Java EE features such as web services or persistence) the respective mapping implementation is factored out and kept in a separate module that non-invasively extends the *templates* package of the *java* module. However, as certain definitions contained in package *java.templates* have to be adapted when targeting e.g., web service based applications, we used the Aspect-Oriented Programming features of openArchitectureWare.

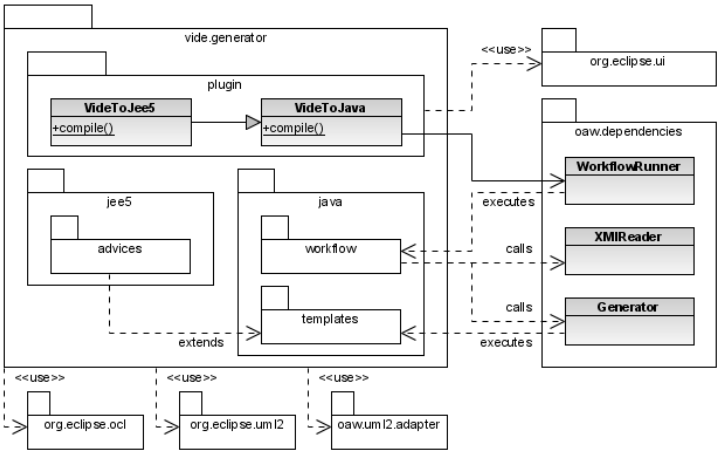


Figure 7. *Generator components*

5. Related Work

Executable software models were postulated already in the 1980s, but they were not widely adopted due to the lack of mature and uniform modeling standards and due to the claim of generating *all or nothing*. The new model-driven approach, based on UML [OMG 03], [OMG 05] and MDA [OMG 01], has overcome these issues and therefore has become very popular in the last decade.

A remarkable spectrum of academic work targets the mapping of UML to Java, mostly focusing on specific incongruences between UML structures and Java concepts. Papers related to this work can be found on topics such as mapping different UML concepts like associations [GÉN 03] or multiple inheritance [TEM 00]. Some papers define mapping rules to simulate the structural aspects of complete UML designs [HAR 00] targeting high-level skeletal Java implementations.

In the field of mapping UML behavior to Java there is some research on the mapping of state machines modeled in UML state charts [NIA 04], [NIC 00]. With regard to the use of state charts, these papers point to the same direction as the pioneer work on executable UML [MEL 02] where action language code is used to specify application logic. However, the authors of these works use the action language solely to specify actions which happen after an object changes its state to a different one.

With respect to tool support, there is a gap between current software capabilities to generate code from structural models and to generate code from behavioral models. Most existing tools generate only code skeletons intended for completion by a developer. Only a few tools consider behavioral models - such as state machines or activity diagrams - for code generation. For instance IBM Rational Rose [IBM 08],

Visual Paradigm [INT 08], and Telelogic Rhapsody [TEL 08] use state machines but most of them restrict behavioral code generation to a tool specific subset of statechart diagram elements. They all use programming languages such as C, C++, C#, or Java as action languages. No tool was found that provides a translation for UML 2.0 actions. Fujaba [PAD 08] uses so called story diagrams, an extension of UML activity diagrams, for Java code generation. Even though the modeling of behavior is well supported by some tools in most cases there are no explicit mappings of the behavioral part to semantically equivalent Java code.

There is not much related work dealing with UML actions as action language. In [HEI 06], Heitz et al. identify UML actions as a promising candidate representation for action languages. The authors see the major advantage of UML actions in their interoperability, as it is part of the UML standard. However, they conclude that the UML actions are not the most practical action language, due to missing tool support and, among other things, the complexity and fine-granularity of the metamodel. In the VIDE project, the complexity and granularity of the UML actions meta-model is hidden from the end users by providing concrete syntaxes and supporting editors (both textual and visual [VID 08a]) that shield the end user from many concepts such as object flow and explicit control flow whilst still being compliant with the UML 2 standard.

6. Conclusion

In this paper, we presented the first mapping from UML 2 actions to Java and a supporting code generator. We are not aware of any comparable mapping from UML actions to a programming language. Our mapping and its implementation prove that modeling business logic with UML actions and generating executable code from these models is possible. It even shows that generating compile-ready code of a whole applications modeled at the PIM level is feasible. Further, the code generator confirms the maturity of the MDA tool OpenArchitectureWare. We validated the presented mapping and the respective code generator through several complex models from the business applications domain.

7. Bibliographie

[ECL 08] ECLIPSE FOUNDATION, « Eclipse Modeling Project : Model Development Tools », `http://www.eclipse.org/modeling/mdt/`, December 2008.

[FAL 07] FALDA G., HABELA P., KACZMARSKI K., STENCEL K., SUBIETA K., « Platform-independent programming of data-intensive applications using UML », *Proc. of the 2nd IFIP CEE-SET Conference*, LNCS, Springer, October 2007, p. 301–310.

[FAL 08] FALDA G., HABELA P., KACZMARSKI K., STENCEL K., SUBIETA K., « Executable Platform Independent Models for Data Intensive Applications », *Proc. of ICCS*, vol. 5103 de *LNCS*, Springer, June 2008, p. 301-310.

[FAR 07] FARBER G., *ABAP Basics*, SAP Press, Harlow, 2007.

[GÉN 03] GÉNOVA G., DEL CASTILLO C. R., LLORÉNS J., « Mapping UML Associations into Java Code », *Journal of Object Technology*, vol. 2, n° 5, 2003, p. 135-162.

[GOG 08] GOGOLLA M., KUHLMANN M., BÜTTNER F., « A Benchmark for OCL Engine Accuracy, Determinateness, and Efficiency », *Proc. of MODELS 08*, Springer, 2008, p. 446–459.

[GRU 05] GRUHN V., PIEPER D., RÖTTGERS C., *MDA – Effektives Software-Engineering mit UML 2 und Eclipse*, Springer Verlag, Berlin, 2005.

[HAR 00] HARRISON W., BARTON C., RAGHAVACHARI M., « Mapping UML designs to Java », *OOPSLA*, 2000, p. 178-187.

[HEI 06] HEITZ C., THIEMANN P., WÖLFLE T., « Integration of an Action Language Via UML Action Semantics », *Proc. of the 2nd International Conference on Trends in Enterprise Application Architecture (TEAA)*, November 2006, p. 172-186.

[IBM 08] IBM, « Rational Rose Technical Developer », March 2008.

[INT 08] INTERNATIONAL V. P., « Visual Paradigm for UML », `http://www.visual-paradigm.com/product/vpuml/`, March 2008.

[MEL 02] MELLOR S. J., BALCER M. J., *Executable UML : A Foundation for Model-Driven Architecture*, Addison-Wesley, 2002.

[MUE 08] MUELLER H., « From UML Actions and OCL Expressions to Java », Diplomarbeit, TU Darmstadt, Germany, March 2008, `http://www.st.informatik.tu-darmstadt.de/database/theses/thesis/ThesisHeiko.pdf?id=79`.

[NIA 04] NIAZ I. A., TANAKA J., « Mapping UML Statecharts to Java Code », *Proc. of IASTED Conference on Software Engineering*, 2004, p. 111-116.

[NIC 00] NICKEL U., NIERE J., ZÜNDORF A., « The FUJABA Environment », *Proc. of the 22nd International Conference on Software Engineering (ICSE)*, June 2000, p. 742-745.

[OMG 01] OMG, « Model Driven Architecture », `http://www.omg.org/mda/`, 2001.

[OMG 03] OMG, « UML 2.0 Infrastructure Specification », `http://www.omg.org/cgi-bin/doc?formal/03-09-15.pdf`, September 2003.

[OMG 05] OMG, « UML 2.0 : Superstructure Specification version 2.0 », `http://www.omg.org/cgi-bin/doc?formal/05-07-04.pdf`, July 2005.

[OPE 08] OPENARCHITECTUREWARE P., « Official Homepage », `http://www.openarchitectureware.org`, June 2008.

[PAD 08] PADERBORN U., « Fujaba Tool Suite », `http://www.fujaba.de`, June 2008.

[TEL 08] TELELOGIC, « Rhapsody », March 2008.

[TEM 00] TEMPERO E. D., BIDDLE R., « Simulating Multiple Inheritance in Java », *Journal of Systems and Software*, vol. 55, n° 1, 2000, p. 87-100.

[VID 08a] VIDE C., « FP6 EU Project VIsualize all moDel drivEn programming », `http://www.vide-ist.eu/`, December 2008.

[VID 08b] VIDE EU PROJECT, « Deliverable 6.1 : Model Compilers », April 2008.

A Domain Specific Language for Expressing Model Matching

Kelly Garcés [*] [§] [£] — Frédéric Jouault [+] [§] — Pierre Cointe [*] [£] — Jean Bézivin [+] [§]

[*] *École des Mines de Nantes*
[+] *INRIA, Centre Rennes - Bretagne Atlantique*
[§] *AtlanMod team, EMN-INRIA*
[£] *AsCoLa team, LINA (UMR 6241) and INRIA*

{kelly.garces, pierre.cointe}@emn.fr
{frederic.jouault, jean.bezivin}@inria.fr

RÉSUMÉ. *Une stratégie de mise en correspondance calcule des liens entre deux modèles en exécutant un ensemble d'heuristiques. Dans ce papier, nous introduisons AML (pour* AtlanMod Matching Language*), un langage dédié à l'expression des stratégies de mise en correspondance des modèles. AML est conçu pour l'ingénierie des modèles et il implémente les stratégies de mise en correspondance par des chaînes de transformation de modèles. Chacune de ces transformations prend un ensemble de modèles en entrée et crée un modèle de correspondances en sortie. Nous présentons un compilateur qui prend en entrée des programmes AML et génère en sortie du code ATL (AtlanMod Transformation Language) et Apache Ant. Le code ATL instrumente les transformations de modèles de mise en correspondance et le code Ant orchestre leur exécution. Nous évaluons cette implantation sur deux stratégies de mise en correspondance composé d'ensembles de transformations issues de la litérature.*

ABSTRACT. *A matching strategy computes mappings between two models by executing a set of heuristics. In this paper, we introduce the AtlanMod Matching Language (AML), a Domain Specific Language (DSL) for expressing matching strategies. AML is based on the Model-Driven paradigm, i.e., it implements model matching strategies as chains of model transformations. A matching model transformation takes a set of models as input, and yields a mapping model as output. We present a compiler that takes AML programs and generates ATL (AtlanMod Transformation Language) and Apache Ant code. The ATL code instruments the matching model transformations, and the Ant code orchestrates their execution. We evaluate this implementation on two strategies including robust matching transformations from the literature.*

MOTS-CLÉS : *Génie des Modèles, Transformation de Modèles, Mise en correspondance.*

KEYWORDS: *Model-Driven Engineering, Model Transformation, Matching.*

1. Introduction

Business and technological changes influence the evolution and integration of software systems. Software engineers have to deal with the heterogeneity among all the artifacts (e.g., many versions for each software artifact) in order to get the systems working. Various solutions have been proposed to tackle this issue. One of them, called *Matching*, has been thoroughly studied in the ontology and database schema contexts. The matching operation is also gaining importance in Model-Driven Engineering (MDE) (Falleri *et al.*, 2008).

An MDE system basically consists of a set of models, i.e., terminal models and metamodels (Jouault *et al.*, 2006a). A metamodel is composed of concepts and relationships. A terminal model contains instances of the metamodel concepts. In general, the matching operation computes mappings between two models. A mapping is a relationship between two model elements similar to each other in a way. The research problem of the following works may illustrate the need of mappings between terminal models and/or metamodels :

– **Metamodel-to-metamodel (M2-to-M2).** (Cicchetti *et al.*, 2008) describes the problem of adapting terminal models to their metamodels evolving. They uses the mappings and differences between two metamodels (i.e., a given metamodel to a former version of the same metamodel) for deriving adaptation transformations.

– **Terminal model-to-terminal model (M1-to-M1).** (Javed *et al.*, 2008) recovers metamodels from a repository of terminal models. The approach translates the terminal models into grammar-based representations, an engine then infers a metamodel from them. This work may motivate the need of mappings between two terminal models from which we can obtain metamodels.

– **Terminal model-to-metamodel (M2-to-M1 or M1-to-M2).** (Favre, 2004) points out that software systems may rapidly become out of sync with the implementation. In MDE systems, an instance of this problem is that the developers often change code without updating the corresponding terminal models and metamodels. In this scenario, we imagine the mappings, between a terminal model (extracted from source code) and a given metamodel, as artifacts for reconstructing the metamodel (e.g., for finding out new metamodel types).

There are many strategies to deliver these kinds of mappings (we will refer to these strategies as *matching strategies*). They typically execute a set of matching heuristics. Inspired by them, we have proposed MDE matching strategies that find out M2-to-M2 mappings (Garcés *et al.*, 2008). Every strategy has been implemented as chains of model transformations, where each model transformation instruments a matching heuristic. A Matching Transformation (indicated as MT in the remaining sections) takes as input a set of models (the models to be match –$ModelA$ and $ModelB$–, a mapping model returned by other transformation, and additional models), and yields a mapping model conforming to a mapping metamodel.

After studying many matching strategies and performing some experimentations, (see full results in (Garcés *et al.*, 2008)), we have observed two main issues that impact the productivity of matching developers :

1) **Coupling of strategies and model representations.** Most of matching approaches (Ohst *et al.*, 2003)(Xing *et al.*, 2005)(Girschick, 2006)(Treude *et al.*, 2007) write heuristics in terms of one internal model representation (as it will have to be). Even though such heuristics compare very standard model features (e.g., labels, structure), these may be no longer applicable when the models to be matched conform to other metamodels. The solution often is to rewrite a given heuristic for each pair of metamodels. As a result, we get multiple heuristics encompassing the same matching logic.

2) **Lack of matching logic constructs.** Matching heuristics have been mostly instrumented using General-Purpose Languages (GPL), e.g., (Melnik *et al.*, 2003) and (Do, 2005). Matching experts may spend much more time for expressing a heuristic, using the GPL constructs, than the time for inventing it. Considering that the number of heuristics is rapidly increasing (OAEI, 2008), the question is : How can we reduce effort to express matching heuristics ?

By considering the fairly good results of our early MDE matching strategies (Garcés *et al.*, 2008), we have decided to make more generic the original approach (i.e., to develop not only strategies to match two metamodels, but to match any pair of models), and to tackle the two issues described above. To achieve this goal, we propose the AtlanMod Matching Language (AML), a DSL that provides constructs for straightforwardly expressing matching strategies. We deal with the coupling problem using a compilation strategy which adapts matching rules to the models to be matched. Whereas a detailed discussion about the compilation strategy is out of scope of the paper, this do focus on the language constructs, and the first significant results of using them.

An outline of the remainder of this paper follows. Section 2 explains the matching domain and the kinds of heuristics, and also introduces a motivating example. Section 3 describes the AML constructs and their semantics. Section 4 shows how to specify the motivating example using AML. Section 5 reports the experimentation results. Section 6 presents the related works. Finally, Section 7 concludes the paper.

2. Matching Domain Overview and Motivating Example

After analyzing the strategies compiled in (Euzenat *et al.*, 2007), we can characterize the matching domain. A matching strategy is a stepwise process that takes two models A and B, along with optional domain knowledge, and produces mappings between the models. A mapping relates (links) an element of A at most one element of B. A mapping has a similarity value (between 0 and 1) that represents how similar the linked elements are. A matching strategy deliveries mappings by applying at least 5 kind of heuristics :

– **Creation** establishes a mapping between the element a (in model A) and the element b (in model B) when these elements satisfy a condition.

– **Similarity** computes a similarity value for each mapping prepared by the creation heuristics. One function establishes the similarity values by comparing particular model aspects : labels, structures, and/or data instances (the latter has sense when the models represent metamodels). In general, a given comparison is direct and/or indirect. Direct means only the model elements, e.g., a and b, are compared each other. The indirect comparison separately examines a and b in relation to a third element (e.g, c) from a domain knowledge resource (e.g., an ontology).

– **Aggregation** combines similarity values by means of one expression. An expression often involves :

- n, the number of heuristics providing mappings.
- $\delta(a_i, b_i)$ similarity value computed by the heuristic i
- w_i weight or importance of the heuristic i, where $\sum_{i=1}^{n} w_i = 1$

– **Selection** selects mappings whose similarity values satisfy a condition.

– **Rewriting** retypes/reorganizes generic mappings that fill a condition. "Retypes" intends to transform a generic mapping to a richer semantic mapping. "Reorganizes" means to arrange mappings when there is a kind of relationships between the linked elements.

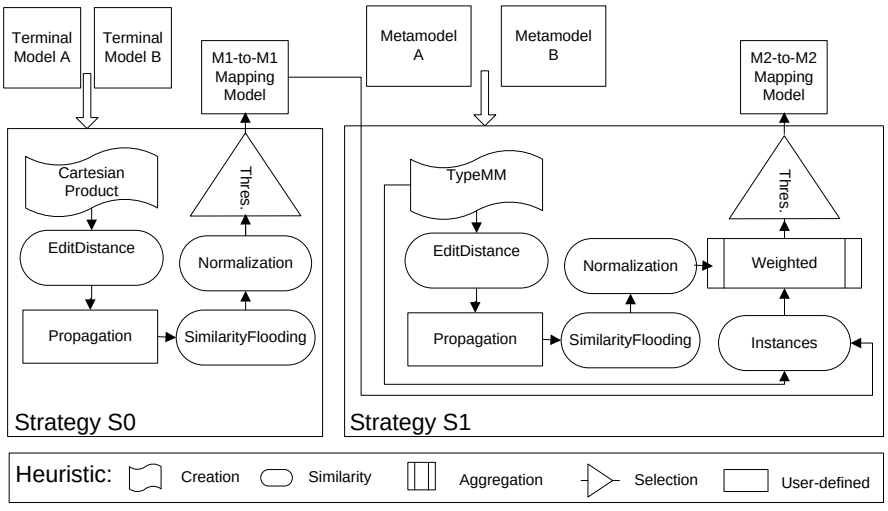


Figure 1. *The strategies S0 and S1*

Fig. 1 shows a motivating example whose purpose is to find out M2-to-M2 mappings using (among other model aspects) data instances. The main strategy,

i.e., S1, matches two metamodels $MetamodelA$ and $MetamodelB$, conforming to the same metametamodel, e.g., EMF/Ecore, KM3. In particular, S1 uses the matching transformation Instances which propagates similarity values of M1-to-M1 mappings to M2-to-M2 mappings. The M1-to-M1 mappings have been earlier computed by the strategy S0. The strategy S0 therefore matches two terminal models, $TerminalModelA$ and $TerminalModelB$, conforming to the metamodels $MetamodelA$ and $MetamodelB$. Remark on both S0 and S1 reuse the matching transformations EditDistance, Propagation, SimilarityFlooding, Normalization, and Threshold. The difference is that the transformations match terminal models in S0, and metamodels in S1. Note that Fig. 1 uses a particular symbol to indicate the type of each transformation. The figure incorporates an additional type, user-defined, which represents heuristics capturing functionality beyond our classification, e.g., Propagation. We describe these transformations in Section 4.

3. The AtlanMod Matching Language (AML) in a Nutshell

3.1. *Overview*

AML is a DSL for expressing MDE matching strategies. AML keeps the declarative flavor (of some model transformation languages) for expressing matching transformations, and adds dataflow programming constructs for describing matching transformation chains. Fig. 2 shows the tools that support AML :

The Compiler takes a given AML program (including the constructs matching rule and model flow) and performs the following tasks :

– Analyze the AML program (syntax and semantic)

– Merge pre-existent matching rules (which are available at a library of strategies)

– Generate a model matching transformation (written in a concrete model transformation language, i.e., ATL) for each matching rule. This transformation contains the matching rule, and additional copying rules. The ATL compiler generates executables from the code.

– Translate the model flow section into transformation chain specification code, i.e., Ant scripts.

The Launcher executes Ant scripts. This specifies the input models to the transformations, and saves the mapping models into a repository.

In this work we envision AML to specify matching strategies as automatic as possible. Model flow allows to specify no user assistance. The user can nonetheless refine mapping models when a strategy execution ends, and apply strategies on manually refined mapping models. The first AML version provides no constructs for rewriting matching logic (described in section 2). As this logic highly depends on a given domain (e.g., metamodel evolution), more work is needed to determinate whether

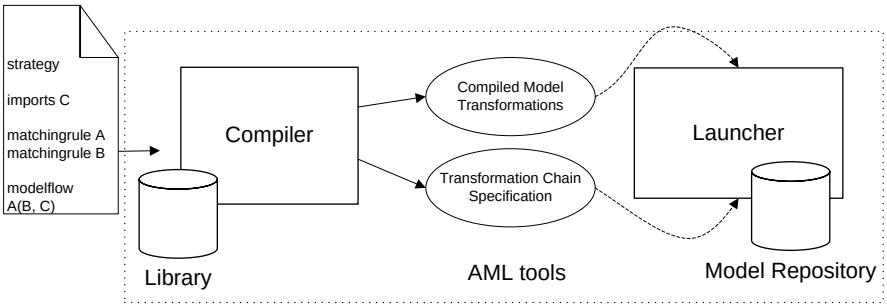


Figure 2. *AML tools*

constructs can factorize its patterns. Although there exist these limitations, AML remains applicable in matching domain.

3.2. *Concrete syntax and semantic*

Listing. 1 gives the overall structure of a matching strategy. A matching strategy needs a name, and contains an import section, a set of matching rule declarations, and a model flow block. Below we present the concrete syntax and informal semantics of the AML constructs.

Listing 1 – Overall structure of a matching strategy

```
strategy S1 {
  imports BasicStrategy;
  uses Propagation(mappingM : MappingMM);
  create TypeMM () {...}
  ...
  sel Threshold () {...}
  modelFlow {...}
}
```

Import section enables to declare AML strategies to be imported. The imported strategies contain matching rules supposed to be reused in a model flow block.

Listing 2 – Import section

```
strategyreflist := (imports strategyref)? ;
strategyref := name | name , strategyref
```

Matching rule declaration specifies the matching rule type, followed by a name, the list of input models (indicating their respective metamodels), the list of libraries to be imported, and the matching rule body.

Listing 3 – Matching rule declaration

```
matchingrule := (create | sim | aggr | sel | uses) name  inputmodellist
    ↪libraryreflist? { matchingrulebody }
```

```
inputmodellist := () | ( inputmodel )
inputmodel := model | model , inputmodel
libraryreflist := uses libraryref
libraryref := library | library , libraryref
```

The developer declares a matching rule using the keywords `create`, `sim`, `aggr`, `sel`, or `uses`. The keyword `uses` declares an external ATL transformation, which should be executed somewhere in the strategy. The developer should only specify the name and list of input models of the ATL transformation. The transformation functionality is therefore defined in a separated ATL module. It is not necessary to specify the mapping model (intended to be manipulated by the matching rule) in the input model list, this remains intrinsic. A library list enables to import ATL libraries containing helpers. A helper can be used to specify operations on model elements (Jouault *et al.*, 2005). In AML, every helper is declared in ATL libraries.

Matching rule body specifies the expressions `inpattern`, `variables`, and `simexp`. An inpattern indicates a set of source types coming from the metamodels specified in a rule input model list. Most of matching rules establish a condition, i.e., an OCL expression (OMG, 2006). When the condition is true, then the mappings are manipulated in some way. In the creation and similarity rules the condition involves the source types and the Mapping type. In the selection rules, in order, the condition involves the similarity value. There is no condition in the aggregation rules. The variables section makes possible to declare a number of local variables. The variables declared in this section can be used in sim expressions. Sim expressions specify similarity functions, these are mandatory in similarity and aggregation rules.

Listing 4 – Matching rule body

```
matchingrulebody:= inpattern? variables? simexp?
variables := using { variableslist }
variablelist := variabledecl | variabledecl ; variablelist
variabledecl := varname : vartype = initexp;
inpattern := intypelist condition
intypelist := (from intypedecl*)?
condition := (when oclexpression)?
intypedecl := varname : type in modelname ;
type := metamodelname ! typename
simexp := is oclexpression
```

Model flow block allows to declare how models flow among matching transformations. A model flow block consists of a set of model variable declarations.

Listing 5 – Model flow section, concrete syntax

```
modelflowblock := (modelFlow { modelvardecl* })?
modelvardecl := (modelname = )? modelflowexp
modelflowexp := (modelexp | transformationcall | weightmodelflowexp )
transformationcall := name parammodelflowexplist
parammodelflowexplist := () | parammodelflowexp
parammodelflowexp := modelflowexp | modelflowexp , parammodelflowexp
weightmodelflowexp := weight : transformationcall | modelexp
modelexp := model
model := modelname : metamodelname
```

A model variable declaration associates a model name to a model flow expression. This can be an input model, a transformation call or weighted model flow expression.

When the model flow expression is a `transformationcall(...)`, the AML system serializes the transformation output as a model conforming to the mapping metamodel.

A transformation call refers to a transformation rule by means of its name. While a transformation rule signature just contains input models, the transformation call indicates the input models and the mapping models to be manipulated. Observe that the developer can refer to input models or output models using a model flow expression or its corresponding model name (if the expression has been previously associated to a model name).

As a final point, weighted model flow expressions allow to associate a weight to a transformation call or to an input model expression (if this refers to a mapping model). An aggregation transformation call should use weighted model flow expressions as parameters.

4. The Motivating Example Using AML

This section presents how to specify the main blocks of a given AML strategy, i.e., model flows and matching transformations.

4.1. *Model flow of strategy S1*

Listing. 6 illustrates the model flow block of the strategy S1. We have omit the S0's model flow due to space constraints, we have taken S1 as a reference because it reuses S0's matching transformations, and it is a more elaborated strategy. The strategy S1 executes 8 MTs. Unlike the transformation TypeMM, the subsequent transformations take the previously produced mapping model as input. Listing. 6 distinguishes different model flow block features. For example, the block contains the two types of model variable declaration. While line 2 declares an additional input model, the rest of lines state matching transformation calls. In line 5, we directly use transformation call as arguments, instead of declaring the transformation call first, and then referring the output models. Whereas it is necessary to declare additional models and mapping models as transformation call parameters (for example, inst and typeMM in line 6), AML keeps the models to be matched intrinsic, e.g., $TerminalModelA$ and $TerminalModelB$ in S0, and $MetamodelA$ and $MetamodelB$ in S1. Finally, the model flow block contains an aggregation MT call (line 8). This notation enables to indicate the importance of similarity values returned by the MTs Normalization and Instances.

4.2. *Matching rules of strategy S1*

We discuss below the main MTs used in the strategies S0 and S1. Each matching transformation has an AML code listing whose number is enclosed between parenthe-

sis. In brief, the main differences between the AML matching transformations and the corresponding ATL versions are :

1) AML constructs hide source and target patterns that respectively specify : a) types of conformance of models to be matched, and b) mapping metamodel concepts. We can refer to them using the constructs thisLeft, thisRight, and thisMapping. The developer uses thisLeft to refer to elements of model A, thisRight to relate to elements b of model B, and thisMapping to refer to mapping elements. In particular, the former constructs allow to develop matching rules without the matter of specifying the metamodel types to be matched. This makes some matching rules generic and applicable to any pair of models.

2) In the AML versions only remain source patterns that queries additional input models, conditions, and functions modifying similarity values.

3) AML provides constructors than factorize code, e.g., Summation, thisWeight, thisSim, thisInstances, thisModel.

Listing 6 – Model flow of strategy S1

```
modelFlow {
  inst = Instance:EqualMM;
  typeMM = TypeMM();
  levMM = EditDistance(typeMM);
  normMM = Normalization(SimilarityFlooding(Propagation(levMM)));
  instanceMM = Instances(typeMM, inst);
  weightedMM = Weighted(0.5:normMM, 0.5:instanceMM);
  thresMM = Threshold(weightedMM);
}
```

TypeMM (Listing. 7) creates a mapping for each pair of model elements having the same type. In contrast to TypeMM, the MT Cartesian Product (in S0) simply specifies true as condition. As a result, we get a mapping for each pair of model elements (without restrictions).

Listing 7 – TypeMM

```
create TypeMM () {
  when
    thisLeft.type = thisRight.type
}
```

EditDistance (Listing. 8) assigns a similarity value to each mapping prepared by the transformation CartesianProduct. The values are computed by the helper editDistance contained in the ATL library Strings. This compares the names of left and right model elements, and returns a value depending on the edition operations that should be applied to a name to obtain the other one. More edition operations, less similarity value.

Listing 8 – EditDistance

```
sim EditDistance uses Strings {
  is thisLeft.name.editDistance(thisRight.name)
}
```

SimilarityFlooding (Listing. 9) propagates previously computed similarity values. This is inspired on Melnik's algorithm (Melnik *et al.*, 2002). Our implementation has two steps. The first step creates an association (i.e., a PropagationEdge) for each pair of mappings (*m1* and *m2*) whose linked elements are related each other. For example, this step associates the mappings `(Class1, Class1')` and `(Attribute1, Attribute1')` because `Class1` of *A* contains `Attribute1`, and `Class1'` of *B* contains `Attribute1'`. The second step propagates a similarity value from *m1* to *m2* because of the relationship. As the first step requires not only mappings but also propagationEdges, we implement it using an external ATL transformation. SimilarityFlooding then takes the external transformation result, and propagates similarity values along mappings. Besides Similarity Flooding, the strategies S0 and S1 execute the MT Normalization which makes similarity values conform to the range [0,1].

Listing 9 – Similarity Flooding

```
1 sim SF () {
2   is
3         thisSim
4         +
5         MappingMM!PropagationEdge.allInstances()
6           ->select(e | e.incomingLink = thisEqual)
7           ->collect(e | e.propagation * e.outgoingLink.sim)
8           ->sum()
9 }
```

Instances (Listing. 10) propagates similarity values from M1-to-M1 mappings to M2-to-M2 mappings. The M1-to-M1 mappings are computed by the strategy S0. These mappings are supposed to be queried but not modified. We use the primitive thisInstances (line 3) to recover, for each M2-to-M2 mapping, e.g., linking the concepts a and b, the M1-to-M1 mapping whose linked elements conforming to a and b. The similarity function (line 5) calculates the average of the M1-to-M1 mapping similarity values. The function reuses the helper maxSim and introduces the helper totalSim (also included in the generic ATL library). This performs an addition of similarity values contained in a mapping set.

Listing 10 – Instances

```
1 sim Instances (instancesModel : MappingMM) {
2   using {
3     inst : Sequence(MappingMM!Mapping) = thisInstances(instancesModel);
4   }
5   is (maxSim(inst) * totalSim(inst)) / inst->size()
6 }
```

Threshold (Listing. 11) selects mappings with a similarity value higher than a given threshold. Line 2 specifies this condition.

Listing 11 – Threshold

```
1 sel Threshold () {
2   when thisSim > 0.7
3 }
```

Weighted (Listing. 12) computes a weighted sum of similarity values of mapping models. This needs relative weights associated to the models. In strategy S1, Weighted takes the mappings returned by the MTs Normalization and Instances, and their corresponding weights, i.e., 0.5–0.5 (see line 7 of Listing. 6). In Listing. 12, the expression Summation adds the similarity values (specified as thisSim), and multiplies them with weights (specified as thisWeight).

Listing 12 – Weighted

```
1 aggr Weighted () {
2   is Summation(thisSim * thisWeight)
3 }
```

5. Experimentation

This experimentation has three goals : 1) express robust matching strategies using AML, 2) apply matching transformations to metamodels and terminal models, and 3) evaluate the accuracy rendered by AML matching strategies.

5.1. *Prototype*

We have implemented the prototype on the AtlanMod Model Management Architecture (AMMA) platform (Kurtev *et al.*, 2006). More specifically, we have implemented the AML concrete syntax using the Textual Concrete Syntax (TCS) tool (Jouault *et al.*, 2006b). TCS generates a parser that converts an AML program from text to model format. We have developed the AML compiler by means of ATL code (of approximately 2300 lines). This code translates an AML model into : 1) a set of ATL matching transformations, and 2) an Ant file instrumenting the transformation execution chain. Additionally, we have used the AtlanMod Model Weaver (AMW) (Didonet Del Fabro *et al.*, 2005) to graphically manipulate the mappings models. We ran the benchmark on a PC with a 2.4 GHz Intel Core 2 Duo processor, 1GB of RAM, and Windows XP OS version 2002.

5.2. *Procedure and dataset*

We have implemented the strategies S0 and S1 using AML. The strategy S1 has been applied to the pair of metamodels Families and Persons (Eclipse.org, 2009), which respectively include the classes `Family-Member`, and `Person-Male-Female`. The strategy S0 in turn has been applied to terminal models conforming to these metamodels. The Families terminal model represents, for example, the family March. We have obtained the Persons terminal model by applying the transformation Families2Persons to the Families terminal model. The models have 8 elements on average. The selection of the models has been driven by three main reasons : 1) we can figure out the expected (correct) mappings from the transformation, 2) the mappings

Tableau 1. *Generated ATL code*

Matching transformation	Number of lines
CartesianProduct	90
TypeMM	90
EditDistance	75
Instances	80
Normalization	73
SimilarityFlooding	86
Threshold	75
Weighted	79

Tableau 2. *Accuracy measures*

Measure	Value
Precision	0.4
Recall	0.3
Fscore	0.38

are simple but interesting enough to be analyzed in a paper, and 3) the metamodels and terminal models are available as open-source.

5.3. *Results and discussion*

Generated code. Table. 1 shows the number of lines generated from each matching rule, involved in the strategies S0 and S1. The Ant Script contains 124 lines. The AML compiler generates this code in 4 seconds.

Although this study has been primarily applied on eight matching transformations and two strategies, it should be noted that AML considerably reduces the number of lines of code to be written. Comparing the number of lines of an AML transformation and its corresponding generated code, we find out that AML saves the codification of 80 lines of ATL code, and 10 lines of Ant code. We hope to implement more validations in future.

Accuracy. We have evaluated AML strategies accuracy using three metrics (Rijsbergen, 1979) : $Precision = \frac{CorrectFoundMappings}{TotalFoundMappings}$, $Recall = \frac{CorrectFoundMappings}{TotalCorrectMappings}$, and $Fscore = \frac{2*Recall*Precision}{Recall+Precision}$. The expected values of these metrics are between 0 and 1. The higher is the precision value, the smaller is the set of wrong mappings. The higher is the recall value, the smaller is the set of the mappings that have not been found. Fscore is a global measure of the matching quality. A high fscore value indicates a matching of high quality.

Table. 2 shows the accuracy values of S1, which have been influenced by the values of S0. In general, the values show that S1 has a relatively slow accuracy, matching the metamodels Families and Persons. Taking fscore as an example, the percentage of correct mappings is 38%. This value includes (for example) the correct mapping `(Member, Male)`. Based on these results, we have concluded that while the strategy S1 is robust enough, it is nevertheless inappropriate to find out mappings between the metamodels of our motivating example. The main reason is that the matching transformation Instances, which has an important percentage of confidence in S1 (50%), renders wrong matches. This is not associated to bugs in the code, but to the nature of instances-based heuristics, which render best results in a few particular cases (Doan *et al.*, 2001). The results thus suggest the need of a more suitable strategy to match the pair of metamodels Families and Persons. Unfortunately, we are unable to determinate from this data the correctness of the AML compiler. A way to do that may be to compare the accuracy of manually-developed transformations to transformations generated by the AML compiler. It seems a interesting work to do in further researchs.

6. Related works

There has been many previous work on the matching problem. In this section, we focus on the related works closer to MDE. We should nonetheless mention the works described in (Wang *et al.*, 2008)(Melnik *et al.*, 2002)(Do, 2005). They present robust strategies, that match schemas/ontologies, which have been implemented using GPLs. The strategies and reported lessons have inspired the conception of the AML constructs in some way.

In the MDE context, (Kolovos *et al.*, 2008) proposes a DSL, named Epsilon Comparison Language (ECL), for specifying model comparison algorithms. (Fleurey *et al.*, 2007)(Rubin *et al.*, 2008) in turn propose frameworks for model composition, i.e., matching and merging. We now compare these approaches and AML taking into account their contributions on model matching :

– Unlike the approaches described in (Fleurey *et al.*, 2007) and (Rubin *et al.*, 2008), AML consider not only homogeneous models but also heterogeneous, i.e., models conforming to metamodels different each other.

– Whereas ECL and (Fleurey *et al.*, 2007) offer only constructs to express similarity between two models to be matched (direct comparison). AML provides a richer set of concepts for expressing complete matching strategies. This includes indirect comparison strategies.

– (Fleurey *et al.*, 2007) and (Kolovos *et al.*, 2008) enable to express comparison algorithms that render binary values (0 or 1, equal or not equal). AML instead applies the notion of probabilistic mappings, which is closer to real world mappings.

– AML promotes matching heuristics not coupled to the models to be matched. The constructs thisLeft and thisRight allow to declare matching transformations without the matter of distinguishing metamodel types. This is improvement on (Fleurey

et al., 2007)(Kolovos *et al.*, 2008)(Rubin *et al.*, 2008) which are metamodel type-based.

EMF Compare is an Eclipse.org tool for model comparison. Its matching strategies are hard-coded into the tool in Java. This prevents the customization of matching strategies necessary in many context, and supported by (Fleurey *et al.*, 2007), (Kolovos *et al.*, 2008), (Rubin *et al.*, 2008), and AML.

(Falleri *et al.*, 2008) automatically detect mappings between two metamodels using the algorithm *Similarity Flooding* described in (Melnik *et al.*, 2002). The algorithm is generic enough to match only metamodels. Remark on our AML version of the *Similarity Flooding* algorithm is applicable to metamodels and terminal models.

After having compared AML to other model comparison approaches, we compare its rule orchestration part to the Uniti (Vanhooff *et al.*, 2007) transformation chain definition framework. While Uniti is more declarative framework for transformation chain definition, AML constructs have been tailored for the representation of matching strategies in a more concise manner. Note that although the AML orchestration syntax is currently compiled into Ant tasks, it could also be compiled into Uniti. This design choice does not impact the user, and we simply chose the alternative we know better.

Although this section has concentrated on AML and the closer related languages, we should refer to (Steel *et al.*, 2007) as an approach dealing with the problem of coupling transformation and model representation. Instead of single-type based transformations, they propose a simple system based on a collection of interconnected types. Notwithstanding this approach augment the reuse of transformations, transformations remain brittle and restricted to the pre-defined type list. In contrast to (Steel *et al.*, 2007), AML leverages transformation reuse by means of constructs that hide model types at implementation time. AML compilation strategy therefore resolves the appropriate types at compilation time.

7. Conclusion

The main purpose of this paper has been to present AML, a DSL for expressing matching strategies. We have validated the DSL expressiveness by implementing two robust matching strategies, and eight matching transformations. The results are overall motivating. Firstly, we have developed matching transformation applicable to metamodels and terminal models. AML likewise saves the implementation of a significant amount of lines of code, i.e., 80 lines of ATL code, and 10 lines of Ant code (for each matching rule). The AML constructs moreover factorize code that make the strategies more readable. Thus, matching developers may analyze strategies, and modify them in order to improve mapping accuracy. From the results, it is also clear that more validations are needed, e.g., to implement more strategies, matching transformations, and to apply them to different pairs of models. Other further research may be : to compare the solutions to the problem of coupling between strategy and model representation, and to investigate constructs to express rewriting logic and matching user assistance.

8. Acknowledgements

This work has been partially funded by the ANR project FLFS.

9. Bibliographie

Cicchetti A., Ruscio D. D., Eramo R., Pierantonio A., « Automating Co-evolution in Model-Driven Engineering », *EDOC '08 : Proceedings of the 12th IEEE International EDOC Conference*, München, Germany, 2008.

Didonet Del Fabro M., Bézivin J., Jouault F., Breton E., Gueltas G., « AMW : A generic model weaver », *Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM05)*, 2005.

Do H. H., Schema Matching and Mapping-based Data Integration, PhD thesis, University of Leipzig, 2005.

Doan A., Domingos P., Halevy A., « Reconciling schemas of disparate data sources : A machine-learning approach », *In SIGMOD Conference*, p. 509-520, 2001.

Eclipse.org, *ATL Example Presentation, Families to Persons*, `http://www.eclipse.org/m2m/atl/basicExamples_Patterns/`. 2009.

Euzenat J., Shvaiko P., *Ontology Matching*, Springer, Heidelberg (DE), 2007.

Falleri J.-R., Huchard M., Lafourcade M., Nebut C., « Metamodel Matching for Automatic Model Transformation Generation », *in* K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, M. Völter (eds), *MoDELS*, vol. 5301 of *Lecture Notes in Computer Science*, Springer, p. 326-340, 2008.

Favre J.-M., « CacOphoNy : Metamodel-Driven Architecture Recovery », *WCRE*, IEEE Computer Society, p. 204-213, 2004.

Fleurey F., Baudry B., France R. B., Ghosh S., « A Generic Approach for Automatic Model Composition », *in* H. Giese (ed.), *MoDELS Workshops*, vol. 5002 of *Lecture Notes in Computer Science*, Springer, p. 7-15, 2007.

Garcés K., Jouault F., Cointe P., Bézivin J., Adaptation of Models to Evolving Metamodels, Technical report, INRIA, 2008.

Girschick M., Difference Detection and Visualization in UML Class Diagrams, Technical report, TU Darmstadt, 2006.

Javed F., Mernik M., Gray J., Bryant B. R., « MARS : A metamodel recovery system using grammar inference », *Information & Software Technology*, vol. 50, n° 9-10, p. 948-968, 2008.

Jouault F., Bézivin J., « KM3 : a DSL for Metamodel Specification », *Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037*, Bologna, Italy, p. 171-185, 2006a.

Jouault F., Bézivin J., Kurtev I., « TCS : a DSL for the specification of textual concrete syntaxes in model engineering », *in* S. Jarzabek, D. C. Schmidt, T. L. Veldhuizen (eds), *Generative Programming and Component Engineering, 5th International Conference, GPCE 2006, Portland, Oregon, USA, Proceedings*, ACM, p. 249-254, 2006b.

Jouault F., Kurtev I., « Transforming Models with ATL », *Proceedings of the Model Transformations in Practice Workshop, MoDELS 2005*, Montego Bay, Jamaica, 2005.

Kolovos D. S., Paige R. F., Rose L. M., Polack F. A., *Epsilon*, `http://epsilonlabs.svn.sourceforge.net/svnroot/epsilonlabs/org.eclipse.epsilon.book/EpsilonBook.pdf`. September, 2008.

Kurtev I., Bézivin J., Jouault F., Valduriez P., « Model-based DSL Frameworks », *Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, OR, USA*, ACM, p. 602-616, 2006.

Melnik S., Garcia-Molina H., Rahm E., « Similarity Flooding : A Versatile Graph Matching Algorithm and ist Application to Schema Matching », *Proc. 18th ICDE*, San Jose, CA, 2002.

Melnik S., Rahm E., Bernstein P., « RONDO - A Programming Platform for Generic Model Management », *Proc. ACM SIGMOD Intl. Conf. Management of Data*, p. 193-204, 2003.

OAEI, *Ontology Alignment Evaluation Initiative*, `http://oaei.ontologymatching.org/`. 2008.

Ohst D., Welle M., Kelter U., « Differences between versions of UML diagrams », *SIGSOFT Softw. Eng. Notes*, vol. 28, n° 5, p. 227-236, 2003.

OMG, *OCL 2.0 Specification, OMG Document formal/2006-05-01*, `http://www.omg.org/docs/ptc/05-06-06.pdf`. 2006.

Rijsbergen C. J. V., *Information Retrieval*, Butterworths, 1979.

Rubin J., Chechik M., Easterbrook S. M., « Declarative approach for model composition », *MiSE '08 : Proceedings of the 2008 international workshop on Models in software engineering*, ACM, New York, NY, USA, p. 7-14, 2008.

Steel J., Jézéquel J.-M., « On model typing », *Software and System Modeling*, vol. 6, n° 4, p. 401-413, 2007.

Treude C., Berlik S., Wenzel S., Kelter U., « Difference computation of large models », *in* I. Crnkovic, A. Bertolino (eds), *ESEC/SIGSOFT FSE*, ACM, p. 295-304, 2007.

Vanhooff B., Ayed D., Baelen S. V., Joosen W., Berbers Y., « UniTI : A Unified Transformation Infrastructure. », *in* G. Engels, B. Opdyke, D. C. Schmidt, F. Weil (eds), *MoDELS*, vol. 4735 of *Lecture Notes in Computer Science*, Springer, p. 31-45, 2007.

Wang T., Pottinger R., « SeMap : a generic mapping construction system », *in* A. Kemper, P. Valduriez, N. Mouaddib, J. Teubner, M. Bouzeghoub, V. Markl, L. Amsaleg, I. Manolescu (eds), *EDBT*, vol. 261 of *ACM International Conference Proceeding Series*, ACM, p. 97-108, 2008.

Xing Z., Stroulia E., « UMLDiff : an algorithm for object-oriented design differencing », *ASE '05*, ACM, New York, NY, USA, p. 54-65, 2005.

From Business Processes to Integration Testing

A public transport case study

Stéphane Debricon [*] — Fabrice Bouquet[*] — Bruno Legeard[**]

[*] *LIFC*
Université de Franche-Comté
16 Route de Gray
25030 Besancon Cedex
{stephane.debricon,fabrice.bouquet}@lifc.univ-fcomte.fr

[**] *Smartesting*
TEMIS Innovation
18 Rue Alain Savary
25000 Besancon
legeard@smartesting.com

RÉSUMÉ. *Cet article se place dans le domaine de la génération de tests à partir de modèle (MBT). Celle-ci vise à fournir des campagnes de tests pour assurer la robustesse, l'adéquation aux besoins ou la sécurité d'une application. Dans la pratique, l'équipe en charge de la validation produit un modèle pour chacun des blocs fonctionnels présents dans le système à tester. Ce modèle est alors une source d'informations à prendre en compte pour produire de nouvelles campagnes de tests sur le système complet.*

Pour cela, nous ajoutons un nouveau niveau d'abstraction permettant la représentation de comportements métiers en utilisant les blocs fonctionnels déjà modélisés. En les associant et en ajoutant les objets échangés par ceux-ci, nous pouvons décrire des processus métiers qui n'étaient pas encore capturés aux niveaux des blocs.
Dans ce travail, nous proposons une approche méthodologique qui s'intègre au processus unifié (UP) et la gestion des conflits induits par la recombinaison des blocs. Nous illustrons notre proposition en l'appliquant à un cas industriel: un système de vente de titre de transport en commun.

ABSTRACT. *This paper is a contribution to the Model-Based Testing (MBT) field. MBT aims at producing tests suites that will be used to check for security, robustness or correct software*

adequacy with requirements expressed by customers. In practice a validation team produces a model for each functional block involved in a system under test. Thereby, the model gathers all kind of information that can be used to produce new tests suites for the complete system.

Therefore, we introduce a new abstraction level, allowing the description of new business behaviours using functional blocks already modeled. Those blocks are combined and augmented with objects exchange, producing business processes not captured at the block level. In this work, first we introduce a methodological approach fully integrated in the Unified Process (UP) and then a conflicts resolution due to block recombination. We illustrate our proposal with an industrial case study: a transit passes selling system.

MOTS-CLÉS : *Test à partir de modèles, Processus métiers, Test d'intégration, BPMN, UML*

KEYWORDS: *Model-Based Testing, Business Process, Integration Testing, BPMN, UML*

---

1. Introduction

Model-Based Testing (MBT) (Utting *et al.*, 2006b) aims at producing tests suites to check for security, robustness or correct software adequacy with requirements expressed by customers. This technology based on UML was used in our previous research to ensure functional coverage of software development (Utting *et al.*, 2006a). It was integrated in a development life cycle called Unified Process (UP) (Arlow *et al.*, 2003) to help its adoption and spreading.

In consideration of a practical application of this method, a development and validation team would produce a model containing functional components representing the system under test. Thereby, the model gathers all kind of information that can be used to reach a new step in tests generation. It contains behaviors, data and tests allowing non-regression testing. We propose to combine all those information to generate new tests suites. Those tests are based on business processes defined from model's elements. As a consequence, we introduce a new abstraction level, allowing the description of new behaviors not yet represented in the model.

Business processes can be seen as activities (or actions) sequences, that might not be described in initial requirements, but are known by experts to be interesting behaviors. Processes can be written by analysts from the validation team (as described in the method introduced in (Bouquet *et al.*, 2006)) or by domain experts. A business process is viewed as an integration test of model's components and is fully integrated in the testing process. A process is described by combining components already modeled, augmented with data information spreading from an activity to an other, to create an integration test. Our goal is not to automatically generate a business process, but rather to give the opportunity for an expert to model a specific and interesting behavior. However, a test model should be available for every components used in the business process, and we believe that this element's composition will produce tests suites for integration testing.

Our intention is to guide a business specialist or an analyst to describe business processes, which cannot be found in analysis model and to generate tests suites from them. Those tests complement tests already produced by automatic tests generation from behavioral models. Modelers can describe a business process or an integration test. A business process can be seen as an integration test since it models a sequence of behavior along with data flow.

The rest of this paper is organized as follow : Section 2 presents the methodology for modeling a business process from existing elements in the model. In this section this approach is applied on the case study. Section 3 details the production of business processes tests suites from the model. The section presents the initial state derivation and how to deal with data flows and control nodes of the business process activity diagram. Section 4 gives an overview on the tests generation technique and the number of tests produced for a simple example.

2. Business process modeling methodology

In this section we describe how to handle a business process and information required for its modeling in an UP analysis phase.

2.1. *Approach*

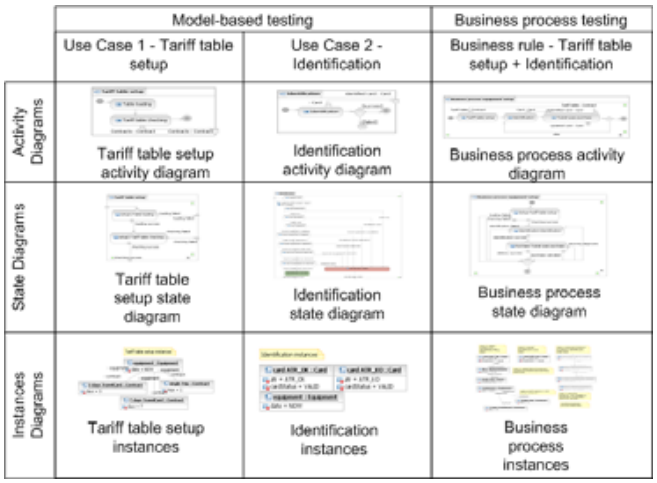


Figure 1. *Model elements reuse*

Usually analysis phase is split in the four following steps (Arlow *et al.*, 2003) : i) use cases modeling, ii) classes and objects detection, iii) relationships between classes and iv) objects definition and use cases realization. We propose to introduce business process in the model available at the end of the last step, linking use case realization with each other. Model also includes data flowing between use cases. From now on, we focus on use case realization. Each use case identified by previous steps of analysis phase are refined with a sequence diagram describing nominal behavior. This sequence diagram should come along with an activity diagram describing use case general behavior. The diagram contains every actions required for use case realization, it also underlines objects flowing from an action to an other. Furthermore, actions introduced in activity diagram, are refined into state machines. Those machines contain use case behaviors expressed by means of states and transitions augmented with OCL code. By the end of this phase, the model should contain activity diagrams for each use case previously identified. At this point our approach introduce business processes modeling. Business activity diagrams should link use cases with each other, through the reuse of their respective activities. Each activity diagram describing a business process is made of references to activities (use cases) of the model. Furthermore, the model will bring to light all data flowing between activities.

Activity diagrams will be used to represent the control flow of business processes and data flows between activities defining them. Activities themselves are described in terms of state machines which can be used for automatic tests generation as presented in (Bouquet *et al.*, 2007). It remains to define an initial state based on data flows expressed in the business process activity diagram. The new business process is created from elements already defined in the model produced for model-based testing. The approach is summarized in Figure 1.

2.2. *Practical application : Transit passes selling software*

The approach described in the previous sub-section is applied on a transit passes selling machine real-life example. This example is the result of the project VALMI (validation of embedded micro-systems) which was carried out by Parkeon (http ://www.parkeon.com/), ERG Transit System (http ://www.erggroup.com/), Smartesting (http ://www.smartesting.com/) and LIFC (http ://lifc.univ-fcomte.fr/). Considering an urban transit system, a customer can buy a transit pass from a vending machine. Transit passes sold by the machine are stored in a contact-less smart card. Delivery steps are described in a standard document called Intercode II (French norm XP P 99-405 / European norm ENV1545), that contains information and processes to be used by urban project stakeholders. Delivery steps are as follow : contact-less smart card detection and identification and stored diagnostics handling. Diagnostics are special events created during the card life-cycle (for example : card invalidation due to dysfunction or e-selling from the back-office). Then the equipment starts the delivery activity that includes the choice of a transit pass, a quantity and payment. The complete model was produced using Rational Software Architect by industrial partners with the help of the LIFC.

At first use cases of the selling application were identified. In this paper, we only consider two use cases : transit pass purchase and tariff table setup.

2.2.1. *Activity diagrams*

Two use cases of the application are selected and refined in this section. Activities are illustrated in Figure 2 and 3. We use two notions of the activity diagram ie the control flow to describe the behavior of the activity. In addition, we use the concept of data flow expressing the sharing of information from one activity to another.

Figure 2 shows an activity made of an action called *Identification*. This activity also presents the smart card processed as an output parameter, this data flow can be used to connect the activity to another one. There are two possible final control flows : *failed* or *success*. Only one of those can be connected to an other activity, i.e. the activity final node (success), the flow final node (failed) cannot be connected to any activity.

Activity presented in Figure 3 contains two actions required to complete tariff table setup. At first the tariff table should be loaded into the equipment, then it should be

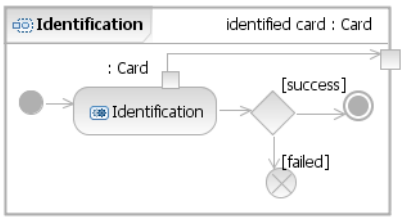


Figure 2. *"Card identification" activity*

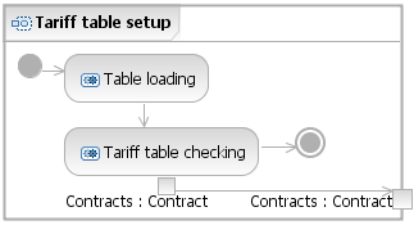


Figure 3. *"Tariff table setup" activity*

verified by an technical operator. Once this checking is complete, the tariff table will be available for any other activity as a set of *Contract*.

The third activity (not illustrated) is the transit pass delivery, it is composed of two actions : *Diagnostics handling* (no details given in this paper) and *Delivery*. The activity has three parameters : two input parameters i.e. the smart card (detected by the equipment) and the tariff table (available in the machine). The smart card updated with the new transit pass purchased is an output parameter. The smart card is an activity's object which state may be changed by activity's actions. This activity contains an initial node and two final nodes, nominal case is represented by the control flow ending with the activity final flow. There is also an error case represented by a flow final node, stopping the activity.

2.2.2. *State machines*

Behaviors modeled in activity diagrams are described more precisely with state machines.

Figure 4 shows a state machine describing every steps required to setup the equipment, that is tariff table loading and checking. The first step is represented in Figure 5. This state diagram provides complementary information to activity diagram in Figure 3. It shows entry and exit pseudo-states to be used to model a business process. Thereby there are three exit pseudo-states : *loading failed*, *checking failed* and *checking success*.

2.2.3. *Business processes*

We have enough information in the model to define a business process. We focus on a process managing an equipment setup before its use. Figure 6 shows a business process activity diagram that gather activities previously introduced, augmented with data flows between activities.

A business process activity diagram contains a control flow describing the way activities are called. The sequence will be as follow : flow starts by business process initial node, then activates the first activity initial node. At the end of the current ac-

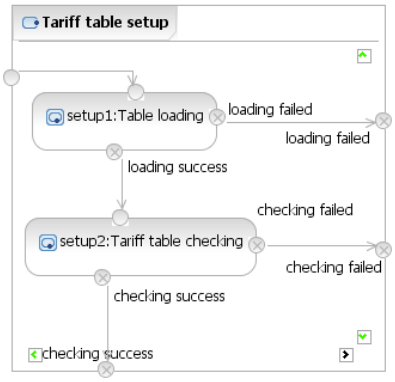


Figure 4. *"Tariff table setup" state machine*

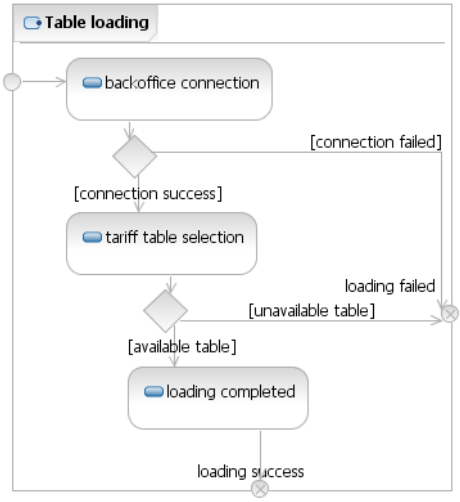


Figure 5. *"Tariff table loading" state machine*

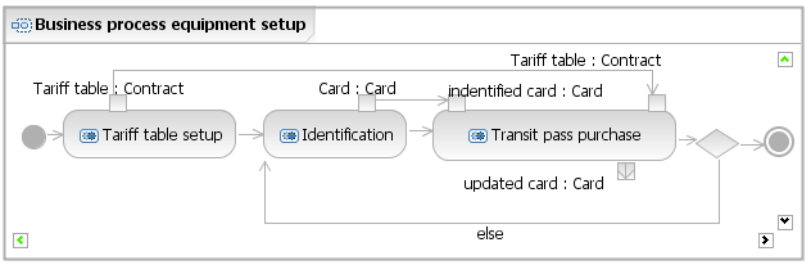


Figure 6. *Business process "Equipment startup and use" activity*

tivity (through activity final node and not any flow final nodes) the flow activates the next activity and so on. On Figure 6, process starts with activity called *Tariff table setup* then *Identification* followed by *Transit pass purchase*. The process goes back to the *Identification* activity waiting for a new card. Scenario ends at equipment shutdown.

The activity diagram also contains data flows between activities. Thereby *Tariff table setup* activity provides tariff table to *Transit pass purchase* activity. *Identification* activity describes smart card detection and recognition of a card used by *Transit pass purchase* activity. This card is not used in any other business process actions, as it is

not relevant for any situation described by the process and there is no need to spread this information.

We introduced a method based on the presence in the model of all necessary elements to create a business process (bottom-up method : from detailed elements to business process). It is also possible to produce all elements from the business process i.e. to describe the process then to detail step by step all behaviours as seen in previous section (down method : from business process to detailed element). A test generation is possible as soon as machines and initial states are produced. We recommend to reuse all knowledge included in the model but both methods (bottom-up or down) can be used to create business processes. A good practice is to mix both of them, that is to use pre-existing elements and to describe missing behaviours.

Providing data exchanged between activities is a key concept for tests generation. We will address in the next section key problems of exchange between activities : data sharing(exchange and initial states), control nodes and tests generation.

3. Tests suites generation for integration testing

In previous section, we have seen how to model a business process using activity diagrams augmented with data exchanged between activities. Scenarios are extracted from the diagram, to test correct integration of activities within the application.

Using the business process activity diagram, we produce a new state machine made of machines composition. The resulting machine is obtained by connecting several sub-machines already defined in the analysis model. Machines entry and exit points should be connected to create a new machine that will model system behavior within a business process. This splitting phase underlines and links several sub-machines connection points, particularly in the case of a machine with several exit points. Data exchanged by activities are not represented within resulting state machines but are only modeled in activity diagram as seen in the previous section. Thus each activity designed for MBT should come with a state machine, an initial state made of instances and a common class diagram.

We are not trying to introduce any automatic composition of state machines, the composition process should be dedicated to an analyst. His task should be to use any information available in the model to create a new diagram guided by business process. Figure 7 shows sub-machine *Tariff table setup* described in Figure 4, with two exit points connected to sub-machines : *Identification* and *Transit pass purchase* (no details of those machines given here).

3.1. *Deriving business process initial state*

The state machine introduced in previous section describes expected functional behavior of the system when activating a business process. The machine is made of

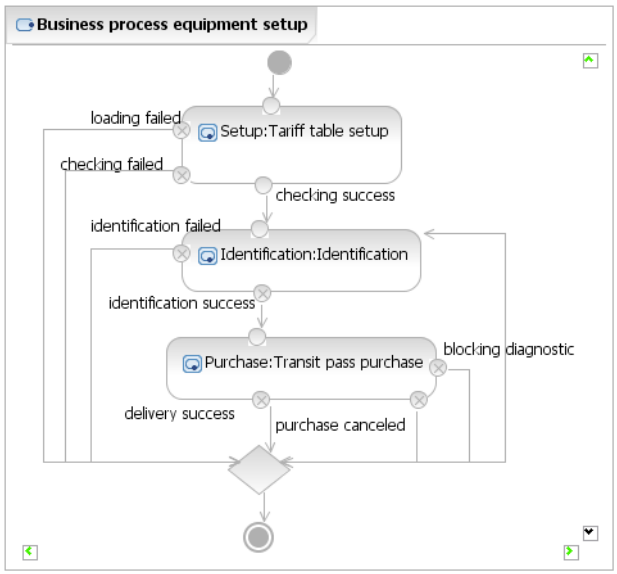


Figure 7. *Business process "Equipment startup and use" state machine*

sub-machines for which tests suites may be available. Those suites should have been independently validated. They come with an initial state made of instances which evolution, through the computation of transition post-conditions, define the expected verdict. Once we have connected our sub-machines, a new initial state must be specified using every sub-machines initial state. Furthermore, information available on business process activity diagram allow us to identify objects we want to retain from an activity to an other, so from a machine to an other.

Choosing values for business process initial state : Every sub-machine composing a business process has its own initial state (Figure 8, 9, 10). A new initial state made of states combination has to be created for the process state machine, we examine every instance diagram to identify instances and slot values to be retained in resulting initial state. To do so, instances must keep the same name (identifier) in the model. That is to say, an instance describing a specific contract, for instance, should have only one identifier. There are three kinds of instances in the model : system under test class instances, spread classes instances (used in business process activity diagram) and all other classes.

System under test class instances are shared by all initial states of the model. We must compare *slot values* in all instance diagrams to define which value to retain for the new initial state :

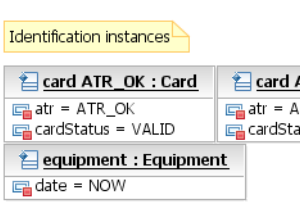


Figure 8. *"Identification" initial state diagram*

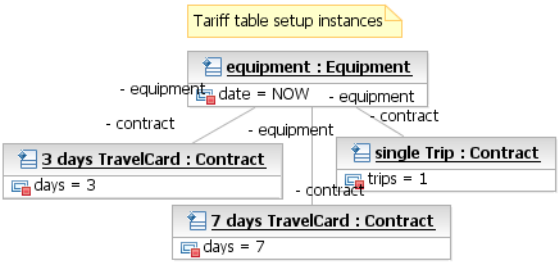


Figure 9. *"Tariff table setup" initial state diagram*

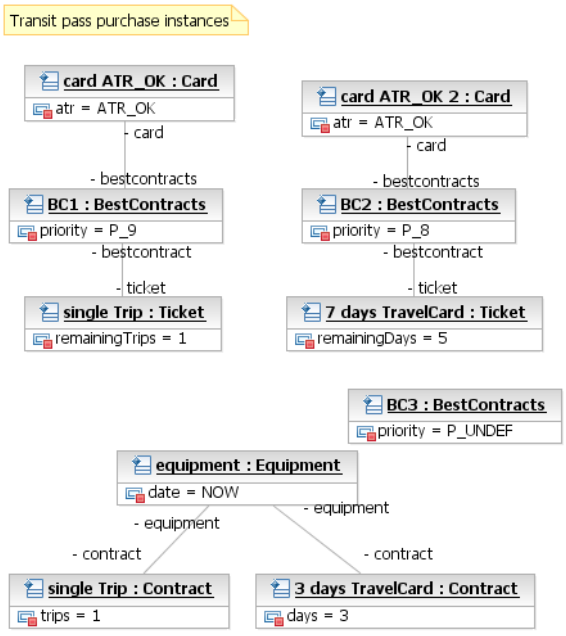


Figure 10. *"Transit pass purchase" initial state diagram*

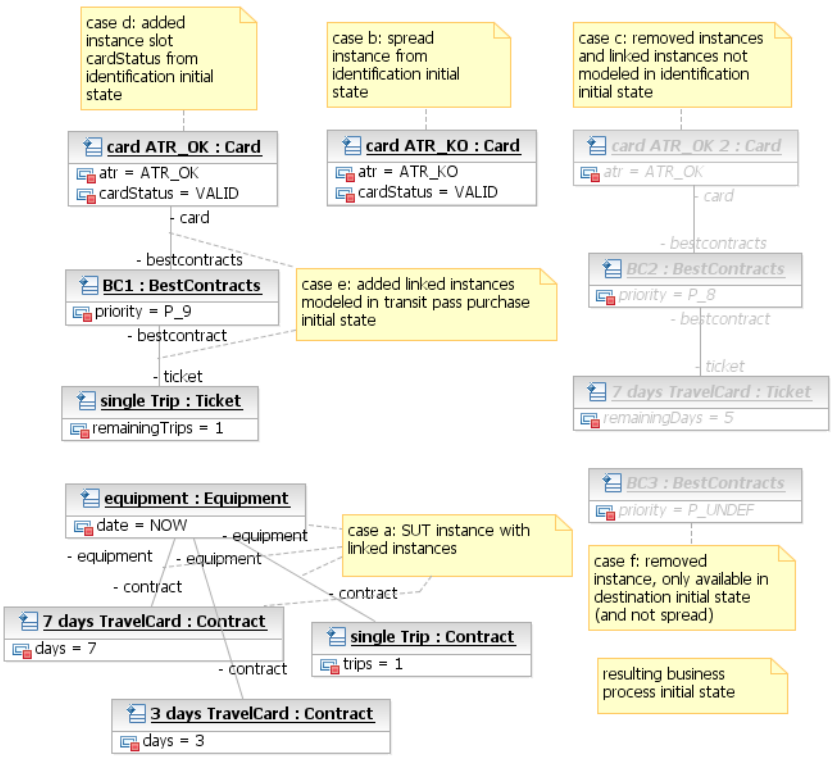


Figure 11. *Resulting initial state diagram*

– If all **values are equal in every initial states**, we keep those values.

– **If that's not the case**, we must specify how to decide value to be retained. Non instantiated attributes in source initial states will be given an arbitrary value during tests concretization, thus will not appear in resulting initial state.

– **Attributes existing in several diagrams** but with only one significative value and all other not instantiated, will remain in resulting initial state with the significative value.

– In the worst case, that is **multiple significative values different in all diagrams**, we have to choose the most significative value in the business process context. This choice can only be made by a domain expert.

We now have a system under test class instance that we should be completed with every linked instances available in initial states, that will be added to the resulting initial state (Figure 11 case a).

Classes of spread instances must not have any setter (an operation that will lead a generator to automatically defines instances values). Calls to those operations will canceled any modeling effort made in business process activity diagram to underline spread instances. Those specific instances have to be handled differently. We must take into account both source and destination initial states (instances are spread from one activity to an other, i.e. a machine to an other as shown in Figure 6). There are three alternative :

1) All instances available in source state are directly used in resulting state (Figure 11 case b).

2) An instance of destination initial state not present in source initial state is not used in resulting state, excluding also linked instances (Figure 11 case c).

3) An instance available in both source state and destination state is used in resulting state with values extracted from initial state (Figure11 case d).

Links between instances are preserved as expressed in source initial state, eventually augmented with links modeled in destination initial state. A comprehensive navigation through diagrams links is mandatory to ensure a complete instances processing (Figure 11 case e). This implies instances functional coherence in object diagrams.

All other classes, not modeled in business process activity diagram, are insignificant in this processing and those instances are not retain in resulting initial state (Figure 11 case f).

3.2. *Dealing with control nodes*

There are two types of control nodes in activity diagrams : fork and decision. We focus on these two types of control nodes. The other two ie merge and join have no impacts on the creation of the initial state. Both have restrictions and characteristics that have to be examined when building initial state.

Through a fork node, an activity starts two concurrent control flows. Objects can pass from activity to activity within those parallel flows. If we take a look at object spreading from a parallel portion to a non parallel one, we identify two cases :

– *two control flows (or more) and two (or more) instances of* ***different classes*** is the trivial case, all instances are used in resulting state.

– *two control flows (or more) and instances of the* ***same class***. This case is only possible when dealing with different instances of the same class. The same instance cannot be used by two parallel flows, that would create a non deterministic model useless for tests generation.

A decision node introduces at least two possible control flows. Just like fork node, a case where different instances spread from a branch to the main control flow, is quite simple. Instances from source initial state are used in resulting initial state. But if an instance spread from one or more branches to the main one, and has equal attributes

values whatever the branches, this instances will be added to resulting initial state. Different attributes values implies that this instance will be cloned into resulting state. A new clone will be created for each new value. Then we will add all links and linked instances available in diagrams.

By the end of this composition phase, we have added to the model, a new state machine, made of sub-machines, a class diagram and a new initial state. All those information will be used in combination with a class diagram, to generate new tests suites that cover business process.

4. Business process test suites generation

In one hand, Business process test generation can be done directly from the expression of this business process such as expressed in (Z. J. Li *et al.*, 2008), where BPEL is used to describe the process. A specific test generation tool is able to cover execution path and produce non-regression tests. In the other hand, we can a UML model. We decide to reuse UML model produced during earlier phases. We can generate tests the same way we would have processed a single state machine describing a simple behavior and using model-based testing. We used Smartesting Test Designer to identify targets from the model and generate tests to cover business process functional behaviors. The tool unfolds the new state machine augmented with the new initial state. We thus obtain logical tests that must be concretized to be executable. It is the publication phase that will transform our logical tests into Junit or Fitness tests. The published tests have to be automated. To do so control points are implemented. Then mapping between the data of the initial state or the verdict must be done to interact with the application to be tested. Thus each state machine should have tests suites to singly guarantee its coherence in regard with requirements. We introduce a new set of tests ensuring a good integration of those separate modules when used by a business process. We produce new integration tests that should not be confused with tests produced to maintain functional coverage.

Business process generated tests offer the benefit of covering a larger functional pane, taking into account data that are known to be business specific. Unreachable targets can be caused by the new initial state, this reveals that a business process may not use every module functional behaviors. Other unreachable targets caused by data exchange between activities can be detected.

Table 1 presents the results of test generation for the business process shown in Figure 12. We can see the 3 simple behaviors and results in terms of requirements, targets and generated tests. For the identification activity, the tool identifies 17 targets corresponding to transitions to activate the state machine. It generates 11 tests to cover these 17 targets and 8 requirements involved. For the 3 simple behaviors we have the same number of tests as we did when generating them independently. The interest lies in the use of a common initial state. The other strong point is the modeling of 3 new behaviors introduced by the user's type dealing with the system. We produce 3 tests, 2

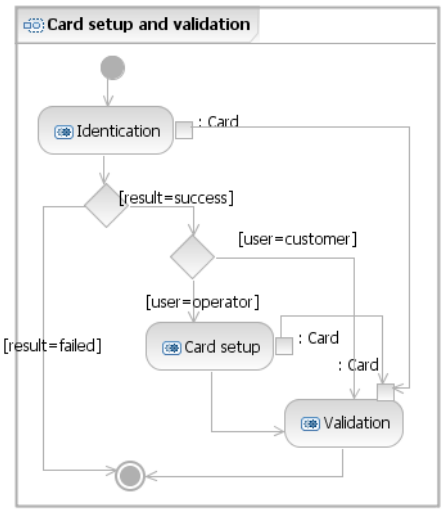


Figure 12. *Card setup and validation business process*

Behaviours	Requirements	Targets	Tests
Card Identification	8	17	11
Card setup	2	2	2
Validation	14	31	20
Business process	0	3	3

Tableau 1. *Results on tests suites generation*

are new tests and the third one is a test from *Identification* failed activity. We can also note that there were no business requirements expressed in the specification, and only recovered requirements from previous activities are available.

A great advantage of this method is to easily produce runnable tests, and though we do not have to deal with combinatorial explosion. That would be the case, if we were automatically connecting every exit points with every entry points of our activities. This would be an interesting feature for robustness testing, but we only want to offer more functional testing of a system. And a better way to achieve it, is to give the opportunity to a business analyst to express useful business behaviours. We produce integration tests considering we are composing behaviours, in addition with data navigating from one activity to another. These data represent a major interest because they allow to define new use case, and ensure a better integration of components with each other. Another advantage is the use of the new initial state by tests already produced

at the activities level. The 33 tests may use any available initial state data. Therefore, some behaviours may not be accessible with these data that can be explained by the fact that a business process does not activate every behaviours available.

5. Conclusion

This work is directly linked to Smartesting technology but can be easily applied to other techniques such as (Briand *et al.*, 2001) and (Nebut *et al.*, 2006). Model-based testing methodology and technology play a large role to ease functional validation of developments, with a permanent concern for software quality improvements. We have introduced in this paper the possibility to model and to use business processes to carry-on with those improvements. They have a great added-value in the model, as they describe new functional behaviors. They also supplement the model with a new level of abstraction, allowing a better model structuring. Methodology presented here was put in practice on a concrete example with industrial partners. However a problem resides in resulting machine depth. Even if tests generation is feasible for each sub-machine composing the final machine, there is no certainty that generation will be possible for the resulting machine.

Expressing a business process in the correct notation was part of our research. Several languages can be found to deal with business processes description : UML, BPMN or DFD. Both notation can be used for this task depending on criteria or measurement grids, and evaluation can be found in (Nysetvold *et al.*, 2005, Wahl *et al.*, 2005, Wohed *et al.*, 2006, Recker *et al.*, 2005). Those languages are related because they were created or improved to deal with the same task : modeling a business process. The main difference resides in the final user qualification. BPMN was created to give business professional a notation to produce models. UML was introduced as an answer to modeling standardization of software development. Activity diagrams were updated in UML 2.0 to deal with business processes modeling specific needs. However UML is still an adequate notation to describe technical domain (White, 2004) and is our final choice for the project.

One key point of this work is to offer a methodology to design business processes from existing model elements. This methodology is a complement to the previous one extending the Unified Process to produce models suitable for tests generation. We now have a complete methodology to achieve functional testing of software applications. A business process is regarded as an integration test, and automatically generated tests augment campaigns already produced by classical model-based testing. Process may be modeled by business expert allowing introduction of information not yet identified. Thus the expert becomes a validation team member and gives a new point of view on the model and on developed functional aspects. A strong point of the methodology is the ability to identify data spreading between activities and to use them for tests generation. Thereby a process without any data spreading would have no added value.

In the same vein, we wish to emphasize the reuse of information from the model. For software product lines it is typically the case. A model will be used for all new versions of application, this may be true for the model to the test. A model previously produced and a specific methodology can lead to a substantial gain in time and money.

6. Bibliographie

Arlow J., Neustadt I., *UML 2 and the Unified Process, Second Edition, Practical Object-Oriented Analysis and Design*, Addison-Wesley, 2003.

Bouquet F., Debricon S., Legeard B., Nicolet J.-D., « Extending the Unified Process with Model-Based Testing », *MoDeVa'06, 3rd Int. Workshop on Model Development, Validation and Verification*, Genova, Italy, p. 2-15, October, 2006.

Bouquet F., Grandpierre C., Legeard B., Peureux F., Utting M., Vacelet N., « A subset of precise UML for Model-based Testing », *A-MOST'07, 3rd Workshop on Advances in Model Based Testing*, London, UK, July, 2007. Proceedings on CD. A-MOST'O7 is colocated with ISSTA 2007, Int. Symposium on Software Testing and Analysis.

Briand L. C., Labiche Y., « A UML-Based Approach to System Testing », *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, Springer-Verlag, London, UK, p. 194-208, 2001.

Nebut C., Fleurey F., Traon Y. L., Jézéquel J.-M., « Automatic Test Generation : A Use Case Driven Approach », *IEEE Trans. Software Eng.*, vol. 32, n° 3, p. 140-155, 2006.

Nysetvold, Krogstie, « Assessing Business Process Modeling Languages Using a Generic Quality Framework », *10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'05)*, Porto, Portugal, 2005.

Recker J. C., Indulska M., Rosemann M., Green P., « Do Process Modelling Techniques Get Better ? A Comparative Ontological Analysis of BPMN », *16th Australasian Conference on Information Systems*, Sydney, Australia, 2005.

Utting M., Legeard B., *Practical Model-Based Testing - A tools approach*, Elsevier Science, 2006a. 550 pages, ISBN 0-12-372501-1. To Appear.

Utting M., Pretschner A., Legeard B., « A Taxonomy of Model-Based Testing », *Technical report 04/2006, Department of Computer Science, The University of Waikato (New Zealand)*, April, 2006b.

Wahl, Sindre, « An Analytical Evaluation of BPMN Using a Semiotic Quality Framework », *10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'05)*, Porto, Portugal, 2005.

White S. A., Process Modeling Notations and Workflow Patterns, Technical report, IBM Corp., 2004. http ://www.bpmn.org/Documents/Notations and Workflow Patterns.pdf.

Wohed P., van der Aalst W. M., Dumas M., ter Hofstede A. H., Russell N., « On the Suitability of BPMN for Business Process Modelling », *Proceedings 4th International Conference on Business Process Management 4102*, Vienna, Austria, p. 161-176, 2006.

Z. J. Li H. F. Tan H. H. L. J. Z., Mitsumori N. M., « Business-process-driven gray-box SOA testing », *IBM Systems Journal*, 2008.

Démarche de développement à base de composants d'applications embarquées tolérantes aux fautes

Mohamed-Lamine Boukhanoufa* — Brahim Hamid — Ansgar Radermacher* — Agnès Lanusse***

*{mohamed-lamine.boukhanoufa, ansgar.radermacher, agnes.lanusse}@cea.fr, ** hamid@irit.fr
*CEA, LIST Boîte courrier 65, GIF SUR YVETTE CEDEX, F-91191 France
**IRIT, 118 Route de Narbonne, 31062 Toulouse cedex 9, France (précédemment CEA, LIST)

RÉSUMÉ. *Dans ce papier, nous proposons une méthodologie de développement d'applications embarquées tolérantes aux fautes combinant l'ingénierie dirigée par les modèles (IDM) et le développement de systèmes à base de composants. Cette approche vise la séparation entre la spécification et l'implémentation des composants pour permettre une certaine flexibilité des implémentations et l'indépendance vis-à-vis des plateformes cibles. La proposition eC3M* [1] *(Modèle Composant Conteneur Connecteur pour l'Embarqué) que nous présentons ici couvre plusieurs facettes : une démarche méthodologique, un environnement de développement et une infrastructure d'exécution (plateforme). Nous décrivons dans une première partie la proposition eC3M au niveau modèle et plateforme d'exécution en précisant les extensions pour la tolérance à travers un exemple d'application robotique sur le robot wifibot 4G.*

ABSTRACT. *In this work, we propose to combine model driven engineering and component based development through a methodology to develop fault tolerant embedded applications. The approach promotes the separation of specification and implementation of components to enable implementation flexibility and efficient use of mechanisms available in target environments. This work contributes in the dissemination of the eC3M proposal (for Embedded Component Container Connector Model). The contribution of this paper is threefold : a methodological approach, a development environment, and a middleware platform. We present the eC3M proposal at two levels : model and execution platform through an example of the realization of a fault tolerant application on a robotic platform wifibot 4G.*

MOTS-CLÉS : *IDM, CCM, connecteur, modèle (UML), système embarqué, tolérance aux fautes.*

KEYWORDS: *MDE, CCM, connector, model (UML), embedded system, fault tolerance.*

1. http ://www.ec3m.net/

1. Introduction

L'ingénierie dirigée par les modèles (IDM ou Model-Driven Engineering) [SCH 06] est une forme d'ingénierie générative. Elle vise à minimiser le coût et le temps de développement des applications en focalisant les efforts sur les phases amont au niveau des modèles et en s'appuyant sur des processus génératifs pour la production de code. L'IDM permet ainsi de répondre rapidement à des évolutions d'exigences aussi bien en termes de fonctionnalités que de contraintes matérielles. Elle se prête donc particulièrement bien au domaine du temps-réel embarqué qui doit faire face simultanément à l'accroissement de la complexité des applications et aux évolutions rapides des cibles matérielles.

De plus, l'exécution des applications déployées sur ces systèmes est souvent contrainte par des exigences de fiabilité et de disponibilité. Le non respect de ces contraintes donne lieu à une dégradation importante de performances, voir à la mise en péril du système et de son environnement. Pour faire face à ces contraintes, une attention spéciale doit être accordée à leur conception et développement.

Dans ce contexte, les approches de développement par composants [LAU 07] permettent de simplifier la conception des systèmes temps-réel embarqués tolérants aux fautes en les construisant par assemblage de composants (préexistants ou nouveaux). Ces approches sont également compatibles avec un objectif de séparation des préoccupations puisque l'implémentation du code métier d'un composant est clairement séparée de sa partie non fonctionnelle.

Nous présentons ici l'environnement de développement eC3M qui facilite la réalisation des applications embarquées à base de composants en combinant ces deux paradigmes. La section 2 introduit eC3M, la section 3 illustre la démarche d'utilisation de cet environnement sur un exemple d'une application robotique. Enfin, nous concluons par une brève discussion et quelques perspectives.

2. L'environnement eC3M

L'environnement eC3M du CEA LIST, rentre dans le cadre des approches IDM (plus particulièrement MDA) ; il est en cours de développement, nous présentons dans cet article une première implémentation mettant en oeuvre des mécanismes de tolérance aux fautes. L'objectif à terme est de fournir une méthodologie et les outils associés, pour le développement d'applications TR/E à base de composants. La proposition eC3M est matérialisée par : 1)la définition d'un processus (approche) de développement , 2) un environnement offrant des moyens de modélisation et de génération de code , 3) une infrastructure d'exécution configurable et des bibliothèques de connecteurs spécialisés notamment pour la tolérance aux fautes (détection de fautes, gestion de redondance).

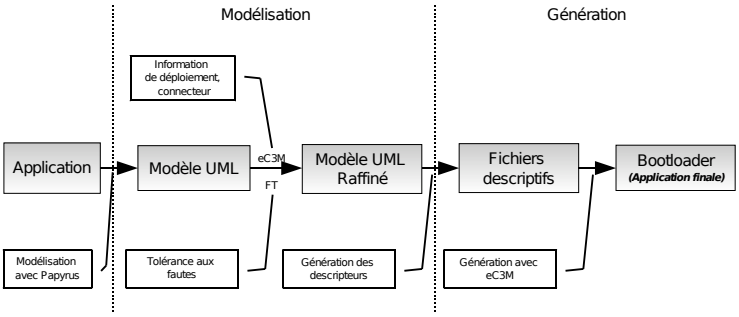


Figure 1. *La chaîne de développement*

L'environnement de développement lui-même comporte : le modeleur Papyrus UML [1] et des extensions UML (eC3M, FT), des outils de génération de code pour le packaging[2] (génération de fichiers de configuration conformes au standard LwCCM) et pour les composants (extension de MicroCCM[3]). Ces éléments sont intégrés dans une chaîne de développement qui supporte le processus proposé (voir figure 1).

2.1. *Niveau modélisation*

Pour les aspects modélisation, notre contribution porte sur la définition du profil eC3M et du profil FT. Le profil eC3M étend LwCCM [OMG 03a],pour la notion de connecteur et pour les aspects déploiement[4]. Ce profil permet d'annoter les modèles dans les étapes de spécification par les informations de déploiement des applications.

Le profil *FT profile* est issu de l'adaptation du profil QoS&FT [OMG 06] pour le contexte de modélisation MARTE [OMG 07]. Le profil QoS&FT a été décomposé en sous-profils exploitant la notion de *Non Functionnal Property* de MARTE en lieu et place de la notion de *QoS Characteristics* et des bibliothèques de types remplacent le catalogue défini initialement dans la norme. Concrètement, le profil FT est composé de stéréotypes permettant de spécifier la politique de détection de faute, le style de réplication, le nombre de répliques d'un composant.

2.2. *Niveau plate-forme d'exécution*

Nous sommes en présence de systèmes répartis constitués d'un ensemble de processus (ou noeuds) et de liens de communication. Les processus communiquent et se

1. Modeleur UML : http ://www.papyrusuml.org/
2. Plugin PackagingTools pour Papyrus
3. Framework implémentant LwCCM développé par Thales en collaboration avec le CEA LIST.
4. compatible avec le profil Deployment and Configuration de l'OMG

synchronisent via l'envoi et la réception de messages par le biais des liens. Un processus peut tomber en panne par l'arrêt permanent de son fonctionnement. Un processus peut également produire des faux résultats de calcul. Les liens de communication sont supposés être fiables. Le réseau est asynchrone : le délai de transfert de messages est non borné mais fini, l'ordre des messages est préservé. Pour la mise en oeuvre de détection de pannes, on utilise une faible forme de synchronie comme celle étudiée dans [CHA 96, HAM 07]. Pour assurer la tolérance aux fautes nous utilisons la technique de réplication active avec vote où toutes les répliques exécutent la même demande et le résultat est voté.

L'infrastructure d'exécution est constituée d'une plateforme LwCCM dont les mécanismes de communication ont été étendus pour supporter le mécanisme de connecteur. La principale motivation pour l'introduction des connecteurs [ROB 05] était de rendre plus flexibles les choix d'implémentation des interactions entre composants de manière à pouvoir utiliser efficacement les mécanismes disponibles sur les systèmes cibles (sockets par exemple). Cela est particulièrement important pour les systèmes embarqués. Dans notre cas, le connecteur offre un intérêt supplémentaire, il permet d'implanter simplement et de manière transparente pour l'utilisateur des mécanismes spécifiques au traitement de la tolérance aux fautes.

Le mécanisme de tolérance aux fautes implémenté se présente comme une infrastructure simple basée sur un ensemble d'éléments non fonctionnels similaires à ceux de FT-CORBA [OMG 04], en revanche, ils sont indépendants d'un ORB particulier. Seul le conteneur et le fragment de connecteur sont concernés par la gestion des aspects de tolérance aux fautes pour ce qui est des communications. En complément de la définition de connecteurs spécialisés nous proposons une infrastructure à trois composants [HAM 08] : *un détecteur de faute (FD)*, *un gestionnaire de réplique (RM)* et *un gestionnaire de tolérance aux fautes (FTM)*.

3. Démarche d'utilisation de eC3M sur un exemple réel (wifibot 4G)

Afin d'illustrer la démarche de développement proposée, nous présentons ici sa mise en oeuvre sur un exemple réel : un dessinateur robotisé de cartographie. Cette application permet de tracer une carte géographique qui donne une description de la surface d'une zone cible. La zone est parcourue par un robot qui indique périodiquement l'état d'occupation (Occupé/Inoccupé/Inconnu) du point où il se trouve. Le système est constitué d'un robot(wifibot 4G) et d'une base(machine). A travers le composant *Robot*, le robot peut indiquer sa position et orientation (x, y, theta) et la distance télémétrique calculée. La base de contrôle traite les informations collectées par le robot pour construire la cartographie finale en passant par plusieurs cartographies intermédiaires à l'aide du composant *Designer*. Ce dernier calcule à chaque itération les coordonnées des positions à explorer sur la zone. Les composants *Pilot, Screen* sont respectivement utilisés pour contrôler le robot et afficher les informations sur l'écran de la base.

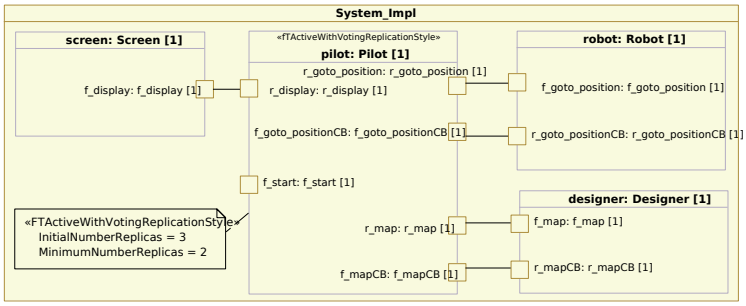


Figure 2. *Application du profil FT pour la spécification de la tolérance aux fautes*

Nous avons simplifié les exigences de notre application, en prenant en compte les fautes du composant *Pilot* (erreurs de calcul et pannes du matériel). Pour ce faire, trois répliques sont déployées sur trois machines différentes. Comme le montre la figure 1 le développement avec eC3M passe par deux phases :

1) Modélisation : la modélisation d'une application se base principalement sur la description :

a) *des interfaces :* la spécification des interfaces permet de préciser quels sont les services requis et fournis par les différents composants. Pour se faire nous utilisons directement les éléments de modélisation UML.

b) *des composants (types) :* Chaque composant identifié pendant l'analyse de cahier des charges est modélisé sur un diagramme de composants UML. Sur l'élément UML "composant" sont placés les ports typés par les interfaces.

c) *des composants (implémentations) :* Chaque composant peut avoir une ou plusieurs implémentations (dans notre exemple une seule). Les implémentations peuvent être directes, écrites dans un langage de programmation (monolithique en terminologie CCM) ou construites par composition d'autres implémentations (assemblage en terminologie CCM).

d) *du système :* Cette étape permet de modéliser l'application par la description de l'assemblage des composants. On utilise le diagramme UML "composite". La figure 2 montre le système de dessinateur de cartographie. C'est dans cette phase que l'on spécifie également les contraintes de tolérance aux fautes par l'application du profil FT via le stéréotype *FTActivewithVotingReplicationStyle* et le profil eC3M pour spécifier les types de connecteurs et leurs implémentations.

e) *de la plate-forme et du diagramme du déploiement :* le modèle de la plateforme est composé d'un ensemble de noeuds et d'interconnexions. Les noeuds (en général une machine) représentent l'environnement d'exécution des instances de composants. Les interconnexions fournissent les connexions directes entre les noeuds. La plate-forme matérielle de notre application est composée de cinq noeuds. Un noeud correspondant au calculateur du Robot, un noeud pour une partie de la base où les ins-

tances des composants *Designer* et *Screen* sont déployées, et trois noeuds contenant les instances du composant *Pilot*. Le diagramme de déploiement d'un noeud indique les instances de composants qui seront déployées sur ce noeud.

2) Génération du code : la génération de code à partir du modèle passe par deux phases. Premièrement, la génération des descripteurs des composants et du plan de déploiement à partir du modèle UML raffiné par les deux profils eC3M, FT. Deuxièmement, la génération du code source C++ de l'application à partir des fichiers générés précédemment, des fichiers IDL (composants et interfaces) et du code métier des composants. Un fichier "Bootloader" est produit pour permettre d'instancier les composants et les fragments des connecteurs sur chacun des noeuds du système.

4. Discussion et positionnement

Notre démarche est une spécialisation de l'approche Accord|UML [GER 00] proposée initialement pour le développement d'applications temps réel orientées objet. Nous l'avons adaptée au cas du développement à base de composants, dans ce sens l'objectif de nos travaux se rapproche de celui des travaux menés autour de l'environnement COSMIC [SCH 02]. Comme celui-ci, elle s'inscrit dans le courant de développement dirigé par les modèles et le principe de séparation des aspects application et plateforme promu par le MDA [OMG 03b]. Toutefois, nous nous intéressons plus précisément au contexte du temps réel embarqué, aussi les solutions basées sur des environnements CORBA sont souvent jugées trop lourdes (encombrement mémoire) et peu efficaces (performance). En préconisant une approche plus flexible de l'implémentation des interactions à l'aide de connecteurs, nous proposons une alternative intéressante pour l'embarqué à plusieurs titres : une implémentation légère et ajustée à la cible des mécanismes de communication, et une possibilité de spécialisation des connecteurs pour la prise en compte de contraintes non fonctionnelles. Notre proposition s'articule autour d'un ensemble d'outils de modélisation (PapyrusUML), similaire à ce que l'on peut trouver dans Fujaba.

5. Conclusion

Dans l'étude présentée ici, nous avons proposé une démarche de développement de systèmes à base de composants pour les systèmes TR/E tolérants aux fautes. Notre contribution porte sur les aspects modélisation (profils FT et eC3M), plateforme (composants dédiés et bibliothèque de connecteurs) et génération automatique de code. Elle est illustrée sur une application robotique réelle. Les mécanismes proposés implémentent des stratégies connues mais leur utilisation est grandement simplifiée (application de stéréotypes et utilisation de bibliothèques de connecteurs). Dans le futur nous envisageons d'enrichir cette première plateforme, une proposition a été faite pour ajouter le traitement de contraintes temps-réel au niveau des connecteurs. La prise en compte d'autres aspects non-fonctionnels (ressources) et le support à l'analyse (optimisation de l'allocation) en s'appuyant sur le profil MARTE sont également envisagés.

Remerciements : *Nous tenons à remercier vivement Arnaud Cuccuru, Sébastien Gérard et François Terrier pour leur aide dans la réalisation de ce travail.*

6. Bibliographie

[CHA 96] CHANDRA T., TOUEG S., « Unreliable failure detectors for reliable distributed system », *Journal of the ACM*, vol. 43, n° 2, 1996, p. 225–267.

[GER 00] GERARD S., « Modélisation UML exécutable pour les systèmes embarqués de l'automobile », PhD thesis, University of Evry, 2000.

[HAM 07] HAMID B., « Distributed fault-tolerance techniques for local computations », PhD thesis, University of Bordeaux 1, 2007.

[HAM 08] HAMID B., RADERMACHER A., VANUXEEM P., LANUSSE A., GERARD S., « A fault-tolerance framework for distributed component systems », *Proceedings of the 34th Euromicro SEAA conference*, IEEE CS, 2008, p. 84-91.

[LAU 07] LAU K., WANG Z., « Software Component Models », *IEEE Transactions on Software Engineering*, vol. 33, n° 2, 2007, p. 709-724, IEEE.

[OMG 03a] OMG, « Lightweight Corba Component Model (CCM) Specification », 11 2003, OMG Document ptc/03-11-03.

[OMG 03b] OMG, « MDA Guide Version 1.0.1 », 2003, omg/2003-06-01.

[OMG 04] OMG, « CORBA Core specification, Version 3.0.3 », 2004, OMG Document formal/2004-03-12.

[OMG 06] OMG, « UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms », 5 2006, OMG Document formal/2006-05-02.

[OMG 07] OMG, « UML Profile for MARTE », 2007, OMG document ptc/07-08-04.

[ROB 05] ROBERT S., RADERMACHER A., SEIGNOLE V., GERARD S., WATINE V., TERRIER F., « Enhancing Interaction Support in the CORBA Component Model », *From Specification to Embedded Systems Application*, Springer, 2005.

[SCH 02] SCHMIDT D., GOKHALE A., NATARAJAN R., NEEMA E., BAPTY T., PARSONS J., GRAY J., NECHYPURENKO A., WANG N., « CoSMIC : An MDA Generative Tool for Distributed Real-time and Embdedded Component Middleware and Applications », *In Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*, ACM, 2002, p. 300–306.

[SCH 06] SCHMIDT D., « Model-Driven Engineering », *in IEEE computer*, vol. 39, n° 2, 2006, p. 41-47.

Etude de la Sûreté de Fonctionnement dans un processus basé sur l'Ingénierie Dirigée par les Modèles

Daniela Cancila* — Hubert Dubois* — Morayo Adedjouma**

** CEA LIST*
Commissariat à l'Energie Atomique - Saclay
Laboratoire d'Ingénierie dirigée par les modèles pour les Systèmes Embarqués
91191Gif- sur-Yvette Cedex FRANCE
{Daniela.Cancila, Hubert.Dubois}@cea.fr
*** ADNEOM Technologies*
126, Rue Réaumur, 75002 Paris France
morayo.adedjouma@tremplin-utc.net

RÉSUMÉ. *Aujourd'hui le marché économique a adopté le bien-connu paradigme "cheaper, faster, better" pour le développement logiciel où les systèmes critiques embarqués temps réel ne font pas exception. Une solution possible est alors offerte par l'adoption d'une approche orientée modèles (IDM). Dans cet article, nous suivons une telle approche et nous étudions comment intégrer les exigences de sûreté de fonctionnement (et plus particulièrement, leur fiabilité) dès les étapes initiales de modélisation. Nous basons notre approche sur trois points : la définition d'un méta-modèle dédié à la sûreté de fonctionnement compatible avec les autres profils standards UML ; la définition d'une méthodologie adaptée et l'adoption de preuves formelles pour prouver ces propriétés. Dans cet article, nous présentons notre stratégie globale et les challenges que nous traitons.*

ABSTRACT. *From the nineties, paradigm "cheaper, faster, better" is becoming a successfully paradigm to develop systems such that real-time embedded systems where safety guarantees are mandatory. Model-driven engineering (MDE) approaches provide a solution to manage the increasing complexity of such systems. In this paper, we adopt a MDE approach and we study how to integrate safety requirements (and, more particularly, the dependability) from the early phases of software development. We base our investigation on three pilasters: definition of a safety metamodel compatible with UML standard profiles; definition of a software development methodology and adoption of formal methods to prove properties. In this paper we discuss our strategy and some challenges that we are faced to.*

MOTS-CLÉS : *ingénierie dirigée par les modèles (IDM), sûreté de fonctionnement, fiabilité, métamodèles, profil UML.*

KEYWORDS: *model driven engineering (MDE), safety, dependability, metamodel, UML profiles.*

1. Introduction

Les systèmes embarqués temps réel jouent un rôle majeur dans la quasi-intégralité des secteurs industriels (domaine de l'automobile, du ferroviaire, des télécommunications, de l'aéronautique, du spatial, etc.). La conception et la réalisation de ces systèmes doit tenir compte non seulement des aspects fonctionnels mais aussi des différents aspects extra-fonctionnels, comme les contraintes temporelles ou la sûreté de fonctionnement. A ces contraintes de criticité est venue s'ajouter une complexité croissante de ces systèmes. Dès lors, la communauté scientifique a cherché à définir des méthodologies orientées modèles à base de composants pour développer ces systèmes.

Ainsi, nous assistons depuis quelques années au passage des approches de développement basées « code » (c.à.d. basées sur l'écriture manuelle du code) vers des approches basées « modèles » ; ces approches, dites « génératives », introduisent un niveau d'abstraction supplémentaire (le modèle), à partir duquel le programme traditionnellement écrit à la main est automatiquement généré. Nous adoptons l'approche d'Ingénierie Dirigée par les Modèles (ou « IDM ») comme celle-ci forme un cadre unifié pour aborder la conception, le développement et la validation des propriétés fonctionnelles et non-fonctionnelles des systèmes embarqués temps réel. Dans les différentes approches de l'IDM pour l'ingénierie du logiciel, le consortium international OMG (*Object Management Group*)[1] soutient les développements autour de UML (*Unified Modeling Language*) et c'est dans ce contexte que nous travaillons. UML est une famille de langages pour la conception et la modélisation de systèmes logiciels (Selic, 2004).

En parallèle de ces travaux sur la modélisation et la représentation des systèmes, de nombreuses techniques ont été définies depuis des années pour permettre d'aider à la prise en compte de la sûreté de fonctionnement dans la définition des systèmes ; ne pouvant ici tout citer, nous invitons le lecteur à consulter l'ouvrage de (Laprie, 1996) comme référence. Ces techniques sont indépendantes du formalisme de représentation des systèmes mais elles s'attachent à un but bien précis ; à savoir fournir des moyens pour détecter les pannes et défaillances des systèmes afin de les corriger dans leur conception des systèmes et de les fiabiliser. Ces techniques ont leurs propres modes de représentations et les modèles utilisés dans ces techniques ne sont pas ceux utilisés pour la conception des systèmes étudiés. Une première approche a été faite dans le profil QoS&FT (Qos&FT, 2004) pour une meilleure intégration en UML.

L'objectif de cet article est de faciliter l'intégration des deux approches dans un processus de modélisation des systèmes basé sur l'IDM et ainsi, permettre plus facilement, à partir des formalismes liés à UML, d'intégrer les concepts liés à la gestion de la sûreté de fonctionnement (noté SdF dans la suite) avec un accent plus particulier sur la fiabilité des systèmes pour leur définition.

[1] Voir site de l'OMG : http://www.omg.org

2. Objectifs et Contributions

Pour pouvoir proposer une modélisation des exigences de sûreté de fonctionnement qui conviennent notamment aux domaines ferroviaires, automobiles ou encore de l'avionique, nous devons connaître ce qu'est une exigence de sûreté et quelles sont celles qui sont utilisées dans les domaines susmentionnés à travers leurs différents référentiels normatifs. La définition d'un profil SdF doit donc offrir la possibilité : (1) de spécifier les exigences ; (2) de définir l'architecture du système ; (3) de déployer les exigences sur l'architecture ; (4) d'avoir la traçabilité des exigences rendant possibles la modification du système ainsi que les transformations de modèles ; (5) de garantir la couverture d'une exigence à partir des composants de l'architecture ; (6) et enfin de démontrer leurs cohérences (notamment pendant la composition de différents composants architecturaux qui satisfont chacun leurs propres exigences).

Pour pouvoir considérer tous les points cités précédemment, il s'agit de formaliser en complément du profil SdF un processus de développement logiciel comme par exemple celui défini dans le cadre du projet MeMVaTEx (Albinet *et al.*, 2008) ou par un profil de *safety* (de Miguel *et al.*, 2008). Le processus et le profil définis dans le cadre de MeMVaTEx offrent une conception et un développement du logiciel basés sur l'intégration d'un profil d'exigences pour le domaine de l'automobile avec des profils déjà existants, notamment SysML et MARTE. De plus, une formalisation mathématique basée sur la théorie des graphes, comme des approches récentes cherchent à le faire (Mens, 2005), devrait contribuer à faciliter la démonstration des propriétés formelles sur le système ainsi modélisé. Notre approche est basée sur trois axes : la définition d'un processus de développement pour faciliter la gestion de la fiabilité ; la définition d'un profil SdF à partir de la composition de profils existants ; un objectif de formalisation à l'aide de la théorie de graphes de propriétés de bonne formation des profils qui ne sera qu'introduit ici car il fait l'objet de développements en cours. Dans la suite, on présente d'abord la méthodologie et ensuite, la problématique liée à la composition de profils existants.

Le premier axe est donc la définition d'une méthodologie permettant d'aider à assurer une bonne Sûreté de Fonctionnement des systèmes développés. Cette méthodologie doit être adaptée à un processus dirigé par les modèles. Une fois les exigences liées à la SdF modélisées à l'aide d'un profil facilitant la gestion des exigences (Albinet *et al.*, 2008), et comme les profils UML existants n'étant pas suffisants pour prendre en compte les propriétés de SdF dans la définition des systèmes, nous devons tout d'abord offrir les mécanismes suffisants afin de modéliser la SdF de ces systèmes. La Figure 1 montre le paquetage `Threads` (extrait du profil SdF (Adedjouma, 2008)) où il est spécifié qu'une faute est la cause à l'origine d'une erreur. Une erreur est vue comme une cause *potentielle* d'un incident non désiré, c'est-à-dire qu'étant donnée une faute, on n'a pas forcément une erreur dans le système (Laprie, 1996). Un sous-système peut aller à l'encontre d'erreurs ou de défaillances.

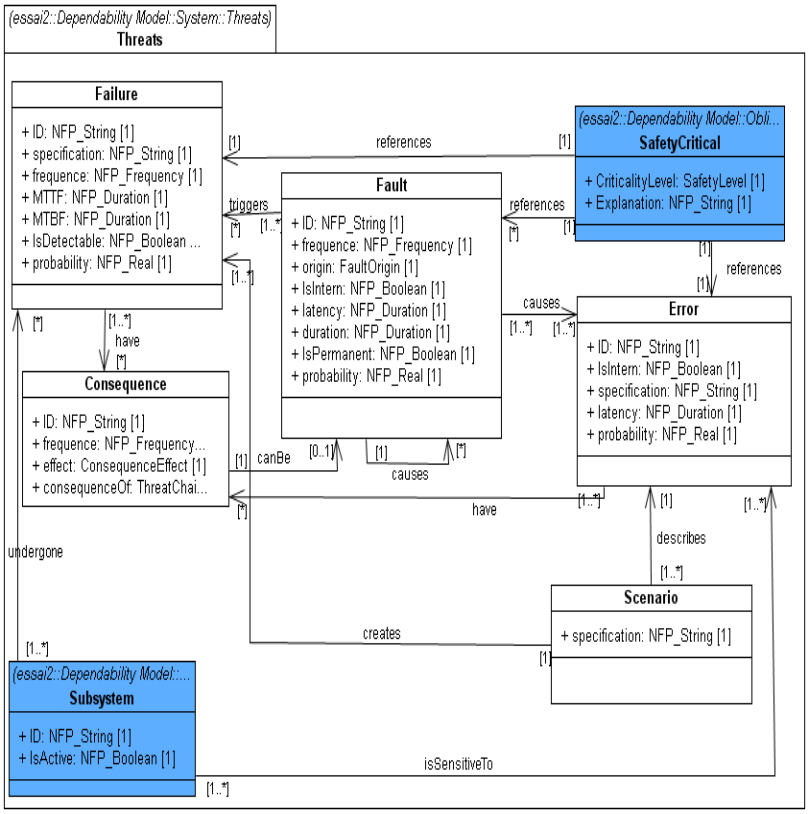


Figure 1. *Le paquetage* `Threat` *du profil SdF*

Le profil SdF nous permet de préciser des modèles UML dans lesquels nous pouvons introduire les notions d'erreur, de défaillances, etc… Les systèmes ainsi modélisés peuvent donc être connectés aux différentes exigences de sûreté de fonctionnement que nous avons définies par ailleurs. Ces profils, définis dans l'outil Papyrus|$_{UML}$[2], permettent d'ores et déjà la modélisation de ces systèmes. Nous avons étudié le profil SdF sur un cas d'étude dans le domaine de l'avionique (Adedjouma, 2008). L'usage du paquetage `Threat` du profil SdF permet ainsi de définir les fautes, les erreurs et les défaillances d'un système, mais aussi les conséquences ou les scénarii qui pourraient les entrainer ou les déclencher.

Le deuxième axe se base sur la définition du profil SdF à partir d'une composition de profils existants. On part ainsi des travaux de B. Selic et C. André

[2] Voir site internet de Papyrus|$_{UML}$: http://www.papyrusuml.org

qui, récemment, ont défini UML comme un ensemble de langages (Selic, 2004) (André, 2007). Intentionnellement, UML introduit des points de variations sémantiques afin de pouvoir représenter un vaste spectre de systèmes et de propriétés. Un profil UML est alors vu comme une famille de langages qui spécialise la sémantique de UML et élimine certaines de ses ambigüités sémantiques (André, 2007). Ces travaux nous suggère qu'on peut définir un profil SdF comme une composition de paquetages appartenant à des profils hétérogènes : des paquetages seront importés de SysML, pour traiter des exigences, des paquetages seront importés de MARTE, pour la prise en compte des contraintes temporelles, et d'autres paquetages seront introduits *ex novo*. Il s'agit donc d'affronter la composition de profils hétérogènes (ici, MARTE, SysML, profil d'exigences et le profil SdF) et de démontrer la cohérence des informations spécifiées sur les différentes vues d'un modèle. La solution n'est pas du tout évidente comme ceci est démontré dans (Espinoza *et al*., 2008).

3. Conclusions et Perspectives

Dans cet article, nous avons discuté des problématiques liées à la prise en compte de la sûreté de fonctionnement et de la fiabilité des systèmes dans une modélisation basée sur UML. Des résultats ont ainsi donné lieu à la définition d'un profil UML permettant d'introduire dans les modèles les notions importantes liées à la sûreté de fonctionnement. Un défi que l'on doit affronter est la réalisation des vues liées à la sûreté de fonctionnement dans le cadre d'UML et comment celles-ci interagissent avec le modèle architectural pendant ses modifications ou lors de transformations de modèles. Nous sommes en train d'affronter ce défi dans le cadre du projet IMOFIS, avec la participation de Renault et Alstom. Ce projet traite de la prise en compte de la sûreté de fonctionnement dans un environnement basé sur les modèles pour les domaines des transports automobiles et ferroviaires. Des études sont en cours afin de confronter ces résultats à des cas applicatifs sur lesquels les normes de certification ou en construction (pour l'automobile notamment avec la norme ISO26262) sont en vigueur. De plus, le processus de modélisation doit également être confronté aux différents processus méthodologiques liés à la prise en compte de la SdF comme l'AMDEC ou la FTA. Dans le cadre de notre travail de recherche, nous souhaitons confronter notre approche avec la méthodologie introduite par la NASA et le MIT dans (Stamatelatos *et al.,* 2002) et nous confronter avec le travail de l'ESA pour les solutions proposées dans la mission lunaire SMART-1 coordonnée par l'ESA (Camino *et al*., 2005). L'objectif est ainsi d'avoir un profil UML SdF tel qu'il soit formé à partir de la composition de certains paquetages précis de MARTE, SysML et d'autres profils métier dédiés par le modèle architecturel.

Remerciements : Nous tenons à remercier les différents relecteurs de cet article pour leurs remarques.

4. Bibliographie

(Adedjouma, 2008) Adedjouma M., Définition d'un méta modèle et des mécanismes de vérification pour représenter les aspects sécurité des systèmes dans le cadre d'une approche de développement centrée sur les modèles, dans le but de faciliter les analyses de sécurité dès les phases amonts de développement, Rapport de stage interne, CEA LIST, 2008

(Albinet *et al.,* 2008) Albinet A., Begoc S., Boulanger J.-L., Casse O., Dal I., Dubois H., Lakhal F., Louar D., Peraldi-Frati M.-A., Sorel Y. et Van Q.-D., « The MeMVaTEx methodology: from requirements to models in automotive application design » *Actes de ERTS*, France, 2008.

(André, 2007) André C., « Time Modeling in MARTE » *Actes de FDL'07*, Spain, 2007.

(Bézivin, 2002) Bézivin J., « Les nouveaux défis des systèmes complexes et la réponse MDA de l'OMG. Une approche de génie logiciel » *Actes de JFIADSMA'02*, France, 2002.

(Caminol *et al.*, 2005) Camino1 O., Alonso M., Blake R., Milligan D., de Bruin J., Gestal D., Ricken S., « SMART-1: Europe's Lunar Mission Paving the way for New Cost Effective Ground Operations » *Actes de RCSGSO*, ESA SP-601. European Space Agency, 2005.

(de Miguel *et al.*, 2008) de Miguel M.A., Briones J.F., Silva J.P. et Alonso A. « Integration of satety analysis in model-driven software development » *Journal of The Institution of Engineering and Technology, Special Issue on Lang. Engeenering.*, vol 2, pp 260-280, 2008.

(Espinoza *et al.*, 2008) Espinoza H., Selic B., Cancila D. et Gérard S., Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. Technical Report CEA, 2008.

(Laprie, 1996) Laprie J.-C., *Guide de la sûreté de fonctionnement*, 1996.

(Mens, 2005) Mens T., « On the Use of Graph Transformations for Model Refactoring » *Inter. Summer School GTTSE*, Portugal, 2005

(Stamatelos *et al.*, 2002) Stamatelatos M., Vesely W. and al., Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance, 2002.

(QoS&FT, 2004) OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms. ptc/04-09-01, 2004.

(Selic, 2007) Selic B., From Model-Driven Development to Model-Driven Engineering, Keynote talk at ECRTS'07, Italy, 2007

(Selic, 2004) Selic B., « On the semantic foundations of standard UML 2.0 », *Actes de SFM-RT*, LNCS, pp. 181-199, 2004

(Smith, 2004) Smith D., Simpson KGL. *Functional Safety*. Elsevier, 2004

(Stringfellow *et al.*, 2007) Stringfellow Herring M., Owens B. D., Leveson N., Ingham M.,Weiss K.A., Safety-Driven Model-Based System Engineering Methodology Part I: Methodology Description, MIT JPL, 2007

Ingénierie des exigences par l'IDM

Assistance à l'ingénierie des systèmes complexes

Eric Le Pors[*,,***] - Olivier Grisvard[*,**,***] - Yvon Kermarrec[*,***]**

[*] *Institut Télécom - Télécom Bretagne, Département LUSSI*
Technopôle Brest-Iroise - CS 8381 - 29238 Brest Cedex 3
<eric.lepors, olivier.grisvard, yvon.kermarrec>@telecom-bretagne.eu
[**] *THALES Systèmes Aéroportés*
Direction Technique AMS - 10 av 1ère DFL 29200 BREST
[***] *Université Européenne de Bretagne*

RÉSUMÉ. *Le respect de contraintes provenant des exigences de niveau système reste un problème ouvert pour l'ingénierie des systèmes complexes. Une incompréhension ou une ambiguïté à ce niveau peut entraîner de graves conséquences pour la réalisation de ces systèmes. Nous décrivons dans cet article une approche pragmatique, s'intégrant aux usages de l'entreprise et à une méthodologie d'architecture des systèmes, qui s'appuie sur l'IDM et un formalisme composant afin de mieux maîtriser la sémantique et le contenu des exigences. Cette approche nous permet de limiter les incohérences et les incompréhensions dans les exigences système. Nous proposons au travers d'un exemple, une mise en application de cette approche.*

ABSTRACT. *Respecting requirement constraints at system level is still an issue for complex systems engineering. Misunderstanding and ambiguity can lead to severe consequences for the implementation of these kind of systems. We describe in this article a pragmatic approach integrated with company practices and with a system architecture methodology, based on MDE and on a component formalism which enables to manage the contents and semantics of requirements. Our approach permits to reduce incoherence and misunderstanding in system requirements. We propose through an example, an application of this approach*

MOTS-CLÉS : *Ingénierie des exigences, IDM, Modèles Conceptuels, Composants.*

KEYWORDS: *Requirement engineering, MDE, Conceptual Models, Components.*

1. Introduction

L'ingénierie des systèmes rencontre de plus en plus de problèmes dans la réalisation et la maîtrise des systèmes complexes. Ces systèmes sont souvent caractérisés par un certain nombre de sous-systèmes hétérogènes et interconnectés par différents types de bus ou de réseaux. Dans le cadre de nos recherches, nous étudions certains de ces systèmes et sous-systèmes développés par THALES Systèmes Aéroportés, notamment les systèmes de Patmar (patrouille maritime). Ces systèmes sont constitués de coffrets de traitements interconnectés, de senseurs comme le RADAR, le FLIR (Forward Looking InfaRed), la centrale de navigation, ... La fonction de ces systèmes est d'effectuer la synthèse d'une situation tactique pour assister les opérateurs de patrouille maritime lors de leurs différentes actions et tâches. Ces systèmes sont utilisés par plusieurs opérateurs humains qui doivent interagir et exécuter une mission commune : La surveillance du trafic maritime, le contrôle de la pollution... Le fait que ces systèmes soient multi-opérateurs, l'augmentation du nombre de fonctionnalités des différents senseurs et par la même, l'augmentation de la complexité de ces sous-systèmes (volumes de données produites, nombre de fonctions à gérer, à réaliser, les paramétrages...) rendent plus difficile la tache des architectes du système. Dans ce contexte de modélisation des systèmes complexes, l'Ingénierie Dirigée par les Modèles (IDM) se propose d'aider à maîtriser cette complexité grandissante (Bézivin *et al.,* 2004). De nombreuses solutions existent pour mieux maîtriser les architectures logicielles, de l'application des patrons de conception (Gamma *et al.,* 1995) à la description des architectures sous différents points de vue (utilisant par exemple, un ensemble de langages spécifiques de domaine). Le Développement Dirigé par les Modèles trouve désormais des applications concrètes dans l'industrie. L'ingénierie du logiciel dispose de nombreux outils ayant atteint une certaine maturité pour l'aider dans le développement, le partage de la solution de conception ou encore pour générer automatiquement une partie du code et de la documentation (Jézéquel *et al.,* 2006).

Nous proposons d'appliquer certaines de ces techniques pour assister l'ingénierie des Systèmes, un autre métier intervenant plus en amont lors de la réalisation de projets. Ce métier est notamment au contact du client, doit formaliser son besoin et construire une solution architecturale cohérente de ce dernier. Dans ce cas, nous ne nous adressons plus à des experts logiciels mais plus à des architectes système, des ingénieurs spécialistes des produits et de leurs fonctions. Ces ingénieurs souhaitent le plus souvent interagir avec le client, lui décrire des éléments du système. Il est évidemment plus difficile de leur demander d'interagir avec les outils existants de la même façon. C'est en partie ce qui a motivé notre démarche d'assistance au métier d'ingénierie des systèmes visant à améliorer le processus d'écriture des exigences.

Nous proposons en effet une assistance à l'ingénierie des systèmes (respectant la norme ISO 15288) dans les étapes de spécification précédant la conception des architectures logique et physique répondant aux besoins exprimés dans les exigences de niveau système. Nous proposons également de poursuivre cette démarche en parcourant le "cycle en V" jusqu'a l'ultime phase d'ingénierie système précédant la phase d'ingénierie logicielle (ou matérielle).

Nous abordons dans la partie suivante la problématique associée à l'ingénierie des exigences de niveau système. Nous décrivons différentes techniques ou approches associées à ce domaine. Nous présentons ensuite notre approche de métamodélisation et de modélisation, ainsi que les outils servant à manipuler ces modèles. Nous terminons enfin par un exemple d'application sur des exigences que nous avons étudié.

2. Problématique

Le contrat que construit l'ingénieur système avec le client est traduit sous forme d'exigences (dites de niveau système) qui deviennent la base de travail pour les ingénieries logicielles et matérielles notamment. Les méthodologies classiquement utilisées dans l'industrie (par exemple à THALES) consistent à décliner ces exigences en éléments architecturaux en s'inspirant ou en utilisant diverses techniques proposées par la communauté de l'Ingénierie Dirigée par les Modèles. Les exigences système sont souvent exprimées en Anglais, en utilisant un langage naturel ou légèrement structuré. Le texte de ces exigences est considéré comme une référence pour construire la solution , il devient donc important de vérifier la cohérence de cette solution avec les besoins exprimés. Le risque étant de voir un client ou un utilisateur refuser le système en fin de cycle de livraison pour incompatibilité avec ses besoins exprimés dans les exigences. Sommerville (Sommerville, 2004) identifie le fait que ces exigences font partie du contrat entre l'acheteur et le fournisseur du système, mais il ajoute surtout que ce document doit décrire exactement ce que devra contenir le système : il doit être précis.

L'ingénierie des exigences regroupe un vaste domaine de recherche sur les exigences (Nuseibeh *et al.,* 2000). De nombreuses études ont conforté notre point de vue sur le fait que la maîtrise de l'expression de ces éléments (constitutifs de la chaîne de production des systèmes) est une clé dans la maîtrise de la complexité. La maîtrise des exigences et de leur contenu évite d'introduire une complexité non désirée, provenant généralement de manques ou d'incompréhensions dans les exigences. Dans la pratique, ces problèmes entraînent un surcoût important en fin de projet (Sommerville, 2007)

Une première approche plutôt bien intégrée au monde industriel consiste à utiliser un ou plusieurs outils de gestion d'exigences. L'intéret de ces outils (comme Telelogic DOORS (Azelborn, 2000) par exemple) est de permettre une gestion des exigences, de leur traçabilité et de leur versionnage. Ce genre d'apporche permet d'effectuer un premier niveau de suivi de la conception de systèmes par les exigences. Ce genre d'approche n'a cependant pas pour but d'améliorer la qualité des exigences. Ceci peut notamment impacter la qualité de la traçabilité (étant faite manuellement entre les différents documents) et donc le niveau du suivi réel de l'évolution du produit depuis les contraintes de niveau système.

De nombreuses études pointent les risques liés à une mauvaise expression ou à un manque dans les exigences : Sawyer (Sawyer *et al.,* 1997) exprime le fait que le

succès d'un projet de développement est intimement lié à la qualité des exigences. Sommerville (Sommerville, 2004) soutient l'idée que le langage naturel laisse trop de place aux incohérences ou à l'imprécision. Scott (Scott *et al.*, 2004) va dans ce sens en expliquant que des exigences "pauvres" sont souvent données pour cause principale dans l'échec de projets, tout en proposant une grammaire d'exigences pour favoriser un certain nombre de vérifications automatiques sur ces dernières. D'autres études ont proposé de construire un langage de description d'exigences (LDE) permettant de simplifier le traitement des exigences par la structuration du langage (Nebut *et al.*, 2004) (Fleurey *et al.*, 2006).

Plusieurs auteurs proposent une liste de "qualités désirables" que doivent respecter les exigences : Wilson (Wilson *et al.*, 1997) propose la suivante : Classées, Complètes, Consistantes, Correctes, Modifiables, Non-ambiguës, Traçables et Vérifiables. Wilson (Wilson *et al.*, 1996) présente également dans le cadre du développement de ARM (pour "Automated Requirement Measurement", un outil de gestion et de mesures sur les exigences) une technique d'identification d'exigences pouvant présenter un risque. Pour ce faire il utilise une technique basée sur la détection de mots ou propriétés non désirés dans les exigences (car trop pauvres ou pouvant impliquer une exigence non claire par exemple). Mais cette technique ne peut évidemment pas éviter les expressions ambiguës, incorrectes, invérifiables, malgré l'utilisation de mots n'appartenant pas à la liste des mots à bannir.

Pour remédier aux défauts des spécifications en langage naturel, et ainsi permettre une meilleure qualité des exigences, certaines études ont proposé l'utilisation de méthodes formelles de spécifications : la notation Z (Spivey, 1988) puis la méthode B (Wordsworth, 1996) par exemple, qui ont été proposées afin de résoudre ces problèmes. Wilson (Wilson *et al.*, 1996) a notamment reconnu l'indéniable avantage de ces méthodes, mais fait le constat, toujours vrai à l'heure actuelle, que leur utilisation n'est pas devenue chose courante. Il donne également un début de réponse au pourquoi de cet état de fait : La nécessité pour les exigences d'être comprises par les deux parties (le développeur mais également le client).

La nécessité de se comprendre apparaît en effet essentielle, aussi bien entre client et fournisseurs, qu'entre les différent niveaux d'ingénierie, voire même les membres d'une même équipe de design. Hammond (Hammond *et al.*, 2001) exprime cette nécessité de compréhension du domaine de l'application, qu'il s'agit d'un point crucial pour savoir si l'on satisfait les exigences. Certains ont choisi pour ce faire, l'utilisation d'ontologies : "Une ontologie est une spécification formelle explicite d'une conceptualisation partagés" (Gruber, 1993). Certaines études sur l'applications d'ontologies à l'analyse sémantique du contenu des exigences ont été proposées dont celle de Kaiya (Kaiya *et al.*, 2005) qui propose par son approche une détection de spécifications non complètes ou incohérentes. L'analyse sémantique, rendue possible par la définition d'ontologies, a un intéret certain.

La construction d'ontologies trouve une application dans de nombreux domaines, Brisson (Brisson, 2006) et Dosbon (Dobson *et al.*, 2006) décrivent OWL (Web Ontology Language) comme une extension au langage RDF (Resource Description Fra-

mework) (Lassila *et al.*, 1999) et comme la solution retenue dans le domaine du Web Sémantique pour l'expression des ontologies. OWL se décompose notamment en trois sous langages : OWL-Lite (dédié aux taxonomies et contraintes simples), OWL-DL (Permettant un niveau d'expressivité plus grand, mais garantissant des conclusions calculables et un temps de calcul fini) et OWL-Full (Destiné à une possibilité d'expressivité totale, mais sans garantie sur la calculabilité).

Ces idées et approches ont fortement inspiré notre démarche, mais afin de réussir une application dans le cadre de notre étude, il nous a cependant fallu répondre à un certain nombre de questions : Comment guider l'architecte système dans la production d'artefacts conceptuels de la taille désirée, possédant les bonnes propriétés pour faciliter l'analyse des exigences, et surtout, permettre une certaine capitalisation dans le cas du développement d'une ligne de produit.

3. Notre approche

Nous abordons dans cette partie la description de notre approche : Dans un premier temps, nous présentons les travaux de métamodélisation permettant le support d'un modèle conceptuel de système. Nous abordons ensuite une description de la démarche outillée de modélisation conceptuelle (le langage de modélisation, les outils que nous avons réalisé...).

3.1. *Modèle Conceptuel de Système*

L'objectif de notre approche est d'assister l'ingénierie système à mieux définir ses systèmes, à produire des spécifications de meilleure qualité afin d'éviter une incompréhension ou un manque qui seraient lourd de conséquence au moment de l'intégration. Nous avons vu dans le paragraphe précédent un certain nombre de techniques permettant d'améliorer la qualité des spécifications comme la définition du domaine. Après un certain nombre d'interview d'ingénieurs système à THALES, nous nous sommes rendu compte que nombre d'entre eux ont déjà utilisé ce principe de définition conceptuelle du domaine même si ce dernier n'était pas formalisé. Il s'agissait par exemple pour les membres d'une équipe de design, de faire tenir sur un tableau l'ensemble des concepts d'un système, de visualiser les interactions, les décompositions de celui ci. Cependant, même si une analyse fonctionnelle est toujours faite, cette pratique de construction visuelle et partagée du modèle mental du système a été abandonnée du fait de l'"explosion" du nombre de concepts, d'interactions. Chaque ingénieur possède néanmoins un modèle mental assez complet des concepts système. Les dangers que nous avons identifié avec eux proviennent de la complexification de ces derniers (et donc de "trous" dans le modèle mental), de l'arrivée de nouveaux collaborateurs, du transfert du savoir et éventuellement d'oublis possible devant une telle complexité à décrire.

Afin d'aider à clarifier ces points, nous avons proposé l'élaboration d'un modèle conceptuel. Nous entendons par "Modèle Conceptuel Système" un ensemble d'"Ontologies" (dans le sens donné par Gruber (Gruber, 1993)) qui doit permettre de représenter, hiérarchiser et associer entre eux l'ensemble des concepts de l'ensemble des domaines impliqués dans la réalisation d'un système. Ce Modèle Conceptuel a plusieurs objectifs : Procéder à une formalisation des modèles mentaux des ingénieurs système afin de permettre l'interprétation du contenu sémantique des exigences, mais également permettre le transfert du savoir vers de nouveaux collaborateurs, voire de capitaliser, réutiliser les concepts déjà décrits quand il s'agit d'une ligne de produit. Les résultats directs attendus étant dans un premier temps une amélioration de l'écriture des exigences par une meilleure maîtrise de la définition des concepts et sous-concepts. Dans un deuxième temps, nous devons être capables de capitaliser sur des concepts existants validés sur affaire pour ne pas oublier d'élément de spécifications. Nous proposons pour ce faire un métamodèle spécifique à la représentation d'éléments de système. Inspirés par les travaux de Consel (Consel *et al.,* 2007) sur les descriptions ontologiques de systèmes informatiques pervasifs, nous nous sommes tout d'abord orienté vers une description ontologique plus classique avant de choisir un formalisme s'inspirant des composants. Les outils liés aux ontologies ne nous permettant pas de guider l'ingénieur système peu habitué à ce genre d'opération, nous avons choisi un métamodèle plus contraint et avec une sémantique plus forte, et surtout, proche des usages des ingénieurs. Cet aspect composant a été abordé en interview avec des experts et assez facilement accepté. Ceci vient du fait que de nombreux langages ou technique d'architectures (par exemple AADL (Feiler *et al.,* 2003) (Dissaux, 2004)) utilisent ce formalisme. Nebut (Nebut *et al.,* 2004) constate, dans le cadre du LDE, que "le système complet peut être décrit comme un composant contenant des sous-composants", ce que nous avons pu également constater au cours des interviews d'experts.

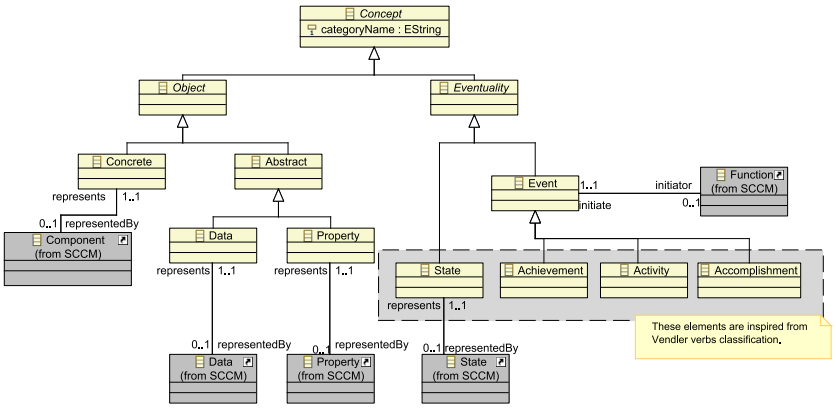


Figure 1. *Hiérarchie conceptuelle*

Ces constats et interviews nous ont inspiré dans la conception du métamodèle MCCS (Modèle de Composant Conceptuel de Système) permettant de représenter les éléments nécessaires à la représentation d'artefacts de niveau système.

Pour ce faire nous nous sommes inscrit dans la continuités des travaux de Grisvard (Grisvard, 2000) avec qui nous proposons la construction d'une hiérarchie conceptuelle décrite dans la Figure 1. Nous avons ensuite cherché à construire un métamodèle permettant de concrétiser les éléments décrits dans cette hiérarchie conceptuelle. Le Modèle MCCS a donc été bâti dans ce sens, en gardant la principale décomposition entre évènementiel et objets : Le Métamodèle (Figure 2) propose :

– Une représentation hiérarchique des composants du système : (Component) Permet de classifier un élément du système par rapport à un ensemble de concepts génériques abstraits (senseurs, calculateurs, interfaces...) ; permet également de capitaliser sur des concepts précédemment décrits dans une même ligne de produit. La décomposition en sous-composants (partitions du composant) sert à modéliser l'ensemble des

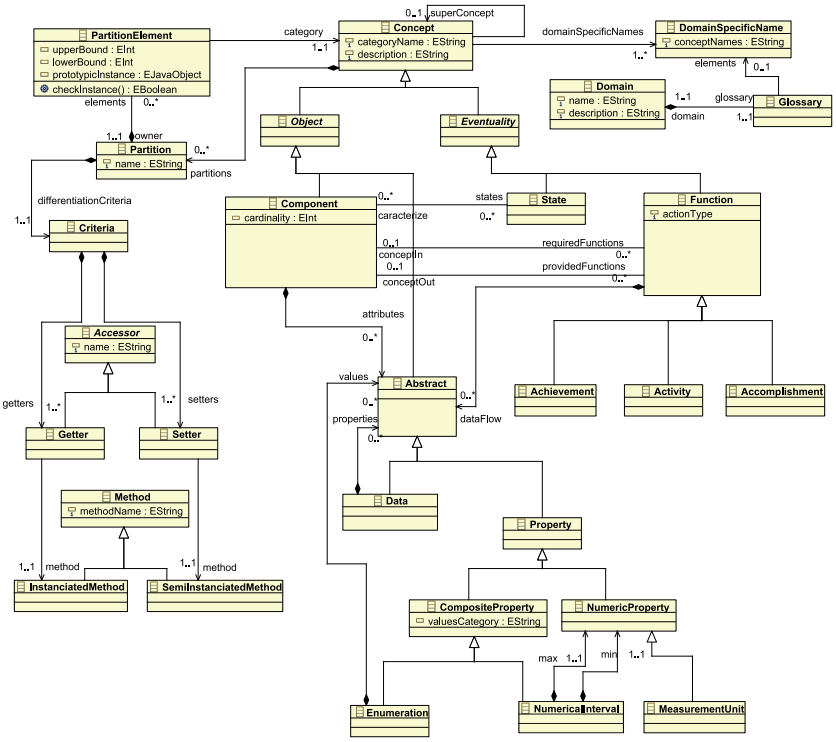


Figure 2. *Métamodèle Conceptuel*

concepts "partie de" notre concept père. Ils sont au niveau conceptuel, l'ensemble des catégories définissant le concept associé à une cardinalité (Les *roues* du vélo, cardinalité 2) ainsi que les attributs qui serviront de discriminant à l'instanciation (la roue *avant* : position)

– Une représentation hiérarchique des interaction entre composants : (Function) les fonctions requises et fournies par un composant. Chaque composant réfère à un ensemble de fonctions hiérarchisées (UpdateDatabase hérite de ManageDatabase pour représenter une forme de classification d'ordre général des fonctions). Ce genre d'héritage permet de généraliser des concepts fonctionnels et par exemple déclarer une contrainte sur l'ensemble des fonctions de gestion de la base de donnée. Les sous-fonctions "partie de" notre fonction mère représentent l'ensemble de la chaîne de fonctions à réaliser. Les fonctions agissent sur les composants (en altérant leurs états par exemple) et transportent ou modifient des données. Les *interfaces* nommées (non décrites explicitement dans la figure 1) déterminent des groupes logiques référençant certaines des "fonctions proposées" du composant conceptuel (ex : FLIRControlInterface : les fonctions de contrôle du FLIR.). Les concepts héritant récupèrent ou peuvent redéfinir ces éléments.

– Une représentation hiérarchique des états : Nous obtenons ainsi une caractérisation des états que l'on retrouve souvent dans les composants des systèmes (*AquisitionState* hérite de *OperationalState*)

– Une représentation hiérarchique des propriétés des composants : Les *propriétés* représentent des concepts valués définissant les propriétés propres à l'objet que l'on souhaite conceptualiser ou ses données.

– Une représentation hiérarchique des "acteurs" (non représenté dans le métamodèle) : Ce sont une famille de composants spécifiques interagissant avec le système par le biais des fonctions proposées par ce dernier. Un acteur ne fournit pas de fonctions, c'est par définition une source d'évènements. (un opérateur, un stimulateur externe...)

Le métamodèle est associé à un modèle de requêtes, et un ensemble de fonctions utilitaires servant à gérer les insertions, suppressions et modifications dans le modèle. Le langage qui a été choisi pour la métamodélisation et pour l'outillage est Smalltalk. Ce langage offre une très grande flexibilité pour les étapes de prototypage et nous a permis de modifier rapidement la structure du métamodèle. Le métamodèle permet à l'origine de concevoir un héritage multiple des concepts, mais nous préférons, sauf si ce n'est pas possible, une représentation hiérarchique reposant sur un héritage simple. Un multi-héritage sans restriction risquant de ne pas permettre une évolution simple dans le temps des modèles.

3.2. *Langage de Modélisation*

Associé à notre approche et au métamodèle de MCCS (SCCM en anglais), nous avons créé une suite outillée et un langage de modélisation en Smalltalk. Le langage

de modélisation LM-CCS repose sur le langage de modélisation de Smalltalk natif, tout en proposant une extension liée au multi-héritage.

La figure 3 représente l'interface principale d'édition et de visualisation des concepts. Elle permet un accès simple à la hiérarchie des concepts depuis la classe abstraite de ConceptualComponent. Cette interface permet de créer facilement un concept depuis un concept existant. Ce qui n'est pas visible dans cette figure concerne l'édition des nom spécifiques aux domaines d'un composant. (Les captures d'écran ont été réalisées sur un modèle d'exemple ayant servi pour une démonstration en interne.)

La figure 4 représente l'interface permettant d'effectuer une recherche dans l'ensemble des composants conceptuels déjà présents. Ceci peut être nécessaire quand la base de donnée de composant commence à devenir conséquente. Nous proposons à l'ingénieur de renseigner un certain nombre de critères de recherches : une liste de propriétés nommées (nommé dans l'outil "attribut"), une liste de fonctions requises ou fournies par le composant recherché. Nous pouvons accessoirement lister un certain nombre de mots clés qui seront recherchés dans la description des composants. Le résultat de la recherche propose alors une liste de choix correspondant aux critères. Ces résultats sont partiellement ou totalement compatibles avec la recherche. Dans les 2 cas, 2 valeurs accompagnent la recherche, un pourcentage de correspondance aux critères de recherche (si un composant possède tous les éléments désirés, il obtiendra 100%) ainsi qu'un indice de pertinence calculé par rapport aux nombre de critères (l'indice de pertinence étant fonction du nombre d'éléments recherchés par rapport au nombre d'éléments constituant le concept).

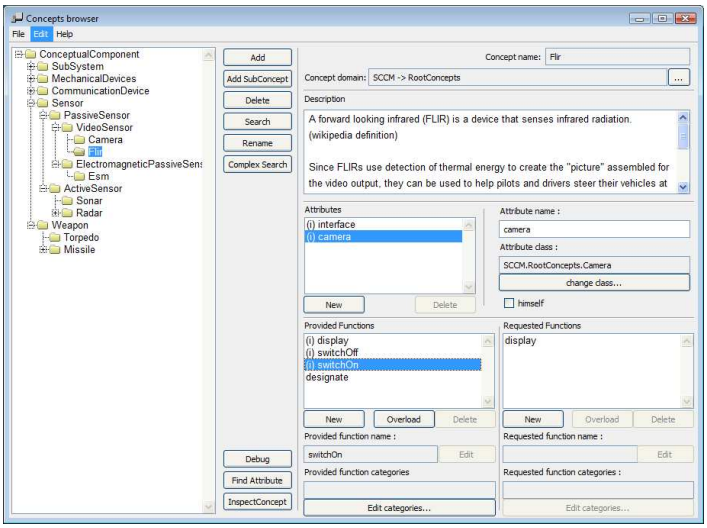


Figure 3. *Editeur hiérarchique de concepts*

La figure 5 représente l'interface principale d'édition et de visualisation des fonctions. Chaque fonction peut s'exprimer entre deux concepts (Exemple : une fonction de désignation peut être déclenchée par un *opérateur* sur un *sous-système*). Un mécanisme de nom spécifique à un domaine a été ici modélisé afin de pouvoir donner un nom spécifique à une fonction dans un ou plusieurs domaines spécifiques correspondant au métier. Une description de cette fonction est proposée ainsi qu'une liste de sous-fonctions décrivant sa décomposition (exemple : une désignation d'objectif se décompose en : sélectionner une cible, demander la désignation, déplacement du senseur...)

Nous avons également implémenté une grammaire simplifiée d'exigence utilisant SmaCC (Brant *et al.*, http : //www.refactory.com/Software/SmaCC/) afin de proposer la vérification d'exigences simples sur un modèle conceptuel. Scott et Cook (Scott *et al.*, 2004) proposent d'utiliser une grammaire plus ou moins adaptées à notre contexte, nous ne détaillerons pas ici notre grammaire qui en est une version simplifiée.

4. Exemple

Dans le cadre de ces travaux, nous avons effectué plusieurs études sur des documents de spécifications de niveau Système chez THALES. Un de ces document porte sur les spécifications d'un sous-système nommé FLIR, permettant l'affichage d'images de jour comme de nuit d'une situation relativement distante. Ce senseur est

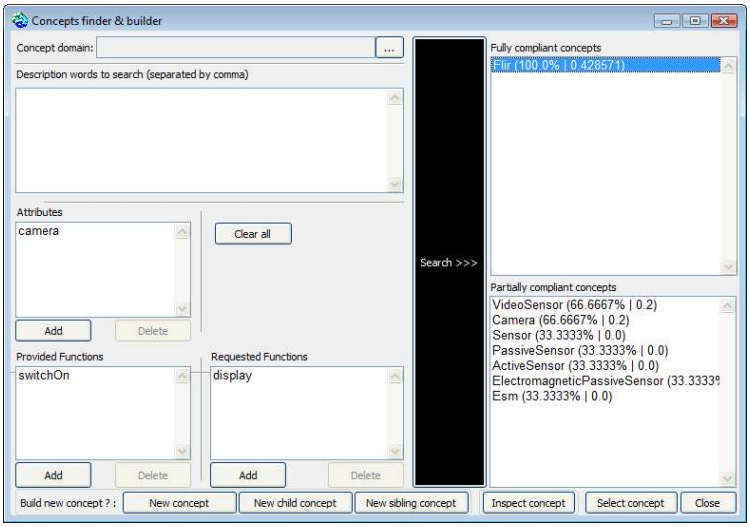


Figure 4. *Editeur hiérarchique de concepts*

utilisé dans le cadre de missions de patrouille maritime. Nous avons commencé l'expérience en tentant de reconstruire le modèle conceptuel par ingénierie inverse. Mais malgré les documentations et les spécifications qui étaient fournies, l'expert en modélisation n'a pas pu reconstruire un modèle conceptuel cohérent. Il manquait aux spécifications, les commentaires et les lumières d'un expert du domaine.

Nous allons prendre quelques exemples issus de cette étude et suivre la démarche de reconstruction du modèle (Les exigences sont exprimées en Anglais) :

Req. XX1 "*The FLIR/TV sensor video shall only be displayed on the Controller Operator and on the User Operator workstation.*".

Cette exigence est une exigence multiple et doit être réécrite, nous avons construit un modèle conceptuel complet du FLIR (le sous-système). Le sous-système FLIR possède un sous composant senseur qui lui même possède un composent de type vidéo. Chaque système peut être sollicité par un opérateur, et il existe dans ce cas deux types d'opérateurs, le contrôleur et l'utilisateur. Les opérateurs ont un poste de travail (workstation). Ici nous avons deux possibilité pour réécrire cette exigence :

1) Deux exigences :

- **FLIR/TV system Controller Operator** shall *Display* **FLIR/TV sensor Video** on **workstation** using **TCS User Interfaces** on.

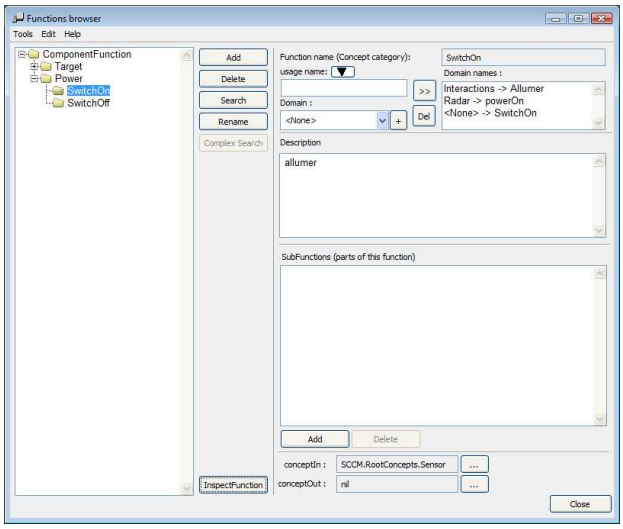


Figure 5. *Editeur hiérarchique de concepts*

- **FLIR/TV system User Operator** shall *Display* **FLIR/TV sensor Video** on **workstation** using **TCS User Interfaces**.

2) Une seule exigence exploitant la hiérarchisation des opérateurs contrôleur et utilisateurs (qui héritent dans le modèle d'opérateur) :

- *Any* **FLIR/TV system Operator** shall *Display* **FLIR/TV sensor Video** on **workstation** using **TCS User Interfaces**.

L'affichage de la vidéo n'est rendu possible que par un affichage sur la station de travail de chaque opérateur. ("Workstation" est exprimée relativement au sujet de la phrase.) Nous précisons ici de quel opérateur nous parlons, de quelle vidéo il s'agit, et par où se déclenche l'action. L'ingénierie logicielle peut ensuite plus facilement décomposer le problème (un manque de clarté par définition implicite peut parfois induire en fin de projet un redéveloppement couteux pour le fournisseur).

Autre exemple :

Req. XX2 "*The control response time for HCU entry shall be less or equal at X,X s*"

Le HCU est un périphérique de contrôle manuel du senseur. En lisant l'exigence, on ne peut être sûr de ce qui se voit contraint en temps de réponse. La notion de "control response time" peut vouloir dire beaucoup de choses... Nous proposons de préciser cette exigence en utilisant le modèle conceptuel :

"*Every* **FLIR/TV Sensor HCU Control function** shall have a **response time** less or equal to X,X s"

Nous avons précisé ici que toutes les "fonctions de contrôle" du HCU (groupement de fonction du périphérique, ne comprenant pas, par exemple les fonctions de calibrage ou d'énergie du concept) seront contraintes par leur attribut "response time" d'une valeur inférieure ou égale à X,X secondes. Nous précisons l'élément conceptuel impacté par la contrainte, et lors de la gestion de traçabilité de cette exigence vers un élément d'architecture, il pourra être récupéré un élément de modèle comme contrainte. Une analyse de l'attribut générique "response time" des fonctions contenues dans la contrainte pourra induire la production d'une vérification automatique sur le modèle d'architecture.

Le métamodèle MCCS inclut un métamodèle d'unités (que nous ne détaillerons pas dans cet article) permettant une capture rapide du type de contrainte (altitude, temps, vitesse...). Par exemple, deux exigences contenant des contraintes incohérentes en terme de type ou de valeurs sont détectables.

La complétude des exigences peut être proposée par l'observation du nombre d'éléments du modèle conceptuel utilisés par rapport à celui qui est exigé pour la construction du système : Il convient alors de vérifier le nombre de fonctions du mo-

dèle conceptuel non utilisées, ou encore le nombre de propriétés des composants non impactées par les exigences.

En effet, le système étant contraint par les exigences de niveau système, ces exigences sont censées décrire l'ensemble des propriétés fonctionnelles et non-fonctionnelles s'appliquant pour ce système. Il convient après analyse de l'ensemble des exigences de vérifier qu'aucune contrainte n'est oubliée par rapport aux éléments du modèle conceptuel. L'analyse sémantique utilise en effet les éléments du modèle conceptuel du système en cours de réalisation (composants, fonctions, etats, propriétés...). Notre idée est dans un premier temps d'analyser l'ensemble des concepts non utilisés en rapport avec le systeme et de valider le fait que leur utilisation n'était pas nécessaire à la description par les exigences. Dans le cadre d'une ligne de produit, l'ensemble des concepts d'un produit antérieur réutilisés pour le produit courant seront vérifiés en priorité.

5. Conclusion et perspectives

Nous avons décrit dans cet article une approche pragmatique permettant la capture d'un modèle conceptuel qui n'était bien souvent qu'une connaissance d'expert difficilement transmise. Nous proposons une alternative aux spécifications utilisant un langage formel par l'utilisation d'un modèle conceptuel associé a des outils d'analyse d'exigences. Nous proposons un formalisme sous forme de composants afin d'obtenir un ensemble d'éléments réutilisables dans le cadre de lignes de produit, mais également un support à la formation lors de l'arrivée de nouveaux collaborateurs. Ces modèles conceptuels devienne donc un élément critique, une formalisation du savoir des ingénieurs système. Un métamodèle MCCS au format Ecore, décrit en figure 2, et KM3 sont en cours d'élaboration de afin de préparer une transition des outils vers l'environnement Eclipse. Les prochains travaux doivent nous conduire à la réalisation d'un modèle conceptuel de taille plus conséquente afin de préparer une vérification du contenu des exigences sur de futurs systèmes. Nous allons également proposer un enrichissement de la méthodologie THALES de modélisation des systèmes par la vérification de règles en provenance des exigences analysées en exploitant les liens de traçabilité (voir exemple). Nous souhaitons intégrer l'outil de modélisation conceptuel à un outil utilisé par THALES, ce qui nous permettra une analyse plus poussée du langage que notre grammaire d'exigences.

Remerciements

Les auteurs souhaitent remercier les ingénieurs de THALES ayant participé aux interviews, Felipe Simon pour l'expérimentation de modélisation, ainsi que Vincent Verbeque, Jean-Luc Voirin, Gabriel Marchalot, Alexandre Skrzyniarz et Jean-Pierre Mével pour leur participations à l'élaboration de l'approche.

6. Bibliographie

Azelborn B., « Building a Better Traceability Matrix with DOORS », *Telelogic INDOORS US*, 2000.

Bézivin J., Blay M., Bouzhegoub M., Estublier J., Favre J.-M., Gérard S., Jézéquel J.-M., « Rapport de Synthèse de l'AS CNRS sur le MDA », CNRS, November, 2004.

Brant J., Roberts D., « SmaCC, a Smalltalk Compiler-Compiler », http ://www.refactory.com/Software/SmaCC/.

Brisson L., Intégration de connaissances expertes dans le processus de fouille de données pour l'extraction d'informations pertinentes, Phd, Université de Nice Sophia-Antipolis, 2006.

Consel C., Jouve W., Lancia J., Palix N., « Ontology-Directed Generation of Frameworks for Pervasive Service Development », *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, IEEE Computer Society Washington, DC, USA, p. 501-508, 2007.

Dissaux P., « Using the AADL for mission critical software development », *2nd European Congress ERTS, EMBEDDED REAL TIME SOFTWARE-21*, 2004.

Dobson G., Sawyer P., « Revisiting Ontology-Based Requirements Engineering in the age of the Semantic Web. International Seminar on" Dependable Requirements Engineering of Computerised Systems at NPPs », *Institute for Energy Technology (IFE), Halden*, 2006.

Feiler P., Lewis B., Vestal S., « The SAE Avionics Architecture Description Language (AADL) Standard : A Basis for Model-Based Architecture-Driven Embedded Systems Engineering », *RTAS 2003 Workshop on Model-Driven Embedded Systems*, 2003.

Fleurey F., Le Traon Y., *Langage et méthode pour une ingénierie des modèles fiable*, PhD thesis, Université Rennes 1, 2006, 2006.

Gamma E., Helm R., Johnson R., Vlissides J., *Design patterns : elements of reusable object-oriented software*, Addison-Wesley Reading, MA, 1995.

Grisvard O., Modélisation et gestion du dialogue homme-machine de commande, Phd, Université Henry Poincaré - Nancy 1, 2000.

Gruber T., « A translation approach to portable ontology specifications », *Knowledge Acquisition*, vol. 5, n° 2, p. 199-220, 1993.

Hammond J., Rawlings R., Hall A., « Will It Work ? », *RE '01 : Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, IEEE Computer Society, Washington, DC, USA, p. 102-109, 2001.

Jézéquel J.-M., Gérard S., Baudry B., *L'ingénierie dirigée par les modèles*, Lavoisier, Hermes-science, chapter Le génie logiciel et l'IDM : une approche unificatrice par les modèles, 2006.

Kaiya H., Saeki M., « Ontology Based Requirements Analysis : Lightweight Semantic Processing Approach », *QSIC '05 : Proceedings of the Fifth International Conference on Quality Software*, p. 223-230, 2005.

Lassila O., Swick R. et al., « Resource Description Framework (RDF) Model and Syntax Specification », 1999.

Nebut C., Jézéquel J., « Génération automatique de tests à partir des exigences et application aux lignes de produits logicielles », *Rennes, Université de Rennes*, 2004.

Nuseibeh B., Easterbrook S., « Requirements engineering : a roadmap », *Proceedings of the Conference on The Future of Software Engineering*, ACM Press New York, NY, USA, p. 35-46, 2000.

Sawyer P., Sommerville I., Viller S., « Requirements process improvement through the phased introduction of good practice », *Software Process : Improvement and Practice*, vol. 3, p. 19-34, 1997.

Scott W., S.C. C., « A Context-free Requirements Grammar to Facilitate Automatic Assessment », *Proc of the Australian Requirements Engineering Workshop, Adelaide*, December, 2004.

Sommerville I., *Software Engineering (7th Edition)*, Pearson Addison Wesley, 2004.

Sommerville I., *Software Engineering (8th Edition)*, Pearson Addison Wesley, 2007.

Spivey J., « Understanding Z : A Specification Notation and its Formal Semantics », *Cambridge Tracts in Theoretical Computer Science*, 1988.

Wilson W. M., Rosenberg L. H., Hyatt L. E., « Automated analysis of requirement specifications », *ICSE '97 : Proceedings of the 19th international conference on Software engineering*, ACM, New York, NY, USA, p. 161-171, 1997.

Wilson W., Rosenberg L., Hyatt L., « Automated Quality Analysis of Natural Language Requirement Specifications », *Proceedings of Fourteenth Annual Pacific Northwest Software Quality Conference*, 1996.

Wordsworth J., *Software engineering with B*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1996.

Formalisation de bonnes pratiques dans les procédés de développement logiciels

Vincent Le Gloahec* — **Regis Fleurquin**** — **Salah Sadou*****

*Alkante SAS, Rennes, France
**IRISA/Triskell, Campus Universitaire de Beaulieu, Rennes, France
***Laboratoire VALORIA, Université de Bretagne Sud, Vannes, France
v.legloahec@alkante.com, regis.fleurquin@irisa.fr, salah.sadou@univ-ubs.fr

RÉSUMÉ. *Les bonnes pratiques sont des techniques et des méthodes, capitalisées par une entreprise ou par une communauté, dont l'efficacité a été constatée lors des projets de développement. Les bonnes pratiques sont conservées par les entreprises au mieux de manière informelle au détour de guides ou de manuels qualité, au pire elles font partie d'une culture implicite fortement tributaire de certaines individualités. Dans cet article, nous proposons d'exprimer de manière formelle les bonnes pratiques orientées processus. Cela offre deux avantages : d'une part une conservation de manière non ambiguë et structurée et d'autre part la possibilité d'automatiser le contrôle de leur respect. Nous évaluons l'intérêt du langage SPEM pour exprimer les bonnes pratiques. Une expérimentation a été menée avec un partenaire industriel spécialisé dans des projets de développement Web.*

ABSTRACT. *Good practices are techniques and methods capitalized by a company or a community, which effectiveness has been observed in development projects. Those good practices are kept in companies at best in a informal manner through guidelines or quality manuals, and at worst they form a part of a tacit knowledge which strongly depends on some individuals. In this paper, we propose to express in a formal manner those good practices. This approach offers two advantages : on the one hand, the storage in a non ambiguous and structured way, and on the other hand, the possibility to automate the checking of their conformance. We evaluate the relevance of the SPEM language to express good practices. An experiment has been carried out in collaboration with our industrial partner, which is specialized in Web development projects.*

MOTS-CLÉS : *Bonnes pratiques, Ingénierie des procédés, Ingénierie des modèles, SPEM, OCL*

KEYWORDS: *Good practices, Process engineering, Model-driven engineering, SPEM, OCL*

1. Introduction

Afin de mieux cerner les activités de développement logiciel, d'identifier les liens de cause à effet, et in fine de dégager des zones de gain potentiel, de nombreuses études empiriques ont été conduites. Ces études ont permis d'identifier des "bonnes pratiques" (BP) qui, lorsqu'elles sont mises en oeuvre de manière pertinente, contribuent à augmenter l'efficacité des développements sur le plan de la qualité et de la productivité. Ces pratiques d'origine interne ou externe, mobilisées à bon escient lors des projets, constituent l'expertise, la valeur ajoutée d'une entreprise. C'est un capital précieux pour garantir la satisfaction de ses clients, se distinguer, briguer des labels, des certificats et faire face à la concurrence (Gratton *et al.,* 2005).

Mais faute de moyens qu'elles jugent adéquats, la majorité des entreprises ne rationalisent pas la gestion de ce type de connaissance. Les rares entreprises qui tentent de capitaliser ces connaissances le font au travers de documents informels, souvent incomplets, parfois mal référencés et épars, conduisant à un usage inadéquat et donc inefficace des BP (Shull *et al.,* 2005). Des travaux proposent d'introduire des processus et des outils informatiques qui facilitent le stockage, le partage et la diffusion de BP au sein des entreprises (Fragidis *et al.,* 2006), (Zhu *et al.,* 2007). Cependant, les BP référencées par ces systèmes ne sont consultables au final qu'au travers de documents textuels et informels.

Nous proposons dans cet article de rendre productives les BP. Nous montrons que, dès lors, il est possible d'adopter une démarche d'ingénierie dirigée par les modèles ; une démarche qui facilite, d'une part le processus de capitalisation des BP et, d'autre part, la vérification automatique de leur respect dans les outils offrant des mécanismes de contrôle de l'activité de développement. Nous nous focalisons, dans la suite du papier, uniquement sur le verrou qui pèse sur la première étape de cette démarche : la définition d'un langage dédié, acceptable par les développeurs et les responsables qualité, qui permettrait de rendre productives les BP. Nous présentons ensuite le langage SPEM et son intérêt théorique pour asseoir les fondations d'un tel langage, en nous appuyant sur une expérimentation conduite au sein d'une entreprise partenaire pour évaluer, cette fois-ci in vivo, l'intérêt pratique de SPEM.

2. Une démarche pour la gestion des bonnes pratiques

Notre objectif est, à terme, comme le montre la figure 1, de bâtir un pont entre, d'une part l'univers des dépôts et d'autre part celui des outils offrant des moteurs capables de vérifier le respect de BP. Nous proposons, pour cela, d'adopter une démarche d'ingénierie dirigée par les modèles (IDM) en rendant les BP productives.

Il est important que les BP soient formalisées et stockées dans les dépôts sous la forme de modèles dédiés de niveau PIM, c'est-à-dire indépendant de tout outil. Cette indépendance vis-à-vis des outils est la seule garantissant un savoir pérenne capable de survivre aux évolutions et changements affectant inévitable-

ment les plates-formes de développement. La capitalisation s'opèrerait donc à ce niveau PIM dans un langage dédié aussi puissant et accessible que possible.

Il devient envisageable de transformer, à la demande, de manière aussi automatique que possible les modèles PIM en modèles de niveau PSM conformes aux langages interprétables par les extensions de chaque outil. La mise en place effective d'une telle démarche suppose la levée de deux verrous : i) l'existence d'un langage de niveau PIM pour exprimer les BP, ii) la définition de transformations pour passer des modèles PIM aux modèles PSM. Dans la suite de cet article, nous nous consacrerons uniquement au premier de ces deux verrous.

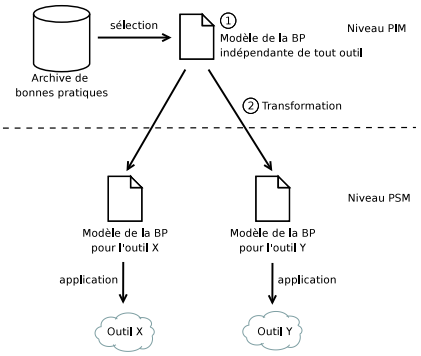


Figure 1. *Démarche de transformation des BP.*

3. Nature des bonnes pratiques

Les BP sont des techniques et des méthodes, capitalisées par une entreprise ou par une communauté, dont l'efficacité a été constatée lors des projets de développement. Dans une entreprise, les BP capitalisées proviennent de deux types de source : des sources externes à l'entreprise (en provenance d'autres entreprises par Benchmarking, d'articles, de rapports, de guides, de méthodes, d'ouvrages faisant autorité, etc.) et des sources internes (par analyse de sa propre activité). Au final, on constate une extrême variété des BP sur le plan de la forme, du niveau d'abstraction et du contexte d'usage : BP de conception, d'architecture, de codage, etc.

Dans cet article, nous nous intéressons uniquement aux BP orientées processus. On peut trouver des exemples de telles BP dans (Thayer, 2002), (McConnell, 1993) et (Phillips, 2004). Nous qualifions une BP "d'orientée processus" lorsqu'elle exprime des contraintes sur la forme que peut prendre tout ou partie d'un processus de développement. Souvent ces BP s'expriment sous la forme d'activités ordonnées agrégées sous la forme de processus. Le respect par exemple d'un cycle de développement en V peut être considéré comme une BP orientée processus. Son expression revient à donner le graphe de séquencement des activités à mener lors d'un projet avec la liste des documents en entrée et sortie de chacune des étapes.

4. SPEM pour modéliser les bonnes pratiques

Dans cette section nous allons évaluer l'intérêt d'un langage comme SPEM porté par l'OMG pour modéliser les BP. Nous décrirons, en particulier, deux stratégies possibles de modélisation, avant de conclure avec quelques exemples de formalisation de BP à l'aide d'une stratégie implicite.

Nous venons d'illustrer le fait que nombre de bonnes pratiques (BP) intègrent des aspects processus, d'où la nécessité de pouvoir représenter les activités, leur séquencement, les personnes et artefacts qui les composent. L'OMG a justement proposé un langage, SPEM, dédié à l'ingénierie des procédés et intégrant l'ensemble de ces aspects (OMG, 2008). Le standard SPEM2.0 se présente sous deux formes équivalentes : un méta-modèle complet exprimé en MOF et s'appuyant sur le méta-modèle UML2.0, et un profil UML. L'intérêt majeur de ce standard est qu'il présente la particularité d'avoir sa syntaxe abstraite décrite à l'aide de MOF. Il est donc utilisable en l'état dans une démarche IDM avec les outils de transformation existant. Il est aussi possible de raffiner SPEM standard en usant du mécanisme de profil défini dans UML.

Ces raisons font qu'il était naturel d'évaluer la pertinence de ce langage pour servir de base à un futur langage de modélisation de BP. Ce qui nous intéresse maintenant est de savoir comment des BP, souvent décrites de manière informelle, peuvent être représentées dans le contexte de l'ingénierie des procédés à l'aide de SPEM. On peut toutefois distinguer deux stratégies possibles pour modéliser des BP en utilisant le langage SPEM : par modélisation explicite ou par modélisation implicite. Nous allons détailler successivement chacune de ces stratégies.

4.1. *Deux stratégies de modélisation*

Il est d'une part possible d'user d'une stratégie par modélisation explicite qui consiste à modéliser directement les bonnes pratiques sous la forme de modèles SPEM à part entière, en les décrivant sous la forme de processus particuliers ou comme des éléments méthodologiques. Cette stratégie a le mérite de permettre un usage complet du langage SPEM pour décrire une BP. En effet, la BP existe au travers d'un modèle SPEM. Le principal inconvénient de cette stratégie est de déconnecter de tout contexte la bonne pratique. Celle-ci ne peut être positionnée dans tel ou tel étape, processus ou domaine.

D'autre part, la stratégie par modélisation implicite consiste à représenter une BP non pas en SPEM mais comme un ensemble de contraintes OCL venant contraindre un processus SPEM particulier défini à un niveau entreprise (un processus générique) ou projet (une instance particulière de processus générique). Cette stratégie permet cette fois-ci de contextualiser la BP en l'associant à un processus cible. Mais elle limite le pouvoir d'expression à disposition à celui d'OCL. L'inconvénient de cette stratégie implicite est qu'une BP ainsi exprimée est, à l'usage, clairement moins puissante, lisible et maintenable que pour la stratégie précédente.

4.2. *Expérimentation dans un contexte de développement Web*

Face à la complexité croissante des projets et aux problématiques liées à la maintenance logicielle, notre partenaire industriel Alkante tend à faire évoluer ses processus de développement classiques. Dans ce contexte, nous proposons un retour d'expérience sur l'usage des langages SPEM et OCL pour la formalisation de BP que souhaite documenter la société. Il s'agit aussi bien de BP de la littérature issues de méthodes reconnues que de pratiques internes émanant de l'expérience acquise par la société lors de ses développements.

La méthode RUP, par exemple, propose de nombreuses BP d'ordre général. Il est souvent nécessaire de raffiner les règles générales afin de pouvoir les exprimer et les formaliser en OCL sur le méta-modèle SPEM. En utilisant uniquement les constructions offertes par SPEM2.0 et le stéréotype *Iteration*, la pratique « Une itération est une activité de haut niveau, qui doit être répétée dans le temps, et qui se décompose en sous-activités » peut être exprimée en OCL de la façon suivante :

```
context: Iteration
inv: self.isRepeatable = true and
     self.nestedBreakdownElement->asSet()
     ->select( be:BreakdownElement | be.oclIsKindOf( Activity ) )
     ->asSet()->notEmpty()
```

Une autre pratique, spécifique à la société, consiste cette fois-ci à saisir à l'aide d'un outil dédié les temps passés par chaque développeur lors des différentes phases d'un projet, ou de toute autre activité de production (documentation, recherche, maintenance, etc.). Dans SPEM, il est possible de représenter cette pratique sous la forme d'une tâche intitulée « Saisie hebdomadaire des temps passés », et d'une contrainte OCL associée. Pour cela, le stéréotype "SaiseDesTemps" qui est de type *TaskUse* est défini dans un Profil SPEM pour RUP spécifique à l'entreprise, et la BP peut être modélisée comme suit :

```
context: SaisieDesTemps
inv: self.isPlanned = true and
     self.planningData.oclIsTypeOf( WeeklyPlannedData ) and
     self.planningData.oclAsType( WeeklyPlannedData ).dayOfWeek=1
```

5. Conclusion

Les bonnes pratiques intégrant souvent des aspects process, nous avons évalué l'intérêt du langage SPEM pour servir de base à la formalisation de BP orientées processus. Notre étude a montré que SPEM se révèle compliqué à appréhender par les développeurs. Il se montre également souvent trop abstrait pour décrire certaines BP. Leur expression suppose l'existence de profils dont la définition reste à la charge de l'entreprise. Les profils spécialisent les concepts manipulés ; des contraintes OCL permettent alors la formulation d'une expression, sur la base de ces concepts.

Passées ces quelques difficultés SPEM se montre à l'aise dans son domaine de prédilection : la modélisation des aspects processus apparaissant dans les BP. Les problèmes se posent de manière insoluble lorsqu'il faut tisser de manière cohérente des aspects processus et modèles inhérents à certaines pratiques. Cette expérimentation a également mis en valeur le besoin d'organiser les BP selon des structures multiples de type hiérarchique, voire de type graphe : selon leur granularité (certaines se construisant comme agrégation d'autres) et selon leur niveau d'application (générales à l'entreprise, propres à un projet, à un domaine, etc.), selon leur niveau d'abstraction dans la chaîne IDM (PIM, PSM, avec éventuellement des stades intermédiaires). Cette structuration et les aspects de tissage entre modèles et processus seront au coeur de nos préoccupations pour l'élaboration d'un langage basé sur SPEM dédié à la formalisation des bonnes pratiques. Sur le second point nous pensons nous inspirer des travaux qui abordent l'ingénierie des méthodes de Génie Logiciel (Henderson-Sellers, 2003), (Gonzalez-Perez *et al.,* 2007).

6. Bibliographie

Fragidis G., Tarabanis K., « From Repositories of Best Practices to Networks of Best Practices », *Management of Innovation and Technology, 2006 IEEE International Conference on*, vol. 1, p. 370-374, June, 2006.

Gonzalez-Perez C., Henderson-Sellers B., « Modelling software development methodologies : A conceptual foundation », *J. Syst. Softw.*, vol. 80, n° 11, p. 1778-1796, 2007.

Gratton L., Ghoshal S., « Beyond Best Practices », *Sloan Management Review*, vol. 46, n° 3, p. 49-57, 2005.

Henderson-Sellers B., « Method engineering for OO systems development », *Commun. ACM*, vol. 46, n° 10, p. 73-78, 2003.

McConnell S., *Code complete : a practical handbook of software construction*, Microsoft Press, Bellevue, WA, USA, 1993.

OMG, « Software and Systems Process Engineering (SPEM), formal/2008-04-01 », http ://www.omg.org/spec/SPEM/2.0/, 2008.

Phillips D., *The Software Project Manager's Handbook : Principles That Work at Work*, John Wiley & Sons, 2004.

Shull F., Turner R., « An empirical approach to best practice identification and selection : the US Department of Defense acquisition best practices clearinghouse », *Empirical Software Engineering, 2005. 2005 International Symposium on*, vol. , p. 8 pp.-, Nov., 2005.

Thayer R. H., *Project Manager's Guide to Software Engineering's Best Practices*, IEEE Computer Society Press, Los Alamitos, CA, USA, 2002.

Zhu L., Staples M., Gorton I., « An Infrastructure for Indexing and Organizing Best Practices », *REBSE '07 : Proceedings of the Second International Workshop on Realising Evidence-Based Software Engineering*, IEEE Computer Society, Washington, DC, USA, p. 4, 2007.

Ingénierie dirigée par les modèles pendant la phase de spécification du besoin

Benjamin Chevallereau[*,***] **— Alain Bernard**[*] **— Pierre Mévellec**[**]

[*] *École Centrale de Nantes - IRCCyN - 1, rue de la Noë - BP 92 101 - 44321 Nantes Cédex 03*

{benjamin.chevallereau,alain.bernard}@irccyn.ec-nantes.fr

[**] *IAE de Nantes - Rue de la Censive du Tertre - BP 62 232 - 44322 Nantes Cédex 3*

pierre.mevellec@univ-nantes.fr

[***] *BlueXML - 40, boulevard Jean Ingres - 44100 Nantes*

RÉSUMÉ. *L'ingénierie dirigée par les modèles a aujourd'hui montré la majorité de ces résultats dans la phase de développement logiciel et tout particulièrement avec l'approche MDA*^{TM}*. Tandis que cette phase est grandement étudiée par la communauté IDM, la phase de spécification et d'expression du besoin est, aujourd'hui, peu approfondie. Notre proposition repose sur la mise en œuvre de l'ingénierie dirigée par les modèles dans cette phase, qui semble être l'une des plus importantes dans le processus de développement logiciel, avec pour objectif d'améliorer la qualité de la spécification des besoins et ainsi apporter une information plus fiable et plus claire aux étapes suivantes. Cette proposition repose sur un méta-modèle de spécification du besoin fonctionnel et d'un mécanisme d'interprétation à l'aide de transformations de modèles.*

ABSTRACT. *Model driven engineering show, today, the most part of its results in the phase of software development and more particularly with the MDA*^{TM} *approach. Whereas this phase is greatly studied by the MDE community, the phase of specification and expression of needs is, today, not many detailed by this community. Our proposition is based on the implementation of MDE to improve the quality of specification of needs and to improve the communication between technical experts and functionnal profiles. This proposition is based on a meta-model, a web modeler and a set of interpretations.*

MOTS-CLÉS : *IDM, Ingénierie des besoins, Modeleur web, Transformation de modèles, Génération*

KEYWORDS: *MDE, Requirements engineering, Web modeler, Model transformation, Generation*

1. Introduction

Nous devenons de plus en plus dépendant des différents systèmes d'information auxquels nous avons accès. Cette dépendance peut nous permettre d'améliorer significativement notre efficacité mais présente également de nombreux risques de perte de productivité dans le cas d'inadéquation entre le processus et l'outil le supportant. On peut naïvement organiser l'ensemble des participants d'un processus de développement en deux catégories. La première catégorie regroupe l'ensemble des profils possédant des compétences spécifiques au métier concerné par le futur système à réaliser, c'est-à-dire les experts fonctionnels. La seconde catégorie regroupe les profils possédant des compétences techniques nécessaires à la réalisation du système, c'est-à-dire les experts techniques. Les profils appartenant aux deux catégories n'utilisent pas la même sémantique. Ils ne connaîssent pas le métier de l'autre et donc ne savent pas formuler clairement leurs idées sous une forme compréhensible. L'ingénierie des besoins a donc pour objectif d'améliorer cette communication afin de mieux comprendre les besoins des experts fonctionnels et ainsi réaliser un outil complet et adapté. Pour résumer, elle a pour tâche de fournir une spécification des besoins qui doit être aussi complète et documentée que possible en utilisant des formats variés et adaptés de représentation afin d'établir une communication suffisante entre les différents participants impliqués (Pohl, 1996).

Traditionnellement, l'ingénierie des systèmes d'information se concentre sur la modélisation conceptuelle. Celle-ci vise à abstraire la spécification du système requis à partir de l'analyse des informations nécessaires à la communauté des utilisateurs (Rolland, 2007). Bien que la modélisation conceptuelle permet aux experts techniques de comprendre la sémantique du domaine, elle ne permet pas de construire des systèmes acceptés par la communauté des utilisateurs. Afin d'apporter des réponses adaptées à la communauté des utilisateurs, il est nécessaire de considérer les systèmes d'information comme un moyen d'atteindre un but déterminé dans une organisation. Comprendre ce but est une condition nécessaire à la conceptualisation d'un système. Il est important de dépasser la définition des fonctionnalités basée sur la vue du modèle conceptuel et d'étendre l'approche *que doit réaliser le système* vers *pourquoi le système est nécessaire*. La seconde question permet d'extraire les objectifs organisationnels et leur impact sur le système d'information.

Pour résumer, notre proposition repose sur trois piliers représentés sur la figure 1. La force de cette proposition est le méta-modèle de spécification du besoin fonctionnel (a) qui est décrit plus longuement par la suite. Pour être exploité de manière optimale, celui-ci est accompagné d'un outil de modélisation (b) adapté aux experts fonctionnels. La spécification du besoin est ensuite considérée source de l'ensemble des interprétations (c). Chacune des interprétations à un public cible, une fonction donnée et un mécanisme particulier. Le concept d'interprétation sera décrit dans les sections suivantes.

La section 2 présentera le méta-modèle proposé pour la spécification d'un besoin utilisateur et le mécanisme d'interprétation sera expliqué dans la section 3.

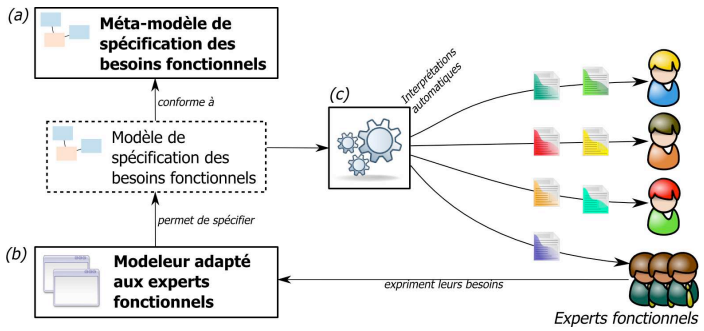


Figure 1. *Schéma global de notre proposition*

2. Un méta-modèle de spécification des besoins

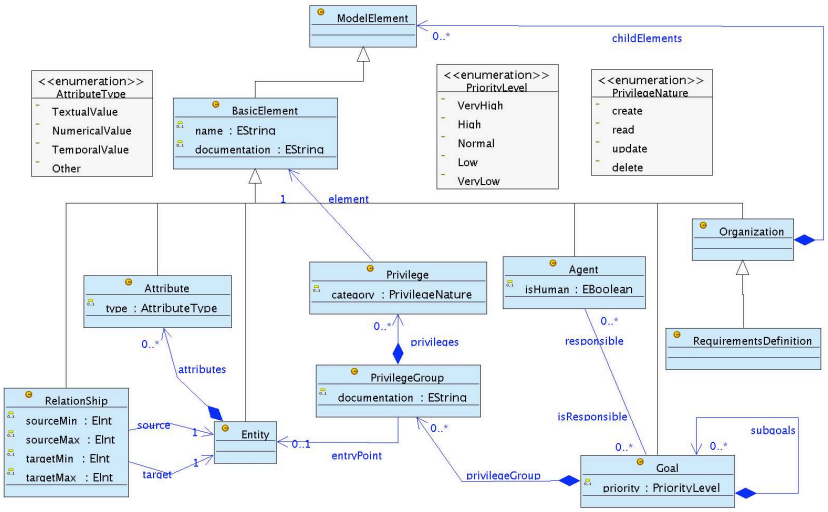


Figure 2. *Méta-modèle proposé pour la spécification du besoin utilisateur*

Notre méta-modèle s'articule autour de sept principaux concepts : *Entity*, *Relationship*, *Attribute*, *Organization*, *Agent*, *Goal* et *Privilege*. Ils peuvent être répartis dans quatre grandes familles de modélisation : la modélisation de l'organisation qui a pour objectif de comprendre la structure dans laquelle l'application sera utilisée (*Or-*

ganization et *Agent*) ; la modélisation des objectifs qui permet de décrire l'activité qui sera supportée par l'outil métier (*Goal*) ; la modélisation de la structure d'information qui est utile afin de décrire les différents termes utilisés, c'est-à-dire le dictionnaire de données utilisé dans l'entreprise (*Entity*, *Relationship* et *Attribute*) ; la modélisation des moyens qui permet de définir les moyens à mettre en œuvre afin d'atteindre un objectif (*Privilege*).

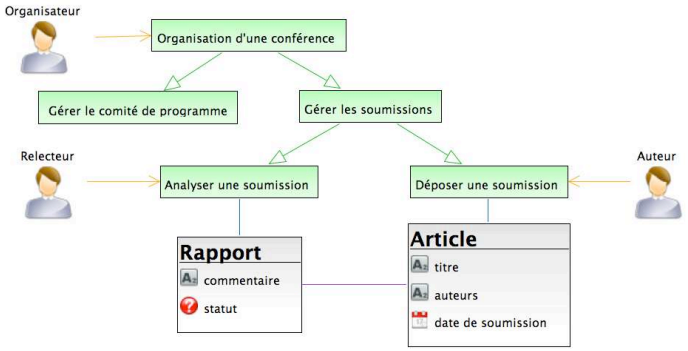


Figure 3. *Exemple de spécification du besoin dans le cas de l'organisation d'une conférence*

Exemple : On retrouve sur la figure 3 une expression partielle des besoins pour un système d'organisation de conférence. La partie à laquelle nous nous intéressons dans cet exemple est la soumission d'article. La première étape est de lire la partie concernant la modélisation des objectifs. L'objectif « Gérer les soumissions » se décompose en deux sous-objectifs. Tout d'abord, il y a la soumission avec l'objectif « Déposer une soumission » sous la responsabilité d'un auteur. Puis, il y a la re-lecture avec l'objectif « Analyser une soumission » sous la responsabilité d'un re-lecteur. La seconde étape se concentre sur la modélisation des informations, c'est-à-dire les entités et relations entre elles ainsi que les attributs caractérisant les entités. Nous pouvons considérer la définition des concepts métier comme un graphe. Celui-ci est réduit dans cet exemple à deux nœuds représentés par les entités *article* et *rapport* reliés entre eux par une relation. Après avoir défini ces deux modélisations, il est possible de les lier entre elles à l'aide des privilèges. Pour que les agents atteignent leurs buts dans le futur outil, nous allons leur définir un parcours dans le graphe des concepts, à l'aide des privilèges, pour chaque objectif sous leur responsabilité. Un parcours dans le graphe des concepts peut également être considéré comme une vue partielle sur le futur système d'information. La soumission d'un article nécessite un parcours assez simple. Il se résume à entrer dans le système par l'entité *article* puis de pouvoir en créer un nouveau. Ce parcours permet à l'auteur de renseigner le titre de l'article et ses auteurs. La re-lecture d'un article nécessite un parcours relativement plus complexe. Le point d'entrée dans le système est tout d'abord le *rapport*, un re-lecteur doit donc créer un

nouveau rapport, saisir son commentaire mais aussi définir quel est l'article qui est commenté. Ce but doit donc permettre de naviguer jusqu'au concept d'*article* pour récupérer l'information nécessaire. Malheureusement, la figure 3 ne permet pas de visualiser les privilèges accordés pour un souci de clarté du diagramme. Le seul lien visible pour le concept de privilège est le point d'entrée dans le système d'information pour chacun des objectifs.

3. Interprétation de l'expression du besoin

Afin d'illustrer l'intérêt du concept d'interprétation dans le contexte de spécification du besoin fonctionnel d'une application, nous pouvons utiliser le modèle de *Shannon* et *Weaver*, considéré par certains comme le modèle canonique de la communication (Shannon, 1948). Ce modèle est décrit par la phrase suivante : « *Un émetteur, grâce à un codage, envoie un message à un récepteur qui effectue le décodage dans un contexte perturbé de bruit*. » Si on applique ce modèle à l'industrie informatique actuelle, un expert fonctionnel, représenté comme l'emetteur, a un ensemble de besoins qu'il doit transmettre à un expert technique, représenté comme le recépteur. La spécification des besoins, représentant le message, doit donc être codée puis décodée pour ensuite être interprétée par l'expert technique. Le codage utilisé est celui des règles généralement utilisées dans le contexte de rédaction d'un cahier des charges. Celui-ci est ensuite décodé pour obtenir des spécifications techniques compréhensibles par l'expert technique. Le problème majeur de ce processus se localise dans l'étape de codage/décodage, dirigée par la maîtrise d'ouvrage, qui peut être perturbée par un bruit généré par l'environnement. Cet environnement est constitué, par exemple, des connaissances et des compétences de la maîtrise d'ouvrage mais aussi par ses expériences antérieures. Le résultat du processus de communication est donc très variable en fonction de l'étape centrale. Elle représente un facteur de risque important en raison de la quantité de bruit généré et de son impact. Afin de réduire la fragilité de ce processus, nous proposons d'automatiser un grand nombre d'interprétations. Ces interprétations vont nous permettre d'améliorer la spécification des besoins en supprimant la grande partie des bruits engendrés par une intervention humaine. Grâce à cette automatisation, nous allons également pouvoir faire intervenir un plus grand nombre de participants à l'élaboration de cette spécification. En effet, afin d'obtenir une spécification représentatrice des besoins réels, il est important de la faire valider par le plus grand nombre de profils dans l'entreprise. Cet ensemble peut regrouper les responsables de l'entreprise, des organisations impactées, du système d'information, de la qualité mais aussi les futurs utilisateurs.

Comme on peut le voir sur la figure 4, le mécanisme d'interprétation se décompose en trois étapes. Ce mécanisme prend en entrée un modèle de spécification des besoins fonctionnels conforme au méta-modèle proposé précédemment. La première étape est une transformation de modèles, réalisée avec ATL (Jouault *et al.*, 2006), qui permet de réaliser l'interprétation du besoin et donc concentre toute son intelligence. Cette transformation de modèles est contrainte à prendre un unique modèle en entrée conforme à

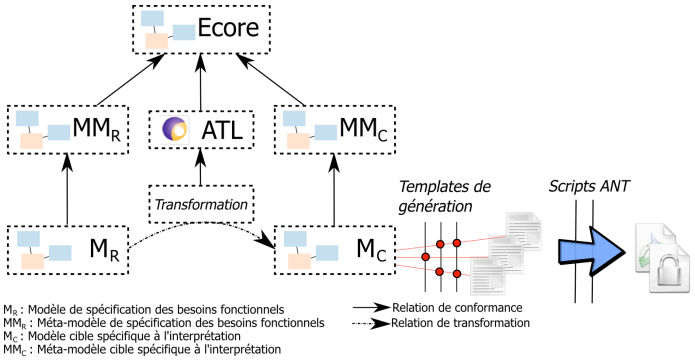


Figure 4. *Mécanisme d'interprétation*

notre méta-modèle et à retourner un unique modèle en sortie. Le méta-modèle cible est spécifique à l'interprétation. Il peut être relatif à une carte conceptuelle, à une documentation, à la modélisation UML, à l'évaluation des risques... La seconde étape est l'étape de génération. Celle-ci est réalisée à l'aide de templates de génération, en utilisant la technologie Acceleo (Obeo, 2008). Chaque template est appliqué sur le modèle issu de la transformation. Puis une étape supplémentaire est réalisée afin de réaliser des opérations difficilement réalisables par transformation ou génération à l'aide de scripts ANT (ANT, 2008). Ce mécanisme est entièrement automatisé et accessible à l'utilisateur à travers l'interface de l'outil de modélisation. Une interprétation est donc structurée sous la forme d'un triplet composé d'une transformation de modèles associée au méta-modèle cible correspondant (carte conceptuelle, documentation...), d'un ensemble de templates de génération s'appliquant sur le méta-modèle précédent puis d'un ensemble de scripts.

L'objectif de cet article n'est pas de présenter de manière approfondie l'ensemble des interprétations réalisées dans ce projet mais seulement de démontrer l'intérêt de cette approche par l'exemple. Il existe deux grandes familles d'interprétation. Il est important d'adapter la visualisation de l'expression des besoins aux différents profils qui devront valider les spécifications fonctionnelles. Cette adaptation regroupe donc l'ensemble des transformations constituant la première famille. La seconde regroupe les transformations dites de « traduction ». Leur objectif est de lire l'expression des besoins, la comprendre et enfin la traduire sous diverses formes. On peut citer les transformations d'analyse, de vérification, de documentation ou de traduction vers un formalisme technique.

4. Conclusion

La première proposition du formalisme est aujourd'hui finalisée. Il permet d'exprimer son besoin pour ensuite l'interpréter correctement vers des représentations visuelles, textuelles ou techniques. Il est désormais important de le tester dans des projets industriels afin de vérifier son potentiel d'utilisabilité. Il est également important de faire valider le méta-modèle en le comparant à des formalismes existants de spécification du besoin sous forme d'objectif mais aussi à une description textuelle ou à un modèle UML. Pour celà, nous pensons utiliser les travaux de S. Patig (Patig, 2008b, Patig, 2008a) pour réaliser notre étude. Une première version de l'outil de modélisation a été réalisé en mode Web, c'est-à-dire directement utilisable dans le navigateur. Cette possibilité permet de faciliter l'expression des besoins par des experts métier en simplifiant son accès. Le temps nécessaire à sa réalisation et à sa maintenance est très important en raison de l'immaturité des technologies utilisées dans le contexte de la création d'un modeleur. De plus, la compréhension des modèles avec cette première version est très difficile en raison de l'abscence de fonctionnalités indispensables. De plus, la notion de modèle et de diagramme y sont fusionnées ce qui rend la modélisation et la transformation plus complexes. Nous avons donc choisi de réaliser une nouvelle version de l'outil intégré dans l'environnement de développement Eclipse (Holzner, 2004). Celui-ci a été généré en suivant l'approche MDD/MDA proposé par le projet Topcased (Pontisso *et al.*, 2006).

5. Bibliographie

ANT, « The Apache ANT Project : `http ://ant.apache.org/` », 2008.

Holzner S., *Eclipse Cookbook*, O'Reilly Media, Inc., 2004.

Jouault F., Kurtev I., « Transforming Models with ATL », *Satellite Events at the MoDELS 2005 Conference*, Springer Berlin / Heidelberg, p. 128-138, 2006.

Obeo, « Acceleo Generator : `http ://www.acceleo.org` », 2008.

Patig S., « A practical guide to testing the understandability of notations », *APCCM '08 : Proceedings of the fifth on Asia-Pacific conference on conceptual modelling*, Australian Computer Society, Inc., p. 49-58, 2008a.

Patig S., « Preparing Meta-Analysis of Metamodel Understandability », *Proceedings of the First Workshop on Empirical Studies of Model-Driven Engineering*, 2008b.

Pohl K., *Process-Centered Requirements Engineering*, John Wiley & Sons, 1996.

Pontisso N., Chemouil D., « TOPCASED Combining Formal Methods with Model-Driven Engineering », *ASE '06 : Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, IEEE Computer Society, Washington, USA, p. 359-360, 2006.

Rolland C., *Conceptual Modelling in Information Systems Engineering*, Springer Berlin Heidelberg, chapter Capturing System Intentionality with Maps, 2007.

Shannon C., « A mathematical theory of communication », *Bell System Technical Journal*, 1948.

Un processus d'imitation de patrons d'ingénierie supporté par l'approche IDM

Nicolas Arnaud* – Agnès Front – Dominique Rieu - Sophie Dupuy-Chessa

Laboratoire LIG
681, rue de la Passerelle – BP 72, 38402 St Martin d'Hères
prénom.nom@imag.fr, nicolas.arnaud@comarch.com

RÉSUMÉ. *Les patrons d'ingénierie ont été introduits afin de capitaliser et de réutiliser des savoirs et des savoir-faire. Dans l'ingénierie logicielle, leur usage est aujourd'hui reconnu, à tous les niveaux (analyse, conception, ...) comme un gage de qualité. Dans cet article, nous nous intéressons à la réutilisation des patrons que nous appelons « imitation ». Nous proposons une nouvelle forme de définition de solutions pour les patrons qui s'appuie sur l'utilisation de plusieurs vues (statique, fonctionnelle et dynamique). Ces solutions se basent sur des extensions du méta-modèle d'UML pour exprimer la variabilité de la solution ainsi que l'essence de la solution sous forme de propriétés génériques définissant les bornes de l'adaptation permises lors de l'imitation. Un processus d'imitation dédié ainsi qu'un outillage basé sur l'approche IDM sont également proposés aux ingénieurs d'applications.*

MOTS-CLÉS : *patrons, variabilité, généricité, méta-modélisation, processus d'imitation.*

KEYWORDS: *patterns, variability, genericity, metamodelling, imitation process.*

1. Introduction

Un patron décrit un problème fréquemment rencontré dans un contexte ainsi que la solution consensuelle qui le résout. Notre objectif est de fiabiliser le processus de réutilisation d'un patron, que nous appelons *Imitation*, et qui permet d'extraire la solution du patron et de l'appliquer dans un système en construction. Nous utilisons l'Ingénierie Dirigée par les Modèles (IDM) pour assister et partiellement automatiser ce processus de réutilisation. Cependant, il n'y a pas de bonne imitation s'il n'y a pas de bonne spécification de la solution. Une bonne spécification est avant tout une spécification complète qui ne peut être atteinte en recourant uniquement à la modélisation statique. C'est pourquoi nous proposons trois vues complémentaires fonctionnelle, dynamique et statique. De plus, dans de nombreux cas, et particulièrement pour les patrons du GoF (Gamma *et al.*, 1995), la description du patron est clairsemée d'informations exprimant des variantes possibles que nous proposons d'intégrer dans les différentes vues.

Les principes de cette approche sont présentés en section 2. La réutilisation des spécifications est mise en œuvre au sein d'un processus d'imitation présenté en section 3. La section 4 présente l'outillage du processus en utilisant l'approche IDM.

2. Spécification d'un patron

Dans la plupart des catalogues de patrons, les solutions de patrons se résument à une vue statique. Ces structures sont souvent accompagnées de notes textuelles qui apportent quelques précisions, mais restent incomplètes pour représenter les aspects fonctionnels et dynamiques. Pour répondre à ce besoin, nous proposons d'exprimer les solutions de patrons sous la forme d'un mini-système composé de 3 vues : la vue fonctionnelle exprimant par des cas d'utilisation les différentes fonctionnalités proposées dans la solution d'un patron, la vue dynamique composée de fragments dynamiques UML2 permettant de montrer comment ces fonctionnalités sont réalisées, et enfin la vue statique composée des différents fragments statiques correspondants. De plus, afin d'apporter une meilleure qualité de réutilisation, nous proposons de distinguer dans la spécification de la solution, les aspects fixes des aspects variables, comme par exemple le fait que les fonctionnalités offertes soient essentielles ou facultatives.

La variabilité est définie comme la capacité d'un système à être changé, personnalisé et configuré en fonction d'un contexte spécifique (Van Grup, 2000). Dans notre cas un contexte spécifique correspond à une imitation. Un **point de variation** est un endroit du système où il y a une variation (Czarnecki *et al.*, 2000), c'est-à-dire où des choix devront être faits afin d'identifier les **variantes** à utiliser. Il existe plusieurs types de variabilité pour un point de variation (Bachmann *et al.*, 2001). Nous en avons retenu trois : *les options* (choix de *zéro ou plusieurs* variantes parmi plusieurs), *les alternatives* (choix de *une* variante parmi plusieurs), *les alternatives optionnelles* (choix de zéro ou une variante parmi plusieurs).

Dans notre approche, l'expression de la variabilité dans un patron est guidée par la vue fonctionnelle. Ainsi, pour l'exemple du patron «Observateur» dont l'intention est de « *Définir une dépendance entre les observateurs d'un même sujet telle que, quand le sujet change d'état, tous ces observateurs soient informés et mis à jour* », nous distinguons (figure 1, diagramme de cas d'utilisation) une fonctionnalité principale *modifier le sujet* et une fonctionnalité secondaire *gérer les observateurs*. Concernant *modifier le sujet*, deux types de modification discutés dans la rubrique *Implémentation* du patron sont possibles : d'une part la *notification implicite* correspondant à la modification de l'état du sujet avec une mise à jour des observateurs automatique, d'autre part la *notification explicite* correspondant au déclenchement de la mise à jour à la charge exclusive de celui qui initie le changement d'état.

La fonctionnalité *gérer les observateurs* traite de l'ajout et de la suppression des observateurs au sujet. Elle est considérée comme secondaire et ne distingue pas de points de variation. Dès lors que les cas d'utilisation sont spécifiés, le mini-système est complété par les aspects dynamiques (figure 1, diagrammes de séquences) et statiques (figure 1, diagrammes de classes) en prenant en compte la notion de variabilité. A chaque cas d'utilisation est associé un fragment dynamique (diagramme de séquences) et un fragment statique (diagramme de classes). Le fragment statique est directement déduit du fragment dynamique.

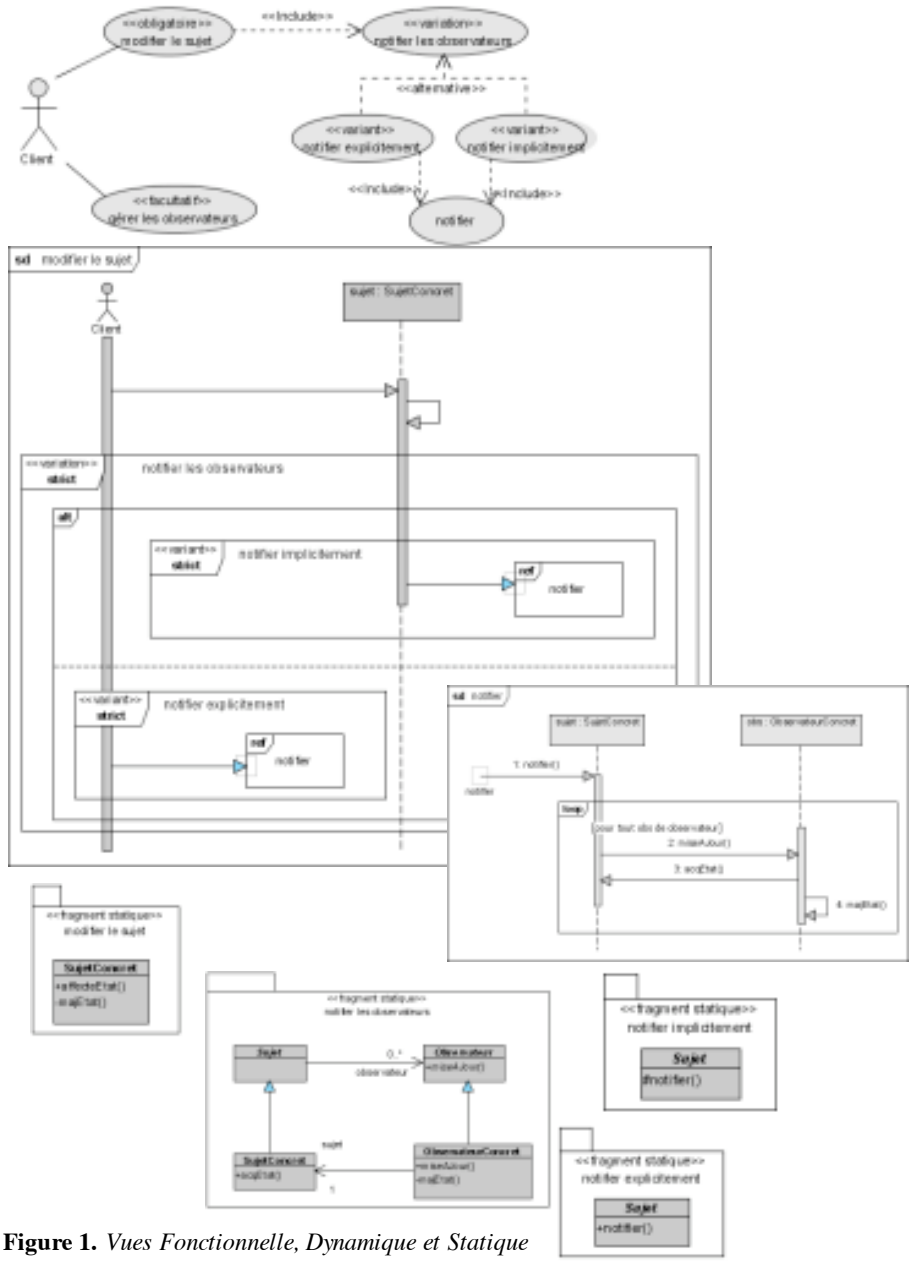


Figure 1. *Vues Fonctionnelle, Dynamique et Statique*

3. Processus d'imitation de patrons

Le processus d'imitation est destiné à assister les ingénieurs d'applications lors de la réutilisation des patrons. Ce processus traçable et automatisable guide le concepteur depuis le choix du patron jusqu'à l'intégration de l'imitation dans le système en construction. Il est composé de deux sous-processus : le processus de réduction et le processus d'application. Le processus de réduction est constitué de deux activités :

– Le *choix du patron* consiste à sélectionner un patron à imiter dans un système de patrons. La solution du patron sélectionné est appelée un modèle imitable et consiste en un mini-système à variantes composé des trois vues comme présentées dans la section 2.

– La *réduction* permet au concepteur de choisir les variantes qu'il désire imiter à partir de la vue des cas d'utilisation. Ce travail est réalisé depuis la vue fonctionnelle du modèle imitable. Il consiste à sélectionner les variantes (et non les points de variation) et les fonctionnalités secondaires que l'on désire imiter.

A partir de cette sélection, il est possible de construire automatiquement un modèle du mini-système épuré de toute information de variabilité fonctionnelle et toujours composé de trois vues. Ce modèle est appelé le modèle applicable. La figure 2 illustre la réduction fonctionnelle du patron « Observateur » via la sélection de l'alternative *notification explicite*. Les vues dynamique (partie droite, figure 2), et statique sont alors automatiquement déduites. Dans la vue dynamique, tous les fragments ne correspondant pas à un cas d'utilisation sélectionné n'ont plus de sens et sont à ignorer. Les stéréotypes sont également supprimés. En ce qui concerne la vue statique, il y a fusion des fragments correspondants aux cas d'utilisation sélectionnés.

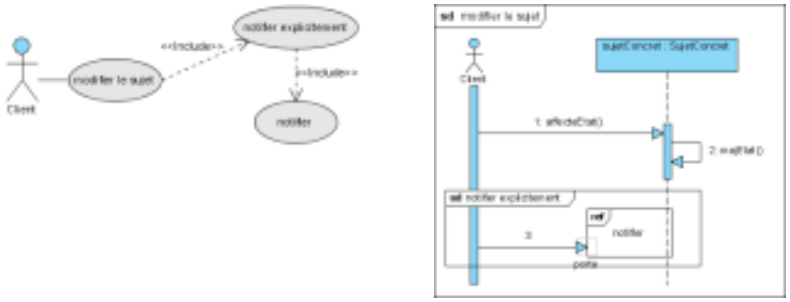


Figure 2. *Vues fonctionnelle et dynamique après réduction*

Le processus d'application est constitué de trois activités qui peuvent être exécutées de manière itérative jusqu'à obtenir un modèle imité. Les trois activités

adaptation, *résolution* et *composition* ont pour but d'adapter la spécification au contexte d'imitation.

4. Outillage de l'approche avec EMF et ATL

Le processus d'imitation présenté dans la section précédente est en grande partie automatisable. Nous proposons une architecture qui s'appuie principalement sur EMF[1] et ATL (Jouault et Kutrev 2005).

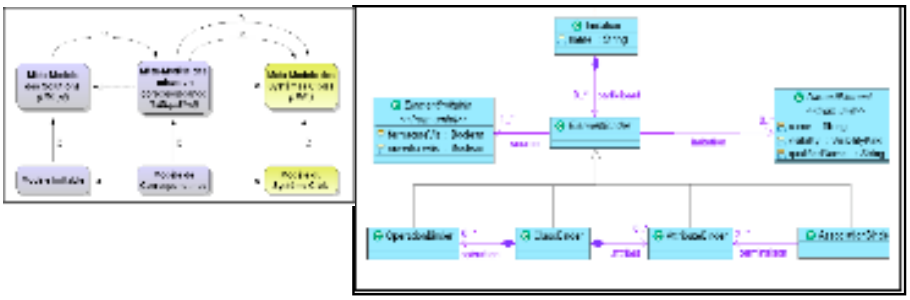


Figure 3. ***Architecture de mise en œuvre***

La partie « Solution » comporte un méta-modèle EMF extension d'UML2 (fourni par *MDT-UML2*) spécifique aux solutions des patrons. Toutes les solutions, stockées sous la forme d'un fichier XML, sont conformes (χ) à ce méta-modèle (nommé *UMLpi*, pour *UML Pattern Imitation*). La partie « Cible » n'est autre que les systèmes cibles et leur méta-modèle UML2. Les éléments issus des imitations y sont adaptés et composés.

La partie « Correspondance » fait le lien entre les éléments imitables de la partie « Solution » et leurs éléments imités de la partie « Cible ». Chaque imitation de solution donne lieu à un modèle de correspondance conforme au méta-modèle *UMLpiBind*, réalisé à l'aide d'EMF.

La figure 3 (partie droite) représente le méta-modèle *UMLpiBind* destiné à mettre en relation les éléments des systèmes cibles (*NamedElement* issu d'UML) avec les éléments imitables dont ils sont des imitations (*ElementImitable* issu d'*UMLpi*) au sein d'un *ElementBinder*. Une imitation se compose de participants (relation *participant*) qui sont en fait des *ClassBinder* ou des *AssociationBinder*. Les éléments imités d'un *ElementBinder* doivent obligatoirement respecter les contraintes induites par les propriétés génériques de ses sources (les aspects génériques ne sont pas présentés dans cet article, voir (Arnaud 2004)).

[1] Eclipse, Eclipse Modeling Framework Project, http://www.eclipse.org/emf/.

5. Conclusion

Nos propositions ont pour objectif de donner une spécification plus opérationnelle de la solution originelle offerte par un patron. Dans ce sens, le processus d'imitation garantit que l'imitation obtenue reste conforme à la spécification de la solution. Ce processus est en partie automatisé par un prototype basé sur une approche IDM et s'appuyant principalement sur les outils EMF et ATL. Ce prototype adopte une architecture tri-partie permettant de découpler les imitations des solutions dont elles sont issues par l'intermédiaire d'un modèle de correspondance. Il nous permet de vérifier que l'approche IDM est bien adaptée au support du processus d'imitation de patrons. ATL a été utilisé avec succès pour réaliser des transformations de modèles dans le sous-processus de réduction et pour vérifier la cohérence des modèles afin de prendre en compte les règles de généricité. C'est donc un langage qui nous paraît particulièrement adapté pour notre approche. L'outil est en cours d'amélioration pour supporter l'ensemble des éléments de modélisation des solutions de patrons (fragments dynamiques en particulier). Il est pour l'instant basé sur un éditeur arborescent généré automatiquement par EMF, mais le développement d'un éditeur graphique est nécessaire pour en faciliter l'utilisation. L'outil GMF[2] est une piste pour le développement de cet éditeur.

Bibliographie

Arnaud N., Front A., Rieu D., *Expression et usage de la variabilité dans les patrons de conception*. Revue ISI-RSTI série ISI, "Conception des systèmes d'information : patrons et spécifications formelles", Vol. 12, numéro 4/2007

Bachmann F., Bass L., *Managing variability in software architecture*, ACM SIGSOFT Software Engineering Notes, Volume 26, n°3, Mai 2001

Czarnecki K., Eisenecker U. W., *Generative Programming – Methods, Tools and Applications*, Addison-Wesley, 2000.

Eden A. H., *Precise Specification of Design Patterns and Tool Support in Their Application*. Thèse de doctorat, Tel Aviv University, 2000.

Jouault, F, and Kurtev, I,Transforming Models with ATL, In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica, 2005.

Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns : Element of Reusable Object-Oriented Software*, Addison-Wesley professional computing series, 1995.

OMG-UML, UML Infrastructure Specification, version V2.1.2, www.omg.org

Pagel B., Winter M., *Towards Pattern-Based Tools*. EuroPLoP'96, 1996.

Van Grup J., Variability in Software Systems, the key to software reuse, Licentiate Thesis, University of Groningem, Sweden, 2000.

[2] http://www.eclipse.org/modeling/gmf

Correspondances et transformations actives dédiées aux IHM

Olivier Beaudoux — Arnaud Blouin — Slimane Hammoudi

GRI
Groupe ESEO
Angers, France
{olivier.beaudoux, arnaud.blouin, slimane.hammoudi}@eseo.fr

RÉSUMÉ. *La récente évolution vers le Web 2.0 a mis en évidence l'importance de l'interaction entre les utilisateurs et les applications Web. Cependant, cette évolution n'est pas basée sur la transformation de données sources en vues cibles (graphiques), malgré l'intérêt déjà démontré d'utiliser les transformations dans un tel contexte. Nous proposons d'utiliser le concept de transformation active en tant qu'objet central des systèmes interactifs. Une transformation active transforme des données sources en une vue cible mise à jour dès lors que les données sources changent. Ce papier définit les principes fondamentaux des transformations actives dans une approche IDM. Il explique comment notre environnement AcT permet la génération automatique de transformations actives à partir de correspondances entre méta-modèles source et cible, et comment ces transformations peuvent être exécutées sur une plate-forme.*

ABSTRACT. *The recent evolution to the Web 2.0 underlines that interaction between users and Web-pages has become a non-negligible aspect of the development of Web applications. However, this evolution is currently not based on transforming the source data into target views, despite the high interest and expressiveness of transformations. This paper proposes the use of active transformations as the first class object within interactive systems, in order to reap the benefits from the transformation concept. An active transformation transforms source data into a target view and updates the target view as soon as the source data changes. The paper gives the foundation of active transformations based on MDE (Model-Driven Engineering). It explains how the presented AcT framework allows the automatic generation of active transformations from a graphical mapping, that can be then executed on an implementation platform.*

MOTS-CLÉS : *Correspondance, Transformation active, Interfaces Utilisateur*

KEYWORDS: *Mapping, Active transformation, Graphical User Interface*

1. Introduction

Le Web a récemment évolué d'un usage centré sur la navigation à un usage autorisant l'édition de documents. Par exemple, la rédaction de courriels est possible avec GMail, tout comme l'édition collaborative de pages HTML peut être réalisée *via* des serveurs Wiki. Le Web 2.0 a introduit l'idée de « Rich Internet Application » (RIA) pour exprimer l'évolution des interfaces clientes du Web vers des interfaces aussi riches que celles des applications de bureau traditionnelles. Cette évolution a démarré par l'utilisation de langages standard du Web (XHTML, CSS et JavaScript) complétés par la technologie Ajax (Garrett, 2005). Cependant, ces technologies ont été introduites avant le Web 2.0 et, par conséquent, leur usage n'est pas correctement adapté à la construction de RIA. De nouveaux environnements RIA sont apparus, tels que Flex (Tapper *et al.*, 2006) et WPF (Sells *et al.*, 2005), afin de permettre le développement unifié d'applications RIA pour le Web et d'applications de bureau traditionnelles.

Cependant, programmer des Interfaces Homme-Machine (IHM) reste indéniablement une tâche lourde et consommatrice de temps. Les travaux récents autour de l'IDM montrent que la génération automatique d'IHM à partir d'un modèle suffisamment explicite peut être utilisée. GMF est un exemple illustrant les bénéfices de la fusion d'outils IDM (EMF) et d'outils IHM (GEF) (Moore *et al.*, 2004; Gronback, 2009). GMF permet de générer des éditeurs de diagrammes complets à partir de seulement deux méta-modèles et d'une relation de correspondance entre ces méta-modèles, réduisant ainsi drastiquement l'effort de programmation.

Le but principal de nos travaux est d'appliquer un tel principe d'usine logicielle au contexte particulier des IHM. La première étape, présentée dans cet article, est résumée par la figure 1 : elle consiste à maintenir un lien durable entre un modèle de données sources en un modèle de vues cibles grâce à une transformation active. Un tel lien permet la mise à jour incrémentale des vues cibles dès lors que les données sources sont modifiées. Nous introduisons le terme de transformation active de manière à refléter l'idée des transformations incrémentales (Villard *et al.*, 2002), et de l'implémentation active de correspondances (Akehurst, 2000) (voir section 2). Parce qu'il est généralement assez lourd de programmer des transformations actives, nous proposons l'utilisation d'un *langage de correspondance*, à la fois graphique et textuel ; les transformations actives sont alors générées automatiquement à partir de leurs correspondances. D'autres travaux de recherche ont été réalisés autour de l'idée de transformation active, mais n'ont pas encore été appliqués au contexte des IHM (voir la section suivante) ; nos travaux visent à combler ce manque. La seconde étape de nos travaux, non présentée dans cet article, concerne le modèle d'interaction qu'il est nécessaire de considérer dès lors que les IHM sont spécifiées, de manière indépendante de la plate-forme, par des correspondances.

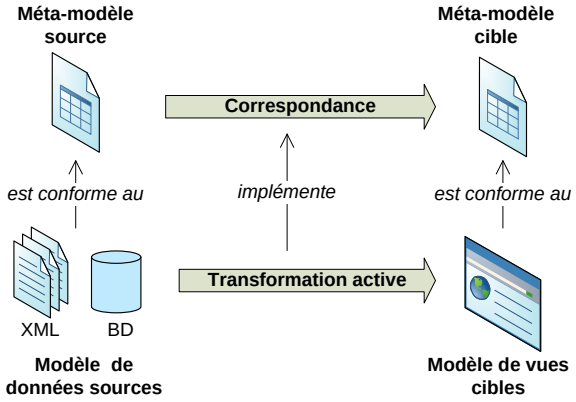


Figure 1 – Correspondance et transformation active

Cet article présente l'environnement *AcT* qui se compose du *langage AcT*, de l'éditeur graphique de correspondances *AcTeditor*, ainsi que d'une implémentation *AcT.NET* permettant d'exécuter les transformations actives sur la plate-forme .NET. Cet environnement est disponible sous licence LGPL à l'adresse *http://gri.eseo.fr/software/act*.

L'article est organisé comme suit. La section 2 présente les travaux de recherche connexes aux processus de transformation et de correspondance entre méta-modèles. La section 3 fournit les principes clés des transformations actives pour le contexte spécifique des IHM. La section 4 explique comment une transformation active est spécifiée par une correspondance *via* l'éditeur graphique *AcTeditor*. La section 5 donne le principe de la génération des transformations actives à partir des correspondances, ainsi qu'à leur exécution sur la plate-forme .NET. La section 6 conclut en synthétisant notre contribution et en donnant les perspectives de l'environnement AcT.

2. Travaux connexes

La séparation des données du domaine de leurs représentations est utilisée depuis longtemps en informatique. Dans le contexte des systèmes interactifs, le motif de conception Modèle-Vue-Contrôleur (MVC) précise explicitement une telle séparation : le modèle définit les données du domaine, la vue et ses contrôleurs présentent le modèle aux utilisateurs (Krasner *et al.*, 1988). MVC permet la synchronisation entre les modèles et leurs vues : dès lors que le modèle change, les vues associées rafraîchissent leur état. Cette synchronisation est implémentée par une liaison *vue-vers-modèle* complétée par un mécanisme de notification de changement d'état *modèle-vers-vues*. La notion de « binding », permettant d'établir le lien des vues vers les données abstraites, est présente dans les plate-formes RIA actuelles (Tapper *et*

al., 2006; Weaver, 2007; Sells *et al.*, 2005), ainsi que dans JFace[1]. Toute implémentation de MVC, basée sur le concept de « binding » ou non, implique que les vues restent dépendantes de leur modèle respectif : les liens sont définis dans les vues. De plus, ces liens dépendent de la plate-forme, contrairement aux correspondances.

L'externalisation du lien entre modèles et vues dans une transformation supprime cette dépendance. Pour cela, des langages de transformation sont utilisés, dont le plus connu est probablement XSLT (Fitzgerald, 2003). Il est largement utilisé pour transformer des documents XML en documents visualisables, tels que des documents (X)HTML ou SVG. Parce qu'XSLT est un processus « batch », il ne maintient pas le lien de synchronisation entre documents sources et documents cibles, contrairement à MVC. Cependant, des versions incrémentales de processus de transformations existent (Beaudoux, 2005b; Villard *et al.*, 2002), mais le codage de transformations incrémentales demeure plus complexe que la création de transformations « batch ».

L'approche IDM peut permettre de s'affranchir de la construction des transformations actives en se focalisant sur le niveau *méta-modèle*. De nombreux langages de transformation de modèles ont été créés (Czarnecki *et al.*, 2006; Blouin *et al.*, 2008b) ; cependant, seulement deux d'entre-eux permettent la spécification de transformations actives. (Akehurst, 2000) propose une *implémentation active* des correspondances entre deux diagrammes de classes. Le système VIATRA permet la construction de transformations dynamiques (« live transformations ») entre des modèles représentés par des graphes (Varró *et al.*, 2002). Ces deux approches illustrent le fait qu'une correspondance devrait être considérée comme la spécification d'une transformation active, dynamique ou incrémentale.

A notre connaissance, les langages de correspondance et leurs implémentations actives n'ont pas encore été spécifiquement appliqués dans le contexte des systèmes interactifs. La seule exception est GMF, mais elle reste spécialisée dans la génération d'éditeurs de diagrammes.

3. Principes fondamentaux

3.1. *Pourquoi les transformations doivent être actives ?*

Parce qu'une correspondance permet d'établir un lien durable entre la source et la cible, son implémentation doit maintenir ce lien à tout moment (Akehurst, 2000; OMG, 2005). Malgré cela, la plupart des langages de transformation ne maintiennent pas un tel lien. Cela résulte probablement de la complexité de l'implémentation des transformations actives et du côté facultatif d'une implémentation active dans le contexte des systèmes d'information. Cependant, Cela s'avère faux dès lors que la cible est une vue (graphique) de la source : dans ce cas, les données sources et les vues cibles doivent rester synchronisées au sens défini par MVC.

1. *http ://wiki.eclipse.org/index.php/JFace_Data_Binding*

Principe 1 – Les transformations pour les systèmes interactifs *doivent* synchroniser les données sources et leurs vues cibles, donc être *actives*.

3.2. *Méta-modèle des données sources*

Les données sources peuvent être enregistrées par différentes *plate-formes de données* : documents texte, documents XML ou bases de données. Puisqu'une correspondance doit rester indépendante de la plate-forme, les données sources doivent être spécifiées par un *méta-modèle*, exprimé par un diagramme de type diagramme de classes. Ce méta-modèle doit pouvoir opérer sur différentes plates-formes de données, d'une manière largement abordée dans la littérature (Jouault *et al.*, 2006; Muller *et al.*, 2006; Domínguez *et al.*, 2007).

Principe 2 – Les méta-modèles des données sources *doivent* être spécifiés par des diagrammes de classes simples. Ils *doivent* être implémentés en se basant sur des passages vers des bases de données, des documents XML ou des documents textes.

3.3. *Méta-modèle des vues cibles*

Les vues cibles présentent les données sources aux utilisateurs. Tout comme les données sources, les vues cibles doivent être spécifiées par un méta-modèle de manière à être indépendantes de la *plate-forme graphique*.

De nombreuse boîtes à outils IHM peuvent servir de *plate-forme graphique*. Cependant, la plupart d'entre-elles implémentent le motif de conception MVC d'une façon particulière en utilisant des stratégies différentes pour l'affichage graphique et la gestion des événements. Cette diversité accentue la difficulté du passage à la plate-forme graphique. Il est préférable de ne retenir que celles basées sur le concept de *graphe de scène* introduit par le domaine de la 3D dans le but de pouvoir spécifier des scènes 3D. Les boîtes à outils IHM modernes sont basées sur un tel concept de graphe de scène, telles que SVG, Draw2d (Lee, 2003) et WPF (Sells *et al.*, 2005). Puisque les graphes de scène décrivent les objets graphiques sous forme arborescente, le méta-modèle d'un graphe de scène peut s'exprimer aisément et de manière unifiée par un diagramme de classes.

Les transformations actives n'adressent pas le problème général du lien entre modèles sources et cibles, mais se focalisent plus spécifiquement sur le lien entre *données* sources et *vues* cibles. Dans le domaine de l'IHM, il est clairement admis que les utilisateurs doivent percevoir le modèle des données sources de manière à pouvoir construire leur propre modèle mental du modèle de l'application (Collins, 1995). L'application est alors considérée comme bien conçue si le modèle de l'application reste proche des modèles mentaux des utilisateurs. Ceci implique que le modèle des vues ne doit pas exagérément différer, en terme *structurelle*, du modèle des données sources, au risque de rendre ces vues peu ergonomiques. Par contre, cela ne signifie

pas que les correspondances soient nécessairement triviales : ce n'est en particulier pas le cas pour les IHM du domaine de la visualisation d'information.

Principe 3 – Les méta-modèles des vues cibles *doivent* être spécifiés par des diagrammes de classes simples, indépendamment de la plate-forme graphique, et permettant à l'utilisateur de percevoir le méta-modèle des données sources. Ils *devraient* être implémentés sur la base d'un modèle à graphes de scène.

3.4. *Correspondance et transformation active*

La figure 1 synthétise la différence essentielle entre correspondance et transformation. Une transformation opère directement au niveau modèle. Ceci offre l'avantage de la simplicité ; par exemple, construire un programme XSLT transformant un document XML en un document HTML reste une tâche simple si la grammaire du document XML n'est pas trop complexe. Cette approche a cependant deux inconvénients : premièrement, le programmeur doit lui-même s'assurer que la transformation fonctionne pour tous les documents XML sources conformes à la grammaire source, et qu'elle génère des documents HTML cibles conformes à la grammaire HTML ; deuxièmement, le programme XSLT reste dépendant de la plate-forme des données (XML) et de la plate-forme graphique (HTML), le rendant inutilisable dans d'autres contextes (*e.g.* avec une base de données relationnelle).

L'utilisation d'une correspondance plutôt qu'une transformation permet de travailler au niveau méta-modèle, supprimant ainsi les deux inconvénients précédents.

Puisqu'il est possible de représenter graphiquement les méta-modèles sources et cibles, les correspondances entre ces méta-modèles devraient également être représentées graphiquement. Les détails des correspondances devraient par contre être spécifiés textuellement.

Principe 4 – Une transformation active implémente une correspondance pour les plates-formes des données et graphique souhaitées. Elle *doit* être entièrement générée à partir de sa correspondance. Sa structure principale devrait être précisée graphiquement au dessus des diagrammes des méta-modèles sources et cibles.

4. Méta-modèle d'une transformation active

4.1. *Méta-modèle de la source et de la cible*

Le méta-modèle de données AcT possède certaines spécificités qui le rende bien adapté à notre problématique de correspondance (*e.g.* une classe *est un* ensemble d'instances). D'autres méta-modèles (*e.g.* Ecore (Steinberg *et al.*, 2008)) devraient par contre rester compatibles avec celui de AcT, voire être utilisés à la place de AcT.

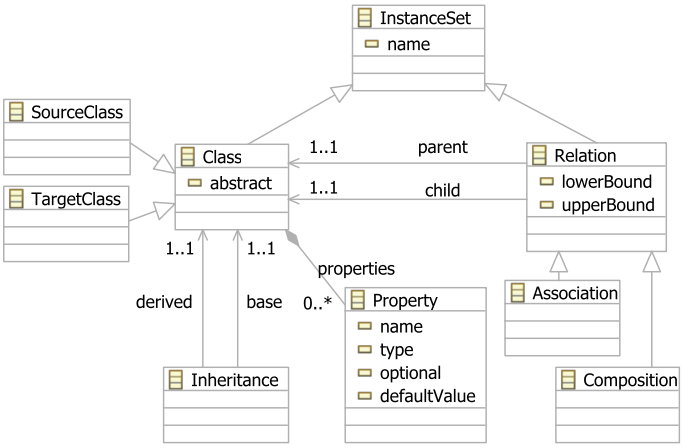


Figure 2 – Méta-modèle source et cible

La figure 2 donne le méta-modèle AcT pour les données sources et les vues cibles. Un méta-modèle source ou cible Σ est un ensemble de classes C, *i.e.* $\Sigma = \{C\}$. Une classe C (méta-classe *Class*) est considérée du point de vue ensembliste : c'est l'ensemble de toutes ses instances (méta-classe *InstanceSet*), noté $C = \{i\}$, dans lequel tout i partage la définition de *propriétés* et de *relations*. Une propriété est une donnée élémentaire représentée par un tuple (*nom, type, optionnel, valeurParDefaut*), où le type est l'un des types simples de *XML Schema* (van der Vlist, 2002). Les instances peuvent être en relation avec d'autres instances, au travers d'associations (méta-classe *Association*) ou de compositions (méta-classe *Composition*). Considérons la relation $r(C_P, C_F)$ entre une classe parente C_P et une classe fille C_F. Pour chaque $i \in C_P$, la relation r définit le sous-ensemble de C_F qui contient les instances en relation avec i, *i.e.* $r(i \in C_P, C_F) = \{j \,|\, j \subset C_F \wedge i \,\text{en relation avec}\, j\}$. Tout comme pour les classes, les *relations* sont des *ensembles d'instances*. Par conséquent, les correspondances entre classes ou entre relations sont traitées de manière unifiée : il s'agit de correspondances entre ensembles d'instances. Finalement, une classe dérivée C_D peut hériter d'une classe de base C_B (l'héritage multiple n'est pas autorisé). La classe dérivée est un sous-ensemble de la classe de base, *i.e.* $C_D \subset C_B$. Une classe de base peut être abstraite, ce qui signifie que $C_B = \bigcup_{k=1..n} C_{D,k}$ où $C_{D,k=1..n}$ sont les classes dérivées de C_B : chaque instance de C_B est nécessairement élément de l'une des classes dérivée $C_{D,k}$.

La distinction des classes sources (méta-classe *SourceClass*) et classes cibles (méta-classe *TargetClass*) est faite de manière à pouvoir définir différentes contraintes. Par exemple, une classe source ne peut pas hériter ni être en relation avec une classe

cible. De plus, la vue cible d'une correspondance est représentée par la classe cible racine qui représente la vue dans son ensemble ; cette classe racine est donc un singleton. Notons que les instances sources et cibles doivent avoir la capacité d'être enregistrées et chargées (dans et depuis un document XML par exemple), ce qui doit être géré au niveau de la plate-forme de données. Le fait que la cible puisse aussi être sauvegardée permet aux utilisateurs de modifier les vues cibles sans pour autant altérer les données sources (*e.g.* suppression d'un lien d'association dans la vue d'un diagramme UML, sans suppression dans le modèle source). Cette capacité implique que le chargement des données sources et des vues cibles soient suivies de leur mise en correspondance, ce qui n'est pas actuellement implémenté dans AcT (la vue est systématiquement construite à partir du modèle).

4.2. *Navigation*

La plate-forme AcT définit son propre langage de navigation permettant de naviguer dans les modèles sources et cibles. Ce langage utilise la notation pointée d'OCL (OMG, 2003), et est également inspiré des chemins et pas de localisation d'XPath (W3C, 2007). Une correspondance AcT utilise la navigation pour spécifier quelles instances et propriétés du modèle source sont en relation avec les instances et propriétés du modèle cible.

Un chemin C est la concaténation de $N \geq 1$ pas $P_{n=1..N}$. Chaque pas P_n s'applique à partir d'une instance donnée i, et désigne soit une propriété de i, soit une relation de i. Un pas respecte les règles de typage suivantes :

– une *valeur* : si $P_n = i.p$ où p est une propriété de i ;

– une *instance* : si $P_n = i.r$ où r est une relation avec une cardinalité égale à 1 ou 0..1 ;

– un *ensemble d'instances* : si $P_n = i.r$ où r est une relation avec une cardinalité égale à $a..b$ avec $a \geq 0$ et $b > 1$.

La concaténation de deux pas P_n et P_m, notée $P_n.P_m$, applique le résultat de P_n à P_m en retournant différents types d'éléments comme suit :

– une *valeur* : si $P_n.P_m = i.r_n.p_m$ où r_n est une relation avec une cardinalité 1 ou 0..1, et où p_m est une propriété de l'instance retournée par $P_n = i.r_n$;

– une *instance* : si $P_n.P_m = i.r_n.r_m$ où r_n et r_m sont deux relations ayant des cardinalités égales à 1 ou 0..1 ;

– un *ensemble d'instances* : si $P_n.P_m = i.r_n.r_m$ où r_n et r_m sont deux relations ayant des cardinalités égales à $a..b$ avec $a \geq 0$ et $b > 1$. Si r_n et r_m vérifient $a > 1$, la concaténation $P_n.P_m$ retourne un ensemble d'ensembles d'instances qui est *aplati* pour devenir un ensemble d'instances.

Tous les autres cas de concaténation sont interdits. Une instance peut être sélectionnée dans un ensemble d'instances en utilisant l'accesseur [], ce qui est utile pour les rela-

tions ordonnées. Par exemple, $i.r[1]$ retourne la première instance en relation r avec i. Il est également possible de se déplacer d'un pas en arrière avec la fonction $Parent()$.

4.3. *Méta-modèle des correspondances*

Une correspondance établit une relation durable entre un méta-modèle source et un méta-modèle cible. Puisque sources et cibles sont composées d'ensembles d'instances (*i.e.* de classes et de relations) et de propriétés, une correspondance peut mettre en relation n'importe quel de ces éléments de modélisation.

Une correspondance entre un ensemble d'instances sources S et un ensemble d'instances cibles C définit, pour chaque instance source $s \in S$, l'instance cible associée $c \in C$ représentant la vue de s. Si S est ordonné, la correspondance précise la relation entre les positions de s dans S et les positions de c dans C. Si S n'est pas ordonné, la correspondance met simplement s et c en relation.

Toute correspondance entre les ensembles d'instances S et C est suivie d'une correspondance entre les instances $s \in S$ et $c \in C$. Cette dernière comprend la correspondance entre leurs propriétés ainsi que la correspondance entre leurs relations. La correspondance entre les propriétés de s et c est réalisée par des opérations mathématiques appliquées à des chemins sources et cibles. Par exemple, les propriétés $prenom$ et nom d'une instance source $s \in Personne$ peuvent être mises en relation avec la propriété $text$ d'une instance cible $c \in TextBox$ en utilisant l'opérateur de concaténation : $c.text = s.prenom +'' \ '' + s.nom$. La correspondance entre les relations de s et c est réalisée de manière identique à la correspondance entre les ensembles d'instances S et C, puisque les relations sont des ensembles d'instances.

On peut se demander à présent s'il est pertinent, du point de vue ergonomie des IHM, de définir une correspondance entre un ensemble d'instances et une propriété. Par exemple, si l'on souhaite afficher le nombre d'instances d'une relation, ce nombre doit être affiché par un widget représentant, par exemple, la liste des instances de cette relation ; ainsi, l'utilisateur a un moyen de percevoir à quoi se rattache cette propriété : la mise en correspondance est donc assurée par le widget lui-même. De même, une correspondance entre une propriété et un ensemble d'instances n'a guère de sens dans notre contexte des IHM.

Enfin, plusieurs ensembles d'instances sources (*i.e.* de classes ou/et de relations) peuvent être considérées pour une correspondance : ceci est possible en considérant l'ensemble résultant de l'union de ces ensembles d'instances en tant qu'ensemble source d'une correspondance. Cette opération d'union n'est pour l'instant pas implémentée dans AcT (donc non incluse dans le méta-modèle des données). En ce qui concerne la capacité pour les correspondances à adresser plusieurs ensembles d'instances cibles, ceci est possible en utilisant soit plusieurs définitions de correspondance (solution triviale adaptée à l'idée de vues multiples), soit en autorisant cet adressage multiple pour une même correspondance en précisant la relation d'ordre (*i.e.* la relation entre les positions des instances des ensembles sources et celles des instances des

ensembles cibles), comme nous l'avons proposé dans nos récents travaux sur Malan (Blouin *et al.*, 2008a). Cette relation d'ordre est actuellement implémentée dans AcT, mais pour un unique ensemble cible (voir l'exemple donné à la section suivante).

La figure 3 donne le méta-modèle des correspondances AcT. Une correspondance est un ensemble d'opérateurs (méta-classe *Operator*). Un opérateur met en correspondance un ensemble d'instances sources S (association *Operator.source*) et un ensemble d'instances cibles C (association *Operator.target*). Deux catégories d'opérateurs sont définies : les boucles *ForEach*/*ForA* et l'opérateur *Map*. L'opérateur *ForEach* définit, pour chaque instance source $s \in S$, quelle instance cible $c \in C$ est en correspondance avec s. L'opérateur *ForA* définit quelle instance source s sélectionnée dans S correspond à une instance cible c, qui représente l'instance racine de la vue cible (voir la section 4.1). Une fois qu'un opérateur *ForA* ou *ForEach* a mis en correspondance s avec c, l'opérateur *Map* décrit comment les propriétés et les relations de s et c sont en correspondance.

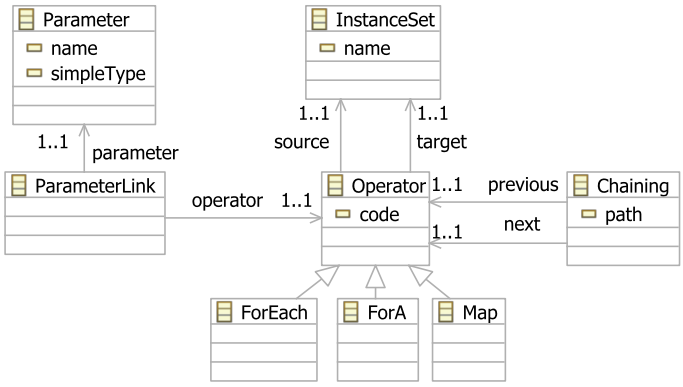


Figure 3 – Méta-modèle de correspondance

Les opérateurs sont chaînés (méta-classe *Chaining*) de la manière suivante. Une correspondance débute par au moins un opérateur *For*, suivi par un opérateur *Map*. A son tour, si l'opérateur *Map* établit une correspondance entre une relation source et une relation cible, il est chaîné à d'autres opérateurs *For* qui mettent en correspondance ces relations.

Une correspondance peut avoir plusieurs paramètres (méta-classe *Parameter*) liés aux opérateurs. Cela permet à l'utilisateur de paramétrer dynamiquement la vue cible, comme dans l'exemple du choix du critère de tri dans un widget de type liste.

4.4. *Exemple*

Cette section détaille un exemple assez complet réalisant la correspondance entre un planning scolaire et sa représentation graphique dans un agenda. La figure 4 est une capture d'écran d'une telle représentation graphique pour un planning enregistré dans un document XML.

La figure 5 (voir page suivante) présente la correspondance réalisée avec l'application *AcTeditor*. La partie gauche définit le méta-modèle source du planning. Une *Semaine* d'enseignement est décomposée en 5 *Jour*s d'enseignement, lesquels accueillent différents *Enseignement*s. Chaque enseignement concerne une *Matiere*, est assuré par un *Enseignant*, a lieu dans une *Salle*, et se déroule sur une ou deux *PlageHoraire*s consécutives. Si aucun enseignant n'est précisé pour un enseignement, le premier enseignant de la matière de l'enseignement (association *Matiere.enseignants*) est considéré. La partie droite de la figure représente le méta-modèle cible basé sur trois composants graphiques : le *CanvasAgenda* qui comprend des *LigneHeure*s ainsi que des *VignetteEvenement*s.

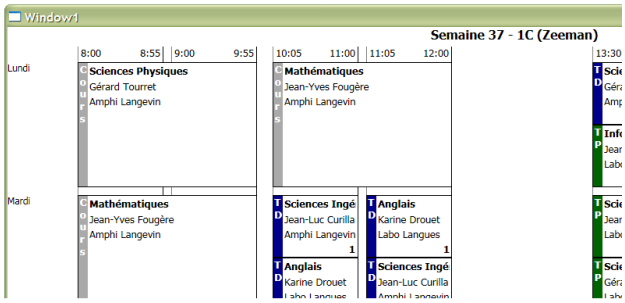


Figure 4 – Un exemple de vue cible

La structure générale de la correspondance est définie par les opérateurs *ForA*, *ForEach* et *Map*, représentés respectivement par les flèches «⟶», «⟶» et ↔. Une flèche → dénote une instance unique et une flèche ↠ dénote un ensemble d'instances. Les opérateurs sont chaînés entre-eux, chaînage représenté par la flèche ⇢. Le code associé aux opérateurs est donné dans des notes pour plus de lisibilité ; dans l'application *AcTeditor*, ce code est éditable dans une boîte de dialogue non représentée sur la figure.

La correspondance commence par l'opérateur *ForA* qui établit la correspondance entre la semaine sélectionnée (mot clé *select*) *via* le paramètre *indexSemaine* parmi les différentes semaines d'enseignement définies dans le document XML (classe source *Semaine*) et l'agenda (classe cible racine *CanvasAgenda*). L'opérateur est ensuite chaîné à l'opérateur *Map* qui établit la correspondance entre la semaine sélectionnée et

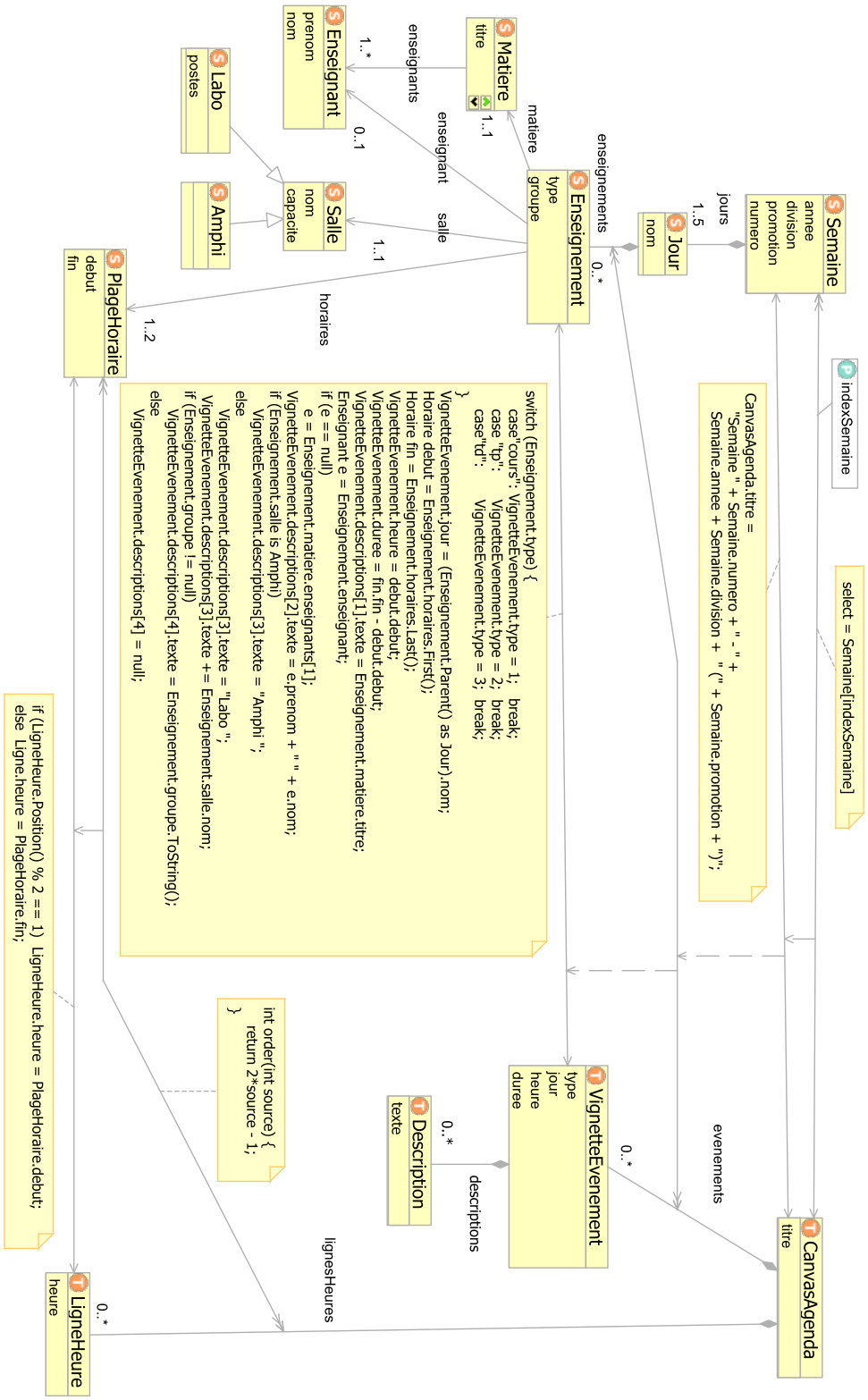


Figure 5 – Un exemple de correspondance

le canvas : le *titre* est formé à partir des différentes propriétés de la semaine. L'opérateur *Map* est chaîné à l'opérateur *ForEach* qui établit une correspondance de relations entre les enseignements de la semaine et les *evenements* du *CanvasAgenda* : il définit le fait que chaque enseignement de la semaine est associé à une vignette représentant un événement de l'agenda. Enfin, chaque enseignement (classe *Enseignement*) est mis en correspondance avec une vignette événement (classe *VignetteEvenement*) par l'opérateur *Map*. La spécification de l'enseignant, de la matière, de la salle ainsi que du groupe est effectuée en faisant correspondre les propriétés de l'enseignement à 4 *descriptions* de la vignette événement.

La correspondance comporte également l'opérateur *ForEach* entre les classes *PlageHoraire* et *LigneHoraire*. Il fait correspondre chaque instance de *PlageHoraire* avec 2 instances de *LignesHeure*, l'une correspondant au début de la plage horaire, l'autre à la fin. Cette mise en correspondance utilise la relation d'ordre entre deux ensembles d'instances *via* la fonction *int order(int s)* qui fournit, pour chaque position des instances de *PlageHoraire* la position des instances de *LigneHeure*. Au niveau de la transformation active qui implémente cette correspondance, cette relation d'ordre induit la création de 2 instances de *Ligne*Heure pour chaque nouvelle instance *PlageHoraire*, ainsi que leur insertion dans la relation *lignesHeures*. L'opérateur *Map* est finalement chaîné à ce dernier *ForEach* pour établir la correspondance entre les propriétés de *PlageHoraire* et de *LigneHeure*. Il utilise un test de parité de la position de *LigneHeure* (fonction *Position()*) de manière à faire correspondre à *LigneHeure* soit le début de la plage horaire, soit la fin.

5. Exécution d'une transformation active

La figure 6 résume le processus de génération des transformations actives pour la plate-forme d'exécution .NET. La plate-forme source est ADO.NET, ce qui autorise les données sources à être stockées dans des documents XML ou des bases de données relationnelles (Hamilton, 2008). La plate-forme graphique, basée sur le concept de graphe de scène et offrant une grand variété de widgets extensibles, est WPF (Sells *et al.*, 2005).

L'utilisateur crée en premier lieu la correspondance entre la *Source* et la *Cible* en utilisant le logiciel *AcTeditor*, comme indiqué dans la section précédente. Cette correspondance est sauvegardée dans le document XML *SourceVersCible.act*. Un menu contextuel de *AcTeditor* permet la génération de la transformation active, qui se compose de fichiers C# et *XML Schema*. Le méta-modèle source est spécifié par un schéma XML dans le fichier *Source.xsd*, lequel est utilisé par ADO.NET pour charger et sauvegarder les modèles sources dans des documents XML ou des bases de données. Le fichier C# *Source.cs* implémente toutes les classes du méta-modèle source, lesquelles permettent le chargement et la sauvegarde des instances du méta-modèle source, ainsi que la navigation à l'intérieur des ensembles d'instances sources. D'une manière similaire, le méta-modèle des vues cibles est spécifié dans le schéma XML *Cible.xsd* et implémenté dans le code C# *Cible.cs*. La transformation active, implémentée dans

le fichier *SourceVersCible.cs*, gère le lien entre la source et la cible en utilisant le motif de conception « observable - observateur » intégré à ADO.NET : la transformation est l'observateur des données sources observables. Le rendu graphique du modèle relationnel de la cible est ensuite réalisé par des vues WPF. Comme pour les transformations actives, les vues sont des observateurs des modèles cibles observables : elles observent le modèle cible de manière à synchroniser le rendu graphique. L'utilisateur doit créer, si nécessaire, ses propres contrôles *via* les fichiers *ControleUtilisateur.xaml* et *ControleUtilisateur.xaml.cs*. Ces contrôles correspondent soit à des formulaires utilisant d'autres contrôles, soit à de nouveaux contrôles spécifiques tels que ceux de notre exemple (*CanvasAgenda*, *LigneHeure* et *VignetteEvenement*). Les environnements *VisualStudio* ou *Expression Blend* peuvent être utilisés à cet effet.

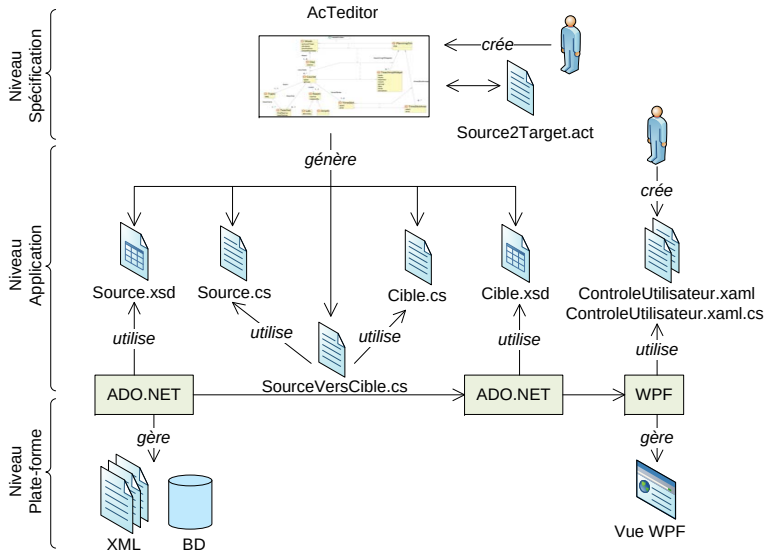


Figure 6 – Implémentation sur la plate-forme .NET

6. Conclusion

Cet article propose une utilisation, basée sur l'environnement AcT, du concept de correspondance adapté au contexte des systèmes interactifs. Une correspondance définit une relation entre un méta-modèle de données sources et un méta-modèle de vues cibles. Une correspondance est exprimée par le langage AcT : la structure générale de la correspondance est précisée graphiquement, tandis que ses détails sont spécifiés textuellement. Cette approche vise à être la plus simple possible et reste ainsi basée

sur un nombre réduit d'éléments de construction. Il ne s'agit pas d'un langage complet pour le contexte général de la correspondance, mais à pour but de répondre au contexte spécifique des IHM, où les vues cibles doivent présenter les données sources à l'utilisateur. L'environnement AcT inclut l'éditeur graphique *AcTeditor* dédié à la construction des méta-modèles sources et cibles, ainsi qu'à l'élaboration de la correspondance entre ces deux méta-modèles. Il permet la génération du code des transformations actives à partir d'une correspondance spécifiée avec le logiciel. Grâce à l'implémentation AcT.NET, ces transformations actives peuvent être exécutées sur le plate-forme .NET en tant qu'objets autonomes.

La plate-forme AcT en est aujourd'hui à sa première version et devra évoluer sous plusieurs aspects. Nous travaillons actuellement sur l'implémentation de notre modèle d'interaction adapté au concept de correspondance appliqué aux IHM. Nos précédents travaux sur eXAcT ont défini les concepts de bases à utiliser pour l'interaction, lesquels sont basés sur la notion d'interacteur de (Myers, 1989). L'interaction devra également inclure le partage des données sources ou/et des vues cibles, en temps réel ou en temps différé, sur la base du concept de point d'événement (Beaudoux, 2005a). D'autres plate-formes d'implémentation et d'exécution seront également à considérer (*e.g.* JavaFX et Flex), de manière à pouvoir générer une même application sur différentes plates-formes. Enfin, le code des correspondances, qui est actuellement un sous-ensemble du langage C#, devra évoluer vers un langage de correspondance de plus haut niveau et indépendant de la plate-forme d'exécution, sur la base de nos travaux sur Malan (Blouin *et al.*, 2008a). Il devra permettre la spécification de correspondances complexes, telles que l'on peut les rencontrer dans les applications de visualisation d'information (*e.g.* à base de zoom sémantique).

7. Bibliographie

Akehurst D. H., Model Translation : A UML-based specification technique and active implementation approach, PhD thesis, Computing Lab., University of Kent, 2000.

Beaudoux O., « Event Points : Annotating XML Documents for Remote Sharing », *Proc. of DocEng'05*, ACM Press, p. 159-161, 2005a.

Beaudoux O., « XML Active Transformation (eXAcT) : Transforming Documents within Interactive Systems », *Proc. of DocEng'05*, ACM Press, p. 146-148, 2005b.

Blouin A., Beaudoux O., Loiseau S., « Malan : A Mapping Language for the Data Manipulation », *Proc. of DocEng'08*, ACM Press, 2008a.

Blouin A., Beaudoux O., Loiseau S., « Un tour d'horizon des approches pour la manipulation des données du Web », *Document numérique - Les documents et le Web 2.0*, 2008b.

Collins D., *Designing Object-Oriented User Interfaces*, Benjamin/Cumming, 1995.

Czarnecki K., Helsen S., « Feature-based survey of model transformation approaches », *IBM Systems Journal*, vol. 45, n° 3, p. 621-645, 2006.

Domínguez E., Lloret J., Pérez B., Áurea Rodríguez, Rubio A. L., Zapata M. A., « A Survey of UML Models to XML Schemas Transformations », *proc. of WISE'07*, Springer, p. 184-195, 2007.

Fitzgerald M., *Learning XSLT*, O'Reilly, 2003.

Garrett J., Ajax : A New Approach to Web Applications, Technical report, Adaptative Path, 2005.

Gronback R. C., *Eclipse Modeling Project : A Domain-Specific Language Toolkit (to be published)*, Addison Wesley Professional, 2009.

Hamilton B., *ADO.NET 3.5 Cookbook*, O'Reilly, 2008.

Jouault F., Kurtev I., « TCS : a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering », *Proc. of GPCE'06*, ACM Press, p. 249-254, 2006.

Krasner G. E., Pope S. T., « A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80 », *Journal of OOP*p. 26-49, 1988.

Lee D., Display a UML Diagram using Draw2D, Technical report, IBM, 2003.

Moore B., Dean D., Gerber A., Wagenknecht G., Vanderheyden P., *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*, IBM, 2004.

Muller P.-A., Fleurey F., Fondement F., Hassenforder M., Schneckenburger R., Gérard S., Jézéquel J.-M., « Model-Driven Analysis and Synthesis of Concrete Syntax », *Proceedings of the MoDELS/UML 2006*, 2006.

Myers B. A., « Encapsulating interactive behaviors », *Proc. of CHI'89*, ACM Press, p. 319-324, 1989.

OMG, UML 2.0 OCL Specification, Technical report, OMG, 2003.

OMG, MOF QVT Specification, Technical report, OMG, 2005.

Sells C., Griffiths I., *Programming Windows Presentation Foundation*, O'Reilly, 2005.

Steinberg D., Budinsky F., Paternostro M., *EMF : Eclipse Modeling Framework*, Addison Wesley Professional, 2008.

Tapper J., Talbot J., Boles M., Elmore B., Labriola M., *Adobe Flex 2 : Training from the Source*, Adobe Press, 2006.

van der Vlist E., *XML Schema*, O'Reilly, 2002.

Varró D., Varró G., , Pataricza A., « Designing the Automatic Transformation of Visual Languages », *Science of Computer Programming*, vol. 44, n° 2, p. 205-227, 2002.

Villard L., Layaïda N., « An Incremental XSLT Transformation Processor for XML Document Manipulation », *Proc. of WWW'02*, ACM Press, p. 474-485, 2002.

W3C, XML Path Language (XPath) 2.0 Recommendation, Normative recommendation, W3C, 2007.

Weaver J. L., *JavaFX Script : Dynamic Java Scripting for Rich Internet/Client-side Applications*, APress, 2007.

L'Ingénierie Dirigée par les Modèles au cœur d'un Framework d'aide à la composition d'interfaces utilisateurs

Couplage des évolutions des services métiers et des IHM

Anne–Marie Déry–Pinna, Cédric Joffroy, Audrey Occello, Philippe Renevier, Michel Riveill

Laboratoire I3S (UMR CNRS / Université Nice–Sophia Antipolis)
930 Routes des Colles
B.P. 145
F-06903 Sophia Antipolis Cedex
{pinna,joffroy,occello,renevier,riveill}@polytech.unice.fr

RÉSUMÉ. *Les applications construites à partir de services posent le problème de la réutilisation et de la composition des IHMs de ces services. Dans la plupart des cas, les anciennes IHMs sont ignorées et de nouvelles sont conçues et implémentées pour la nouvelle composition de services.*

Nous proposons de déduire des compositions des services métiers les informations utiles pour guider la composition des IHMs associées afin de réutiliser ces dernières. Cet article présente un framework d'aide à la composition des IHMs qui met en œuvre un processus global de composition qui s'articule autour de plusieurs modèles (des IHMs et des services) et autour de transformations plus ou moins directes entre eux.

ABSTRACT. *Service oriented applications raise the problem of reusing and composing UI associated with each service. The more the application evolve (addition/suppression of services), the bigger this problem becomes. In such a case, old UIs are thrown away and new UIs are created to correspond to the new services composition usage.*

We propose to deduce UI composition from services composition. This article presents a framework helping to compose UIs. This framework proposes a composition process based on different models which describe UI and services and based on transformations.

MOTS-CLÉS : *composition d'IHM, règles de déduction, transformation de modèles*

KEYWORDS: *UI composition, deduction rules, model transformation*

1. Introduction

La distribution commerciale ou gratuite des produits informatiques n'a cessé d'évoluer. Originellement les solutions ad-hoc étaient faites sur mesure. Maintenant les entreprises informatiques vendent des composants ou des services "sur étagères". Ces briques de base s'intègrent pour former des solutions globales, à l'image des PGI (progiciel de gestion intégré). Solutions informatiques dédiées à la gestion informatique des entreprises (ressources humaines, ventes, distributions, approvisionnement, etc.), ces derniers reposent sur une normalisation des données échangées, ce qui permet le partage de celles-ci entre plusieurs applications. Ainsi des moteurs de *workflow* peuvent être mis en place pour propager les modifications des données entre ces applications. Cette propagation peut se faire sans même que les utilisateurs ne s'en rendent compte. Il existe donc des architectures pour mettre en œuvre la réutilisation et l'intégration de services.

Cependant, cette intégration reste au niveau des données et ne concerne pas encore les Interfaces Hommes-Machines (IHM). En effet, lors du couplage de plusieurs services entre eux :

– soit chaque interface reste indépendante et les utilisateurs n'ont pas forcément connaissance des liaisons entre les applications. Dans ce cas, la propagation des modifications des données permet de ne pas saisir plusieurs fois les mêmes modifications. Mais aucun usage "harmonieux" des applications n'est possible : les utilisateurs doivent basculer de l'une à l'autre, au risque d'oublier une étape ou de ne pas respecter l'ordre des procédures.

– soit les parties métiers de ces services sont conservées, mais une nouvelle interface est créée. Dans ce cas, l'éditeur de services doit reconcevoir l'interface voire recoder une grande partie de celle-ci. Les sources des interfaces initiales peuvent éventuellement être reprises, à condition qu'elles soient écrites avec des technologies compatibles. Par ailleurs, aucune information provenant de la composition des "services métier" n'est automatiquement utilisée pour la réalisation de l'interface. En quelque sorte, c'est comme si le développement de la nouvelle interface repartait de zéro.

D'autre part, les services ne sont pas intégrés une fois pour toutes. En effet, les besoins des utilisateurs de ces applications intégrées évoluant dans le temps, des services sont amenés à être ajoutés pour proposer de nouvelles fonctionnalités ou à être supprimés lorsqu'ils ne sont plus utiles. Les applications proposées aux utilisateurs changent donc et leurs interfaces facilitant l'usage doivent évoluer et s'adapter en conséquent. La "reconstruction" de la partie IHM des applications qui ont évolué devient alors un problème récurrent. L'enjeu est donc la réutilisation des IHM des services pour ne pas avoir à tout redévelopper et la composition de ces IHM pour en former des nouvelles.

Le plus souvent les services interagissent entre eux via un *workflow* ou un *dataflow* qui permet de propager les données d'un service à un autre. Nous supposons que ces interactions sont une forme d'implémentation des enchaînements de tâches utilisateurs au niveau des services métier et qu'elles doivent être exploitées afin d'obtenir des informations sur la façon de composer les IHM associées à ces services. Le fra-

mework raisonne sur un ensemble de modèles afin de pouvoir faciliter la composition des services et des IHM technologiquement hétérogènes.

Cet article présente en section 2 le framework et en particulier les modèles reflétant les applications ainsi que la façon de les manipuler afin d'atteindre nos objectifs de composition. La section 3 détaille dans un premier temps les méta-modèles en présence et des exemples de modèles conformes à ces méta-modèles puis les transformations de modèles mises en œuvre pour faciliter le processus de composition au niveau IHM. Nous terminons par les limites actuelles et les perspectives.

2. Modèles et utilisation des modèles dans le framework

Dans cette partie, nous présentons une vue d'ensemble de notre framework en axant essentiellement sur les modèles impliqués dans le processus de composition. Nous commençons par préciser les objectifs de nos travaux et les hypothèses de travail s'y rapportant, nous les illustrons ensuite au travers d'un scénario de composition. Nous terminons cette partie par la description de l'architecture globale du framework, dans laquelle nous présentons brièvement les modèles et transformations, qui sont détaillés dans la partie suivante.

2.1. *Objectifs*

L'objectif principal que nous nous fixons est la mise en place d'une synergie entre le monde des services métier et celui des IHM.

En effet, en recherche comme dans la pratique, il existe une bivalence entre les services métier et l'IHM. Cette décomposition est illustrée par le modèle architectural Arche (Bass *et al.*, 1992) qui offre une décomposition fonctionnelle d'un système interactif en séparant l'IHM des fonctionnalités métiers (Noyau Fonctionnel–NF). Les architectures proposées en IHM, telles que MVC (Reenskaug, 1979), PAC (Coutaz, 1987) ou encore les solutions récentes pour la plasticité comme AMF (Tarpin-Bernard *et al.*, 1999) ou Comet (Demeure *et al.*, 2006), ont pour point commun de dissocier au moins 2 niveaux : le niveau noyau fonctionnel (service métier) et le niveau présentation (ihm finale). Dans la suite de l'article nous utiliserons indifféremment les termes composants métiers et services fonctionnels pour illustrer le NF. Aussi, nous avons retenu l'hypothèse suivante : "la partie métier et la partie interface sont clairement identifiées et dissociées".

Les compositions sont étudiées indépendamment dans les deux domaines de recherche IHM et génie logiciel. Par exemple, il existe en IHM des approches pour composer et adapter un système interactif en intégrant les modèles conceptuels aux applications afin d'en re-générer des parties en cas de changement de contexte. Dans

(Sottet *et al.*, 2006), un modèle est associé à chaque niveau du cadre CAMELEON[1] (I.S.T.I. - C.N.R., 2004) pour chaque contexte d'usage et donc l'adaptation se fait par transformations successives de modèles. Mais la décomposition du cadre CAMELEON, allant des tâches aux interfaces finales, n'inclut pas la partie fonctionnelle au niveau des modèles. La composition d'IHM est de plus en plus au centre des travaux en IHM comme dans (Gabillon *et al.*, 2008) avec la mise en place de planificateurs ou avec des outils plus proches des utilisateurs comme les mash-up…

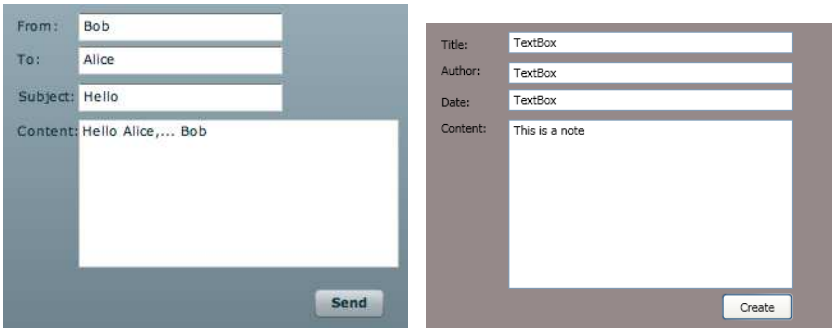
Réciproquement, les travaux en Génie Logiciel et en IDM ont, jusqu'à un passé proche, considéré les IHM comme des blocs monolithiques non réutilisables. Les travaux de composition, très avancés sur les services métier, n'ont donc pas pris en compte cette dimension IHM. Par exemple, la réalisation de compositions de services par la spécification BPEL4WS permet de définir des orchestrations (Khalaf *et al.*, 2003). Le résultat d'une orchestration est un nouveau service, ce qui permet une composition récursive des orchestrations. Cependant, les services initiaux ne sont pas modifiés même s'ils ne sont pas utilisés pour toutes leurs fonctionnalités dans la composition. Là où la présence de ces fonctionnalités non utilisées ne perturbe généralement pas l'utilisation d'un service métier, la présence de codes non exploités nuit à l'utilisabilité de l'interface. En effet, ce code superflu se traduit par la présence d'éléments inutiles qui surchargent l'IHM en terme d'exécution et d'ergonomie.

Ainsi, si des compositions existent aux niveaux IHM et services, ces compositions ne traitent qu'un seul de ces deux points de vue. Nous visons donc à faire collaborer ces deux mondes. Cet objectif principal se décompose en trois objectifs intermédiaires : (i) la réutilisation maximale des services métier et des IHM, (ii) la propagation des compositions des services métier au niveau des IHM et (iii) la propagation des compositions des IHM au niveau des services métier. Dans cet article nous présenterons uniquement les deux premiers points, les travaux sur le dernier aspect n'étant qu'amorcés à l'heure actuelle.

2.2. *Scénario de composition d'IHM*

Supposons deux services potentiellement développés dans des technologies différentes : un service de courriels et un service de notes/mémos. Chaque service est défini par un ensemble de fonctionnalités – pour le service de courriels, "envoyer et recevoir un message", pour le service de notes, "créer une note". Un message se compose d'un expéditeur, d'un destinataire, d'un sujet et d'un texte. Une note est composée d'un titre, d'un auteur, d'une date et d'un texte. La Figure 1(a) donne un exemple de présentation de courriels. La Figure 1(b) donne un exemple de présentation de notes.

1. Par ordre de niveaux décroissants d'abstraction : les tâches (buts et sous buts de l'utilisateur) et le domaine (objets manipulés) ; l'interface abstraite (expression de l'interaction indépendamment des modalités d'interaction) ; l'interface concrète (expression de l'interaction indépendamment de la plate-forme d'exécution) ; l'interface finale (l'interface opérationnelle).

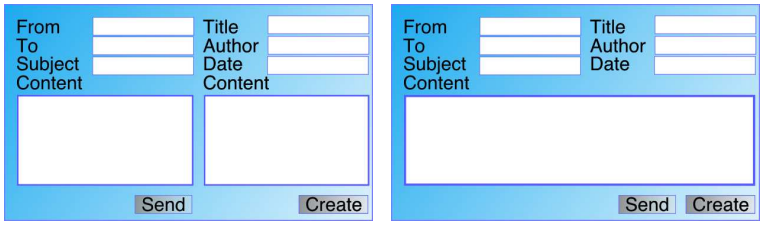


(a) IHM de Mail (b) IHM de Notes

Figure 1. *Exemple de deux IHM qui serviront de base à la composition*

L'assemblage de ces deux services au niveau fonctionnel permet de créer une fonctionnalité de plus haut niveau qui permettra à la fois de créer une note et d'envoyer un mail. Ce premier assemblage correspond à une imbrication de services. Une présentation du résultat de cet assemblage est illustrée par la Figure 2(a).

Le second assemblage correspond à une imbrication avec fusion d'entrées. En effet, on peut s'apercevoir que les deux services possèdent une entrée commune qui est "Content". Il est donc intéressant de les fusionner au niveau fonctionnel. L'IHM qui doit résulter de cette composition ne doit donc plus posséder qu'un seul champ "Content" ce qui pose donc un problème de choix de l'élément que l'on doit conserver. La Figure 2(b) illustre, par une IHM, le résultat attendu avec cet assemblage.



(a) Union des deux IHM (b) Union des deux IHM avec fusion des éléments communs

Figure 2. *Résultat possible de la composition*

Ces déductions sont faites à travers l'utilisation des modèles présents dans notre framework.

2.3. *Architecture globale du framework d'aide à la composition*

L'architecture globale, illustrée par la figure 3, a pour objectif d'intégrer des éléments existants (interfaces concrètes et services métier) afin de construire une représentation modèle de ces services fonctionnels interactifs en s'appuyant sur la décomposition service–interface.

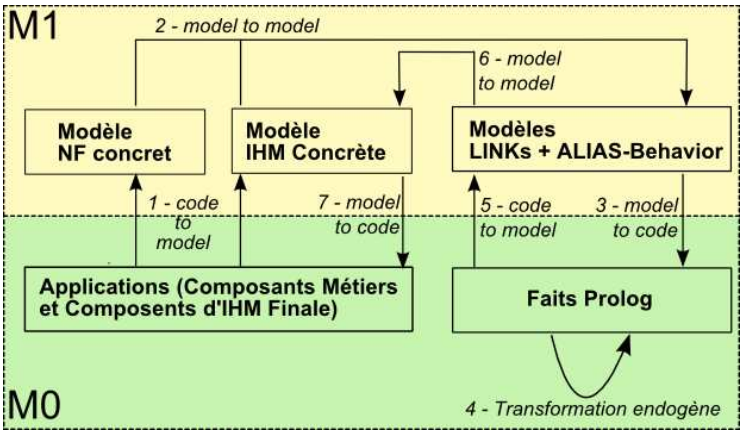


Figure 3. *Framework*

2.3.1. *Les modèles au cœur du framework*

Le cœur du framework repose sur le modèle ALIAS-Behavior. Il représente les services métier et les IHM sous forme de composant à partir des flots de données (entrées et sorties) et des événements. Le modèle LINKs permet d'exprimer les relations entre deux modèles ALIAS-Behavior.

Pour prendre en charge l'ensemble des services métier constituant le cadre applicatif, le framework repose sur :

– le modèle ALIAS-Behavior (cf. Figure 3) qui permet d'abstraire les services métier sous la forme de composants hiérarchiques en identifiant les requis (données en entrée) et les fournis (données en sortie), ainsi que les éléments de contrôle (événements correspondant au déclenchement des opérations du composant). Ce modèle permet de décrire ces services et leur composition selon le flot de données. L'expression des relations entre services métier et composants d'IHM avec le modèle LINKs (cf. Figure 3) fait ressortir les enchaînements de tâches utilisateurs au niveau des services métier.

– les services métier concrets reflétant l'application d'un point de vue « métier » (Modèle NF concret cf. Figure 3). Ces services sont orchestrés ou assemblés. Ce modèle correspond aux entrées du framework.

Pour prendre en charge la partie IHM qui correspond à l'ensemble des interfaces et éléments d'interfaces associés aux services métier, le framework repose sur :

– le modèle ALIAS-Behavior complété de ALIAS-Structure (représentation plus axée sur la structure des données de l'IHM) et ALIAS-Layout (représentation de l'organisation de l'IHM) pour affiner les IHM .

– des modèles des interfaces concrètes (Modèle IHM Concrète cf. Figure 3) manipulées par les utilisateurs finaux reflétant l'application d'un point de vue « interaction ». Ces modèles peuvent être construits à partir de descriptions abstraites des IHM. Ils sont des entrées et des sorties du framework.

En raisonnant sur ces modèles, le framework peut déduire des compositions au niveau IHM. Pour cela, le framework comprend également un moteur de déduction Prolog permettant de déduire des assemblages d'IHM à partir des enchaînements de tâches. Ces enchaînements sont exprimés implicitement dans les assemblages métiers et des liaisons explicites entre la partie fonctionnelle et la partie IHM.

2.3.2. *Processus de composition et utilisation des modèles*

Les trois grandes étapes du processus de composition sont les suivantes :

Etape 1 : Construction dans le framework d'un service fonctionnel interactif. Cette construction implique de a) créer un composant d'IHM correspondant à l'IHM concrète du service, b) créer un composant métier à partir de la partie fonctionnelle et c) créer les liaisons entre les deux composants. Les deux premières activités impliquent une transformation des modèles IHM Concrète et NF Concret vers deux modèles ALIAS-Behavior via des liaisons LINKs.

Etape 2 : Construction des assemblages des composants métier que l'on souhaite composer. Cette construction ajoute un composant composite au niveau modèle ALIAS dans le framework.

Etape 3 : Déduction à l'aide du moteur Prolog des assemblages des composants d'IHM (création d'un composite au niveau Modèles) à partir des informations cidessus, liaison de ce nouveau composant avec le composite métier via LINKs puis transformation du composant d'IHM dans une IHM concrète.

Ces trois étapes correspondent à un enchaînement de transformations, illustrées par les flèches numérotées de 1 à 7 sur la Figure 3 et étiquetées par la nature des transformations.

Les transformations no 1 permettent de récupérer dans l'existant les informations essentielles sur les composants métier et les composants d'IHM. Pour permettre de déduire une composition, il faut transformer (transformation no 2) les modèles NF Concret et IHM Concrète en modèles ALIAS-Behavior (plus ALIAS-structure et ALIAS-Layout dans le cas des IHM Concrètes). La transformation no 3 peuple la base de faits Prolog avec les informations relatives aux deux modèles ALIAS-Behavior obtenus. A l'intérieur du moteur Prolog est réalisée une transformation endogène no 4 qui consiste à déduire la composition des faits Prolog en entrée pour produire en sortie

d'autres faits Prolog représentant les nouveaux assemblages à réaliser. La transformation no 5 permet de représenter les faits Prolog reflétant la composition en ALIAS-Behavior. La transformation no 6 permet de répercuter les changements au niveau du modèle IHM Concrète. L'étape 7 permet de régénérer les composants d'IHM finaux utilisables dans l'application. Les détails d'implémentation des différentes transformations sont explicités dans la section 3.3.

Pour passer d'un modèle à un autre, les transformations s'appuient sur les méta-modèles du modèle de départ et du modèle cible de la transformation. Les modèles ALIAS-Behavior, ALIAS-Structure, ALIAS-Layout et LINKs sont conformes respectivement aux méta-modèles de même nom : ALIAS-Behavior, ALIAS-Structure, ALIAS-Layout et LINKs décrits en section 3. Ces méta-modèles sont une des contributions du framework. Les autres modèles (IHM Concrète et NF Concret) peuvent être basés sur différents méta-modèles qui ne font pas partie du framework. Par exemple, les modèles NF Concret peuvent être basés sur le méta-modèle des Web Services (W3C Working Group, 2004) fonctionnant sur des plates-formes .net ou java ou celui des composants Fractal (Objectweb Consortium, 2008), OSGi (Open Services Gateway initiative, 2003) ou SCA (IBM, 2006). De même, les modèles IHM Concrètes sont créés à partir d'interfaces abstraites dont le méta-modèle peut être celui basé sur les grammaires des langages associés à Flex (Adobe, 2006), ou encore celui basé sur la grammaire XaML (Microsoft, 2007) ou sur UsiXML (Vanderdonckt, 2007).

Ce framework permet de proposer des alternatives de compositions possibles, identifier des conflits ou des ambiguïtés au niveau de la composition d'IHM déduite et aiguiller et faciliter la réutilisation des interfaces initiales. L'architecture permettra à terme de faire des transformations vers des modèles d'IHM pour la plasticité et d'améliorer le rendu des interfaces finales. Les règles de composition du moteur Prolog sortent du cadre de ce papier. La suite de l'article se focalise sur les aspects IDM liés au framework ALIAS et présente donc les différents méta-modèles en présence ainsi que les transformations présentées ci-dessus.

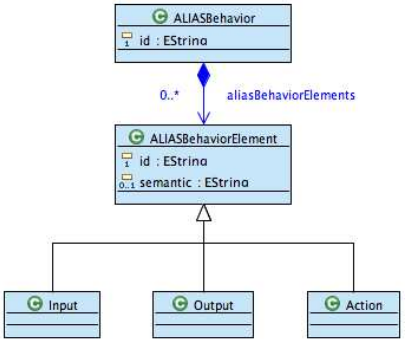
3. Méta-modèles et transformations

Dans cette section, nous détaillons les principaux méta-modèles et les transformations outillées dans la version actuelle du framework.

3.1. *Méta-modèles ALIAS* - (M_{ALIAS})

ALIAS-Behavior (Figure 4(a)) est le méta-modèle central conçu pour plus facilement exprimer l'assemblage au niveau de la présentation. Il est basé sur une représentation d'entrées/sorties (*i.e. Input* et *Output*) et d'événements (*i.e. Action*). Chaque élément peut posséder un champ sémantique afin de permettre d'étendre par la suite les opérateurs de compositions pour ajouter de la résolution sémantique. La Figure 4(b)

représente une instance du méta-modèle qui correspond à la description de l'IHM du Mail.



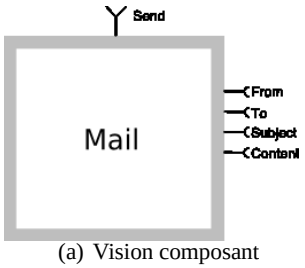
(a) Méta-modèle d'ALIAS-Behavior

```
<?xml version="1.0" encoding="ASCII"?>
<ALIASBehaviorJava:ALIASBehavior xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ALIASBehaviorJava="http://ALIASBehaviorJava"
  xsi:schemaLocation=
  "http://ALIASBehaviorJava ../model/ALIASBehaviorJava.ecore"
  id="mailUI">
  <aliasBehaviorElements
    xsi:type="ALIASBehaviorJava:Input"
    id="from"/>
  <aliasBehaviorElements
    xsi:type="ALIASBehaviorJava:Input"
    id="to"/>
  <aliasBehaviorElements
    xsi:type="ALIASBehaviorJava:Input"
    id="subject"/>
  <aliasBehaviorElements
    xsi:type="ALIASBehaviorJava:Input"
    id="content"/>
  <aliasBehaviorElements
    xsi:type="ALIASBehaviorJava:Action"
    id="send"
    semantic="trigger"/>
</ALIASBehaviorJava:ALIASBehavior>
```

(b) Instance du méta-modèle d'ALIAS-Behavior décrivant MailUI

Figure 4. *Modèle MailUI et son méta-modèle ALIAS-Behavior*

ALIAS-Behavior permet également de représenter la partie Noyau Fonctionel du fait de ses entrées, sorties et événements que l'on peut faire correspondre avec les différentes opérations proposées par le service (*Action*), les différentes entrées associées aux opérations (*Input*) et sorties (*Output*). De ces entrées, sorties et événements, il est possible de tirer une représentation sous forme de composant qui associe à chaque entrée/sortie un port de données et à chaque événement un port d'événement. Ainsi, on obtient la représentation illustrée par la Figure 5(a).



(a) Vision composant

```
element_nf(mail,
[from,to,subject,content],
[], [send], []).
```

(b) Vision Prolog

Figure 5. *Noyau Fonctionel Mail*

Pour permettre la composition, cette même représentation sous forme de composant [2] est également représentée en faits Prolog (Figure 5(b)). On retrouve donc les mêmes éléments décrits sous forme de liste de ports qui sont inclus dans un *element_nf*. Les différentes listes incluses dans cet *element_nf* sont : (i) les ports d'entrée données, (ii) les ports de sortie données, (iii) les ports d'entrée événements et (iv) les ports de sortie événements.

ALIAS-Behavior possède pour la partie présentation une autre représentation proche de la représentation composant que l'on a du Noyau Fonctionel. Cette représentation est basé sur le méta-modèle décrit par la Figure 6. Ainsi, cela nous permet de conserver une cohérence avec la partie fonctionnelle.

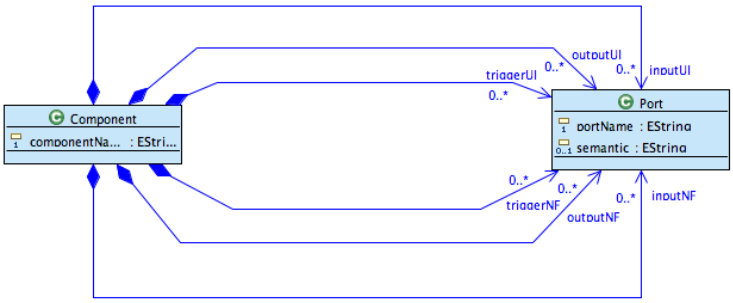


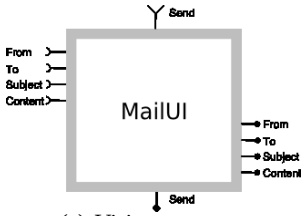
Figure 6. *Méta-modèle composant d'ALIAS-Behavior*

Le composant IHM contient plus d'éléments que le composant NF. En effet, nous avons ici une double représentation des entrée, sorties et événements due au fait que ce composant fait le lien entre le monde du l'IHM et le monde du NF. A l'heure actuelle, nous avons une correspondance directe entre les éléments c'est-à-dire que les *Inputs* (resp. *Outputs*) de la partie IHM deviennent des *Outputs* (resp. *Inputs*) pour le NF. Il en est de même pour les événements.

Ainsi, à partir du méta-modèle présenté dans la Figure 6, il est possible d'obtenir les représentations graphiques (Figure 7(a)) et Prolog (Figure 7(b)). Ces représentations correspondent à l'IHM de Mail.

Le méta-modèle ALIAS-Structure permet de préciser le type de données contenues par une entrée (ou une sortie) ou bien encore si l'élément en entrée peut être sélectionnable ou pas. Par exemple, ALIAS-Structure permet de décrire si l'on a une liste ou un ensemble de *checkbox*. Actuellement, ALIAS-Layout permet de faire du regroupement d'éléments d'ALIAS-Behavior afin de déterminer que ceux-ci appartiennent au même *Container*. A terme, l'objectif d'ALIAS-Layout est de fournir des informations précises sur le placement de chacun des éléments les uns par rapport aux autres.

2. les composants composites ne sont pas illustrés par soucis de simplification



(a) Vision composant

```
element_ui(mail_ui,
[from,to,subject,content],
[], [send], [], [],
[from,to,subject,content],
[], [send]).
```

(b) Vision Prolog

Figure 7. *IHM de Mail selon ALIAS-Behavior*

Afin de faciliter l'explication du prochain méta-modèle, nous distinguerons le méta-modèle fonctionnel (M_{NF}) et le méta-modèle présentation (M_{AUI}) qui correspondent à la vision composant de chacun des deux mondes.

3.2. *Méta-modèle liant les Méta-modèles fonctionnel et Présentation -* (M_{LINKs})

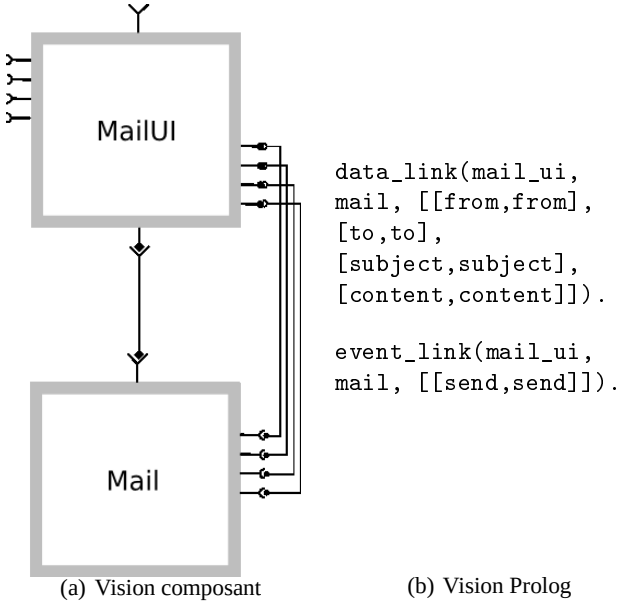
Le troisième méta-modèle sert de trait d'union entre les deux méta-modèles précédents pour décrire le flot de données qui passe d'un composant métier à un composant d'IHM et inversement. Il s'agit de faire correspondre, pour un composant d'IHM et son composant métier associé : 1) les ports de sortie de données de l'IHM avec les ports d'entrée de données du métier, 2) les ports d'entrée de données de l'IHM avec les ports de sortie de données du métier et 3) les ports de sortie de événements de l'IHM avec les ports d'entrée de événements du métier. Ce méta-modèle (M_{LINKs}) n'est donc pas un méta-modèle classique, c'est un méta-modèle de correspondance qui englobe le méta-modèle (M_{NF}) et le méta-modèle (M_{AUI}). L'assemblage de deux composants (métier et IHM), à travers la composition des trois méta-modèles (M_{NF}), (M_{AUI}) et (M_{LINKs}), forme alors le service fonctionnel interactif final. La Figure 8 illustre la liaison entre un composant métier et un composant d'IHM.

3.3. *Description des différentes transformations*

Maintenant que nous avons présenté les différents méta-modèles avec leurs modèles associés, il est intéressant de voir les différentes transformations que nous avons mises en œuvre pour valider le framework (cf. Figure 3)

3.3.1. *Transformation d'IHM existante vers ALIAS-Behavior*

Cette transformation illustre les transformations 1 et 2 de la Figure 3. Dans le processus de composition, nous supposons la transformation 1 fournie. Pour valider l'approche, nous avons pour le moment procédé à une transformation code-to-model qui permet de passer directement de l'application à ALIAS.



(a) Vision composant (b) Vision Prolog

Figure 8. *Lien entre le NF et l'IHM pour un service fonctionnel interactif de Mail*

La transformation d'abstraction d'une interface existante permet d'obtenir les instances de (M_{AUI}). L'abstraction permet de pouvoir homogénéiser des IHM existantes qui auraient été décrites dans divers langages afin de pouvoir les composer. L'abstraction permet aussi de pouvoir conserver le maximum d'information en vue de concrétiser le résultat de la composition. Actuellement, nous pouvons abstraire une IHM à partir d'une description de type XML comme Mxml ou Xaml. Il s'agit donc d'une transformation de type code-to-model (reverse engineering) basé sur le mode visiteur selon la classification de (Czarnecki *et al.,* 2003). Pour réaliser ce type de transformation, nous avons donc mis en place un modèle minimaliste qui permet de représenter les différents types de widgets existants ainsi que certaines propriétés qui peuvent leur être associés dans ces différents langages. La transformation est effectuée en deux temps : 1) une transformation de XML vers le modèle minimaliste, 2) une transformation qui applique un pattern Visiteur pour transformer les objets du modèle minimaliste en des objets des modèles ALIAS-Behavior, ALIAS-Structure et ALIAS-Layout.

3.3.2. *Transformation d'ALIAS-Behavior vers une IHM Concrète*

Cette transformation illustre les transformations 6 et 7 de la Figure 3. Dans le processus de composition, nous supposons la transformation 7 fournie. Pour valider l'approche, nous avons pour le moment procédé à une transformation model-to-code qui permet de passer directement de ALIAS à l'application .

La transformation de concrétisation d'une description ALIAS permet d'obtenir une interface utilisable. Pour le moment la concrétisation n'exploite que ALIAS-Behavior ; une structure et un layout par défaut étant appliqués. Il s'agit d'une transformation de type model-to-code toujours suivant la classification de (Czarnecki *et al.*, 2003).

Pour cette transformation, différents mécanismes ont été mis en place. La première solution se base sur un pattern Visiteur qui nous permet de générer une IHM en Java Swing. La seconde solution s'appuie sur l'utilisation des *Conditional Transformations* (Kniesel *et al.*, 2003) et qui permet de générer de l'HTML (Bihler *et al.*, 2008). Enfin la dernière solution s'appuie sur l'outils Acceleo[3]. Cette dernière transformation permet d'obtenir un fichier MXML qui correspond à une interface Flex. Cette transformation s'appuie sur une description du méta-modèle en ECORE (Figure 4(a)). A partir de ce méta-modèle il est possible de décrire un *template* qui définit la transformation à effectuer. La Figure 9 correspond au *template* de génération de fichier MXML. Cette transformation prend en entrée une instance du méta-modèle (un fichier XMI comme par illustré dans la Figure 4(b)) et produit un fichier MXML (qui une fois compilé permettra d'obtenir l'interface de la Figure 1(a)).

```
<%
metamodel /ALIASBehavior/model/ALIASBehaviorJava.ecore
%>
<%script type="ALIASBehaviorJava.ALIASBehavior" name="flex1" file="<%id%>.txt"%>
<?xml version="1.0"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Box direction="vertical">
 <%for (aliasBehaviorElements) {%>
  <%for (eClass().replaceAll("\"," " ).toString().split(" ")) {%>
   <%if toString().equalsIgnoreCase("Input") {%>
  <mx:Label text="
   <%}%>
   ...
  <%}%>
   <%id%>"
   id="<%id%>"/>
  <%for (eClass().replaceAll("\"," " ).toString().split(" ")) {%>
   <%if toString().equalsIgnoreCase("Input") {%>
  <mx:TextInput/>
   <%}%>
   ...
  <%}%>
 <%}%>
 </mx:Box>
</mx:Application>
```

Figure 9. *Extrait du* template *pour la génération de fichier* MXML

3. http://www.acceleo.org/

3.3.3. *Transformation d'ALIAS-Behavior objet vers ALIAS-Behavior composant*

Afin de traiter les transformation 3 et 5, nous faisons une transformation model-to-model qui nous permet de passer d'une représentation objet (obtenue lors de la transformation d'abstraction) à une représentation composant qui est la base du moteur de composition. Afin de faciliter cette transformation, nous nous sommes appuyés sur des outils de l'IDM. En effet, les différents méta-modèles sont décrits en ECORE (comme illustré par les Figure 4(a) et 6). La transformation est quant à elle décrite avec ATL[4]. ATL permet de décrire des règles de transformation pour passer d'un modèle à un autre modèle. Ces règles sont ensuite appliquées sur une instance d'un méta-modèle (un fichier XMI) et permet de générer une instance conforme à l'autre méta-modèle.

Dans notre cas, ces règles ont pour but de transformer des *Input*, *Output* et *Action* en ports qui seront associés à un composant en complétant les différentes listes. La Figure 10 est un extrait des règles qui permettent de faire la transformation model-to-model. Les différentes méthodes annexes sur lesquelles s'appuient sur les règles ne sont pas présentées dans la figure.

```
module Java2Prolog; -- Module Template
create OUT : ALIASBehaviorProlog from IN : ALIASBehaviorJava;

rule ALIASBehavior2Component {
 from
  ab : ALIASBehaviorJava!ALIASBehavior
 to
  out : ALIASBehaviorProlog!Component(
   componentName <-ab.id,
   inputUI <- thisModule.inputSet ->collect(e | thisModule.Input2Port(e)),
   outputUI <- thisModule.outputSet ->collect(e | thisModule.Output2Port(e)),
   triggerUI <- thisModule.actionSet ->collect(e | thisModule.Action2Port(e)),
   inputNF <- thisModule.outputSet ->collect(e | thisModule.Output2Port(e)),
   outputNF <- thisModule.inputSet ->collect(e | thisModule.Input2Port(e)),
   triggerNF <- thisModule.actionSet ->collect(e | thisModule.Action2Port(e))
  )
}

lazy rule Input2Port {
 from
  input : ALIASBehaviorJava!Input
 to
  t : ALIASBehaviorProlog!Port (
   portName <- input.id
  )
}
```

Figure 10. *Extrait des règles de transformations décrites en* ATL

4. ATLAS Transformation Language (`www.eclipse.org/m2m/atl/`)

4. Discussions et perspectives

Le framework présenté dans cet article met en place les principaux niveaux relatifs aux architectures d'IHM qui préservent la cohérence et l'indépendance du NF et de l'IHM. Actuellement les modèles intégrés à la plateforme permettent de raisonner essentiellement sur les capacités d'assemblage et de composition. Nous n'avons pas encore mis en place toutes les transformations et en particulier celles qui permettent de construire la partie NF (transformation no 1 de la Figure 3) ni celles qui permettent d'obtenir un rendu des IHM (transformation no 7 de la Figure 3) équivalent aux solutions actuelles proposées par les chercheurs en IHM sur la plasticité. Nos perspectives dans ce cadre sont d'intégrer à la plateforme les résultats de ces travaux autour d'ALIAS-Layout et ALIAS-Structure.

Comme illustré dans cet article, nous avons fait des expérimentations avec plusieurs outils de transformation (ATL, CT et visiteurs java). Nous avons en effet pour objectif à terme de pouvoir réagir à l'apparition et la disparition de services et il est alors important d'évaluer les solutions les plus adaptées pour une approche dynamique de la composition des IHM.

Nous exploitons actuellement le framework essentiellement du point de vue développeur logiciel qui doit minimiser le coût de reconstruction des IHM. En effet, nous sommes partis du principe que les assemblages fonctionnels embarquent l'enchaînement des taches utilisateurs et que nous pouvons donc en déduire le comportement minimal attendu d'une IHM.

La démarche orientée développeur ne prend pas suffisamment en compte les aspects ergonomiques des IHM. Ceci est alors fait au détriment des utilisateurs. En effet les applications actuelles dans le cadre de l'informatique ambiante exploitent de plus en plus la capacité d'un système à découvrir de nouveaux services et de les proposer aux utilisateurs. Cependant ces solutions sont souvent intrusives et ne laissent que peu de possibilités à l'utilisateur de sélectionner les services de son choix et de déterminer la façon dont il souhaite les agencer. Aussi comptons-nous exploiter le framewok afin de construire des applications dédiées à chaque utilisateur qui pourrait sélectionner les services qui l'intéressent et donner des guides de présentations (préférences). Ceci nécessite d'embarquer d'autres modèles plus proches de l'usage ; description du domaine, des tâches, des règles d'ergonomie...

Le fait que le framework repose sur des modèles et des transformations dans une architecture éprouvée pour les systèmes interactifs est prometteur pour atteindre nos objectifs.

5. Remerciements

Nous remercions le projet DGE M-Pub 08 2 93 0707 pour son financement.

6. Bibliographie

Adobe, « Adobe Systems Inc. », http ://www.adobe.com/products/ flex/, 2006.

Bass L. J., Coutaz J., « A metamodel for the runtime architecture of an interactive system : the UIMS tool developers workshop », *SIGCHI Bull.*, vol. 24, n° 1, p. 32-37, 1992.

Bihler P., Fotsing M., Kniesel G., Joffroy C., « Using Conditional Transformations for Semantic User Interface Adaptation », *10th iiWAS International Conference*, ACM, Nov, 2008.

Coutaz J., « PAC : an Implementation Model for Dialog Design », *Conference Proceedings of Interact'87*, Stuttgart, 1987.

Czarnecki K., Helsen S., « Classification of Model Transformation Approaches », *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

Demeure A., Calvary G., Coutaz J., Vanderdonckt J., « The Comets Inspector : Towards Run Time Plasticity Control based on a Semantic Network », *Conference Proceedings of TAMODIA 2006*, 2006.

Gabillon Y., Calvary G., Fiorino H., « L'IDM passerelle entre IHM et planification pour la composition dynamique de systèmes interactifs », *4ème Journées sur l'Ingénierie Dirigée par les Modèles*, p. 51-56, 2008.

IBM, « Service Component Architecture », http ://www-128.ibm.com/developerworks/library/specification/ws-sca/, 2006.

I.S.T.I. - C.N.R., « Cameleon Project : plasticity of user interfaces », http ://-giove.cnuce.cnr.it/cameleon.html, 2004.

Khalaf R., Mukhi N., Weerawarana S., « Service-Oriented Composition in BPEL4WS », *WWW (Alternate Paper Tracks)*, 2003.

Kniesel G., Koch H., Program-independent Composition of Conditional Transformations, Technical Report n° IAI-TR-03-1, ISSN 0944-8535, CS Dept. III, University of Bonn, Germany, July, 2003. updated Feb. 2004.

Microsoft, « XAML Overview », http ://msdn2.microsoft .com/en-us/library/ms752059.aspx, 2007.

Objectweb Consortium, « The Fractal Component Model », http ://fractal.objectweb.org/, 2008.

Open Services Gateway initiative, « OSGi Service Platform (3d Release) », http ://www.osgi.org/, 2003.

Reenskaug T. M. H., « MVC XEROX PARC 1978–1979 », http ://-heim.ifi.uio.no/ trygver/themes/mvc/mvc-index.html, 1979.

Sottet J.-S., Calvary G., Favre J.-M., « Towards Mapping and Model Transformation for Consistency of Plastic User Interfaces », Workshop CHI 06, 2006.

Tarpin-Bernard F., David B., « AMF : un modèle d'architecture multi-agent multi-facettes », *Techniques et Sciences Informatiques*, vol. 18, n° 5, p. 555-586, May, 1999.

Vanderdonckt J., « UsiXML homepage », http ://usixml .org/, 2007.

W3C Working Group, « Web Services Architecture », http ://www.w3.org/TR/ws-arch/, 2004.

L'IDM pour la construction d'un environnement intégré support à la scénarisation pédagogique

Christine Ferraris* — **Salim Ouari**** — **Laurence vignollet*** — **Christian Martel***

* *Syscom, Université de Savoie, Campus Scientifique, 73376 Le Bourget-du-Lac cedex*
{christine.ferraris,salim.ouari,laurence.vignollet,christian.martel}@univ-savoie.fr

** *LIG, équipe metah, Campus de St-Martin-d'Hères, 38041 Grenoble cedex*

RÉSUMÉ. *La scénarisation des activités pédagogiques semble une voie prometteuse, capable de répondre aux enjeux de l'e-formation. La réussite de cette ingénierie spécifique, appelée Learning Design, dépend en grande partie de la capacité de ses promoteurs à se doter des outils réellement adaptés aux professionnels exerçant dans les entreprises du domaine. Si plusieurs langages arrivent aujourd'hui à maturité dans le domaine de l'expression et de la formalisation des activités pédagogiques, les outils associés sont encore trop difficiles à manipuler par les ingénieurs pédagogiques et trop peu intégrés. La réalisation d'un IDE servant de support à cette ingénierie devient cruciale. Nous envisageons la construction d'un tel IDE en nous appuyant sur une approche IDM, qui devrait en particulier faciliter la définition et l'usage de DSL destinés à ces ingénieurs.*

ABSTRACT. *Learning Design seems to be a promising way to answer to the e-learning stakes. Its success depends largely on its ability to offer tools really adapted to the e-learning professionals. Even if several languages have come to maturity in this field, associated authoring tools are still too difficult to be manipulated by the instructional designers. That's why we propose to design and build an IDE dedicated to Learning Design. The realization of such a framework should take advantage from using theoretical and practical results of MDE. In particular, the integration in this framework of DSL dedicated to pedagogical engineers should be eased.*

MOTS-CLÉS : *Domain Specific Language (DSL), Environnement intégré de développement (IDE), Learning Design (LD), Procédés Pédagogiques (PP), Application de l'IDM*

KEYWORDS: *Domain Specific Language (DSL), Integrated development Framework, Learning Design (LD), Pedagogical Procedures (PP), application of MDE*

1. Introduction

L'augmentation qualitative et quantitative de la demande en e-formations, due en grande partie au développement de la formation tout au long de la vie, a conduit les ingénieurs pédagogiques en charge de la réalisation des produits de formation à prendre progressivement conscience de la place centrale occupée par la conception (Paquette, 2002). Ils éprouvent la nécessité de faire évoluer leurs pratiques vers une véritable ingénierie spécifique fondée sur des modèles, des méthodes, des outils et des normes industrielles. Le learning design (LD) (Koper *et al.,* 2005) semble être une voie prometteuse pour cette ingénierie. Il propose en effet non seulement des modèles pour décrire des activités pédagogiques (par exemple IMS-LD (IMSLD, 2003) ou LDL (Martel *et al.,* 2006c)) mais aussi des langages pour formaliser ces modèles et des outils d'interprétation de ces langages pour transformer les modèles d'activité en de véritable activités tournant sur des environnements de travail en ligne. Des cycles d'ingénierie ont été identifiés (Martel *et al.,* 2007) et instrumentés. Pour la conception notamment, des éditeurs fondés sur les langages de modélisation ont été développés. C'est par exemple le cas de ModelEditor pour construire des scenarios LDL, de Reload (Milligan *et al.,* 2005) et plus récemment Recourse (Griffiths *et al.,* 2008) pour des scénarios IMS-LD.

Il semble cependant que, malgré les efforts réalisés, les résultats n'aient pas encore atteint un niveau de maturité tel que les ingénieurs pédagogiques ou les enseignants puissent s'en emparer et naturellement les mettre en oeuvre. Force est de constater, en effet, qu'il n'existe pas, dans ce domaine de l'ingénierie, d'environnement de conception offrant à l'ingénieur pédagogique, à travers une interface unique, la possibilité :

– de choisir, dans un entrepôt de méthodes, un scénario correspondant à l'activité pour laquelle il souhaite développer un Environnement Informatique d'Apprentissage Humain (EIAH) ou, si un tel scénario n'existe pas, de le modéliser à partir de primitives facilement appréhendables,

– de spécialiser ce scénario à la résolution d'un problème spécifique en lui ajoutant les objets d'apprentissage et les services qui conviennent,

– de choisir l'Environnement Numérique de Travail (ENT) le plus adapté au type d'activité envisagé,

– d'opérationnaliser simplement cette activité dans cet environnement,

– d'en suivre le déroulement,

– de modifier éventuellement le scénario initial à partir des usages observés lors de ce suivi.

La réalisation d'un tel environnement devient ainsi incontournable pour qu'une véritable ingénierie pédagogique puisse se développer. Nous en expliquons les raisons dans la section 2 de cet article. Un tel environnement peut être vu comme un IDE (environnement de développement intégré) support à la scénarisation pédagogique. Nous décrivons plus précisément, dans la section 3, ce que doit être, de notre point de vue, un tel IDE. Dans la partie 4, nous expliquons pourquoi, après des premiers dévelop-

pements « ad-hoc », nous avons fait le pari d'opter pour une approche d'ingénierie dirigée par les modèles pour la réalisation de cet environnement. La partie 5 décrit les composants d'ores et déjà développés pour cet IDE : elle met notamment en avant le DSL proposé pour les ingénieurs pédagogiques et les outils associés. Nous évoquons au final les problèmes que nous rencontrons actuellement.

2. Pourquoi un environnement intégré support à la scénarisation pédagogique.

2.1. *Supporter la totalité d'un cycle de vie complexe : celui des EIAH*

Les Environnements Informatiques d'Apprentissage Humains sont généralement considérés comme des applications très difficiles à spécifier et à modéliser, en raison des nombreux facteurs personnels et interpersonnels qui conditionnent les apprentissages et influent sur leur déroulement. Les résultats récents de la recherche dans le domaine de la scénarisation des activités pédagogiques semblent montrer qu'il est possible de gérer cette difficulté et d'offrir de nouvelles perspectives à l'industrie de la formation en ligne. Ces travaux mettent clairement en évidence, à travers de nombreux exemples, l'existence d'un cycle de vie des EIAH composé de cinq phases (Hernandez-Leo *et al*., 2007) :

– une phase de définition « métier » du scénario par un concepteur initial qui peut être un enseignant ou un ingénieur spécialisé,

– une phase de formalisation du scénario métier dans un ou plusieurs langages de notation génériques,

– une phase d'opérationnalisation du scénario formalisé prenant compte des caractéristiques diverses des différentes infrastructures d'exécution,

– une phase d'ajustement du scénario opérationnalisé en fonction des observations effectuées durant son exécution,

– une phase d'analyse et de conception renouvelée du scénario en vue de sa réutilisation ultérieure.

Une ingénierie spécifique doit être développée pour supporter ces cinq phases. Cette ingénierie doit trouver appui sur un environnement support dédié.

2.2. *En raison du public auquel on s'adresse*

Les phases mentionnées ci-dessus supposent de faire appel à des formalismes différents pour modéliser et représenter les modèles sous-jacents. S'il est envisageable de demander à des ingénieurs en informatique de jouer avec ces différents formalismes et de les manipuler, cela n'est en revanche pas possible pour les personnes qui sont les vrais acteurs de la conception pédagogique en ligne : les ingénieurs pédagogiques mais aussi les enseignants. Toute la complexité de l'élaboration et de la mise en oeuvre

de ces modèles doit de fait être cachée derrière une interface de manipulation simple et unique, qui aborde le problème sous l'angle de la définition "métier".

Par ailleurs, l'étude des usages de ces acteurs (Henri *et al.,* 2007) montre l'écart qui existe actuellement entre, d'une part, les méthodes formalisées et parfois normalisées proposées (sur lesquelles s'appuient les outils), et d'autre part, les pratiques effectives. Face à ces résultats, les auteurs plaident pour une approche de conception mieux adaptée : "on aurait sans doute avantage à disposer de méthodes plus souples, moins linéaires et plus adaptables à différentes approches de conception. La progression linéaire et séquentielle suggérée par l'approche traditionnelle incite le concepteur à déterminer dès le départ les orientations pédagogiques [...]. Une nouvelle approche pourrait tenter de respecter l'importance centrale accordée au contenu tout en établissant un lien direct entre contenu et pédagogie [...] ". La réalisation d'un IDE est un moyen de repenser cette approche.

2.3. *Les solutions existantes ne sont pas satisfaisantes*

Le développement de méthodes[1] pour l'ingénierie pédagogique fait l'objet de différents travaux. Certains se focalisent sur la phase de définition "métier" en travaillant à la définition de modèles et langages pour décrire une activité pédagogique (voir par exemple les langages visuels décrits dans (Botturi *et al.,* 2008)). Il s'agit ici d'expliciter un DSL. L'ingénieur pédagogique est alors confronté à la multitude des langages existants et il lui est difficile de choisir le langage le mieux adapté à la situation qu'il souhaite modéliser. La plupart des travaux se limitent par ailleurs le plus souvent à une description du DSL, sans qu'un support informatique à l'utilisation de ce DSL ne soit fourni. Lorsqu'un tel support existe, le passage à un modèle computationnel est rarement traité.

Les approches "Learning Design" considèrent la totalité du cycle de vie d'un scénario. Des instrumentations complètes de ce cycle de vie ont été réalisées avec par exemple les travaux réalisés sur LDL et IMS-LD ou encore l'environnement LAMS (Dalziel, 2008). LDL et IMS-LD ont mis l'accent sur l'expressivité du langage. LDL est ainsi fondé sur l'analyse de ce qu'est une activité collaborative. Il leur est cependant reproché d'être trop abstraits, trop complexes et donc difficilement manipulables par des ingénieurs pédagogiques (les éditeurs associés supposent en effet de connaître le modèle sous-jacent). De plus, les outils associés sont peu intégrés. Lams a pris une position opposée en mettant l'accent, de façon très pragmatique, sur la facilité de construction d'un scénario et sur l'intégration. Une "séquence" Lams est ainsi un enchainement d'outils (un forum, un éditeur support à la rédaction d'un devoir, ...) convoqués au sein d'une activité pédagogique. Son expression ne repose pas sur un méta-modèle explicite ni fondé de ce qu'est une activité d'apprentissage. C'est cette fois-ci l'expressivité qui est mise en cause.

1. au sens où l'entendent les chercheurs en système d'information, à savoir des modèles, des langages, des outils et des méthodologies.

3. Qu'est-ce qu'un IDE pour la scénarisation pédagogique ?

Le concept d'IDE figure parfaitement l'outil que nous souhaitons réaliser. Si un tel terme est en général reservé à des environnements de développement destinés à des informaticiens, il convient ici d'en faire une version spécifique pour les ingénieurs pédagogiques. Ces derniers pourront ainsi construire simplement des EIAH en exhibant le modèle de l'activité visée.

La première étape pour aboutir à cet IDE a donc été d'analyser et de comprendre ce qu'un ingénieur pédagogique est prêt à faire pour formaliser un scénario, l'opérationaliser, l'exécuter, le réguler et le réviser. Cette analyse a permis de dégager les besoins en matière de scénarisation pédagogique. Ils sont présentés dans ce qui suit et sont suivis de la description de l'architecture logique de l'IDE.

3.1. *Les besoins en matière de scénarisation pédagogique*

L'IDE donne *accès à l'ensemble des outils supports au cycle de vie des scénarios pédagogiques*. L'ingénieur pédagogique peut ainsi, à partir de la même interface, concevoir, opérationaliser, exécuter, superviser, réguler et réviser les scénarios d'apprentissage qu'il souhaite mettre en oeuvre.

Etant entendu que dans la vision actuelle du cycle de vie ces étapes sont entrelacées, *l'interface de l'IDE doit permettre le passage entre les différents modèles supports à chaque étape du cycle de vie du scénario* : le modèle de l'activité (le scénario), le modèle de l'opérationalisation et le modèle des services ou de la plate-forme cible. Il doit par exemple être possible de désigner certaines ressources manipulées dans le scénario, sans attendre que la conception de ce dernier ne soit finalisée.

Cet IDE doit être ouvert pour permettre l'utilisation de plusieurs langages de formalisation . Nous avons indiqué que la manipulation des concepts du domaine, à travers un langage "métier", s'impose aujourd'hui comme point d'entrée de l'environnement, pour en faciliter la prise en main par les ingénieurs pédagogiques. Toutefois, cette première formalisation n'est qu'une étape dans l'obtention d'un scénario opérationalisable, les langages de modélisation pédagogique tels que LDL ou IMS-LD restant le formalisme cible pour permettre l'exécution des scénarios.

L'IDE doit donc aussi *offrir des guides méthodologiques et des supports technologiques à la transformation des scénarios* pour permettre d'aboutir à une version opérationalisable du scénario. A terme, cela devrait permettre de faire cohabiter plusieurs langages "métier", voire plusieurs langages de modélisation pédagogiques.

L'ouverture préconisée s'applique aussi aux ressources (Learning Objects, services des LMS, référentiels de compétences, annuaires, etc.) qu'il doit être possible d'utiliser pour mener à bien une activité d'apprentissage. En effet, devant l'augmentation de l'offre de telles ressources, *l'IDE se doit d'inclure des fonctionnalités permettant*

de sélectionner, déployer et exécuter des ressources hétérogènes en provenance de sources ou plates-formes différentes. Il est donc un moyen d'en homogénéiser l'accès.

Enfin, l'IDE doit être complètement *accessible via le Web*, ceci afin d'affranchir les utilisateurs finaux de toute installation qui nécessiterait des compétences qu'ils ne sont pas censés avoir. Cela devrait faciliter sa dissémination.

Il s'agit de fait d'un atelier de génie logiciel dédié au développement de formations en ligne.

3.2. *Architecture logique de l'IDE*

Les besoins énoncés ont guidé la définition de l'architecture logique de l'IDE, représentée sur la figure 1.

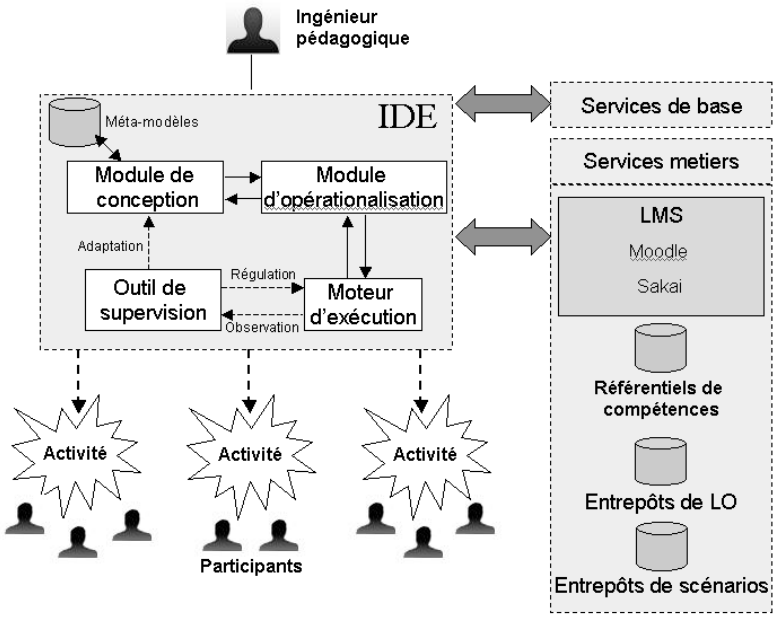


Figure 1. *L'architecture logique de l'IDE*

L'ingénieur pédagogique pourra choisir, dans un entrepôt de scénarios prédéfinis, celui correspondant à l'activité qu'il souhaite mettre en oeuvre. Il aura la possibilité de l'adapter à son besoin. L'IDE lui donnera aussi les moyens d'opérationnaliser ce scénario, c'est-à-dire de désigner les ressources et les participants d'une part, et d'assurer

le déploiement des objets correspondants dans les environnement cibles d'autre part (par exemple, une ressource moodle pourra être créée si elle n'existe pas déjà). Une fois complété, le scénario pourra être exécuté. L'IDE fournira les interfaces d'exécution et de supervision aux participants à l'activité, conformément à leurs rôles. Enfin, les résultats produits par l'observation, en fonction du choix des observables opéré lors de la création du scénario, permettront la régulation de l'activité en cours, ou l'amélioration du scénario initial dans une perspective de réingéniérie pédagogique.

S'il souhaite construire un nouveau scénario, l'ingénieur pédagogique devra pouvoir tout d'abord en donner une version simple, exprimée à l'aide de concepts "métier". Il aura la possibilité à tout moment d'y intégrer les ressources (contenus et services) nécessaires à la réalisation de l'activité visée. L'étape suivante consistera à transformer ce scénario pour le formaliser dans un langage de scénarisation pédagogique.

Plusieurs méta-modèles co-existent donc. L'IDE devra fournir les moyens de faciliter le passage de l'un à l'autre, par des transformations automatiques ou guidées. Il devra aussi conserver en permanence les liens entre les différents modèles pour en garantir la cohérence.

4. Les raisons de choisir l'IDM

L'IDM nous semble être une réponse au problème de la complexité de la modélisation pédagogique. L'IDE à développer s'adresse, nous l'avons mentionné (cf. 2.2), à des ingénieurs pédagogiques. Il doit leur permettre de se focaliser sur leur coeur de métier (la conception de situations d'apprentissage) sans avoir à se préoccuper (ou le moins possible) de problèmes techniques. Il faut pour cela leur fournir un ou plusieurs langages "métier" adaptés (des DSL). Pour construire l'IDE, nous avons de fait besoin de définir de tels langages et de les tester auprès de ces usagers. Nous avons par ailleurs besoin de disposer de mécanismes de transformation des modèles produits avec ces DSL vers des modèles plus «informatiques».

L'approche IDM (Bézivin, 2004), et notamment le Domain Specific Modelling (Kelly *et al.,* 2008), semble fournir le cadre conceptuel et technique répondant à ces besoins. Elle nous permet de réfléchir à l'introduction de nombreux DSL à moindre coût : elle fournit en effet les moyens de générer assez rapidement des éditeurs correspondants et de passer relativement facilement d'un langage à l'autre.

Par ailleurs, les différentes phases identifiées dans le processus d'ingénierie pédagogique mettent en oeuvre des modèles relativement hétérogènes. Modèles d'activités, modèles d'opérationalisation, modèles de plate-formes (LMS) et profils d'usagers forment les pièces du puzzle que l'ingénieur pédagogique doit désormais assembler et entrelacer pour obtenir l'Environnement Hautement Interactif qu'il imagine et qu'attendent les apprenants, en formation initiale ou en formation continue. Les propositions à la fois théoriques et pratiques formulées dans le cadre de l'IDM semblent, là

encore, fournir un support intéressant quand il s'agit de tisser les différents modèles ou d'appliquer des transformations.

Enfin, l'IDM laisse entrevoir des solutions pour traiter de l'adaptation des activités d'apprentissage, de leur observation, de leur régulation et de leur évaluation. Ces perspectives seraient un grand apport du point de vue du Learning Design. Nous nous intéressons principalement aux propositions formulées pour modifier le modèle d'une activité pendant son exécution (run time). En effet, même si l'adaptation "à la volée" d'une activité en cours à des événements non prévus dans son modèle est inhérente au domaine de l'apprentissage, aucun des EIAH actuellement proposés ne la rend possible.

5. Les premiers composants de l'IDE

5.1. *Le socle de base : LDL*

Nous disposons d'ores et déjà d'un certain nombre de briques entrant dans la construction de l'IDE. En ce qui concerne les meta-modèles et les langages associés, nous avons défini le langage de scénarisation LDL (Martel *et al.,* 2006b). A LDL sont associés un éditeur graphique, qui couvre le module de conception, ainsi qu'une infrastructure (LDI) qui englobe pour sa part l'opérationnalisation, l'exécution et des interfaces de suivi de l'activité.

LDL est destiné à produire des modèles d'activités collaboratives. Son élaboration s'est appuyée sur une analyse rigoureuse de ce que sont ces situations, analyse elle-même fondée sur des résultats de recherche en sciences humaines (en sociologie, linguistique et ethnométhodologie notamment - voir (Ferraris *et al.,* 2008) pour plus de détails). Le nombre de concepts y est limité (7 concepts principaux) pour que le langage reste relativement simple. Il a fait la preuve de son expressivité lors de la modélisation de plusieurs activités collaboratives variées et complexes (SVL (Lejeune *et al.,* 2007), Mates (Adam *et al.,* 2008), Planet Game (Martel *et al.,* 2006a)) . Par ailleurs, LDL est un langage computationnel. Tout scénario décrit en LDL peut ainsi être interprété pour être opérationnalisé et transformé en une activité en ligne. Cela a été fait pour les applications citées précédemment, dont deux ont donné lieu à des expérimentations écologiquement fondés dans des lycées et collèges (SVL et Mates). C'est pourquoi LDL est une pièce maîtresse dans la construction de l'IDE.

Sa relative simplicité a cependant une contrepartie : les concepts proposés sont pour certains très abstraits. Ainsi, lors des travaux réalisés sur SVL et Mates par exemple, nous avons pu constater que les ingénieurs pédagogiques ou les enseignants ne pouvaient pas eux-mêmes formaliser les activités, ou difficilement. L'éditeur graphique associé à LDL s'est avéré trop complexe car trop proche du langage lui-même.

C'est pourquoi, en collaboration étroite avec des ingénieurs pédagogiques, nous avons travaillé à la définition d'un DSL adapté aux situations qu'ils ont l'habitude de mettre en oeuvre, qui va constituer une autre des briques de l'IDE. Ce DSL s'ap-

puie sur le concept de *procédé pedagogique* (PP), que nous avons formalisé. Un éditeur de PP a été développé, conformément à une approche IDM, avec les outils GMF d'eclipse. Nous travaillons à l'heure actuelle sur le problème de la transformation de modèles, du modèle de PPs vers LDL. Cela nous permettra de continuer à bénéficier de l'expressivité de LDL et de son caractère computationnel. Ces différents élements sont présentés dans ce qui suit.

5.2. *Le DSL des PP*

5.2.1. *Les procédés pédagogiques*

La scénarisation des activités pédagogiques est un domaine de la conception qui s'appuie sur une intuition partagée par bon nombre d'enseignants : certaines activités sont plus propices que d'autres aux apprentissages et favorisent la construction des connaissances par les élèves. Ainsi, l'expérimentation, le débat et la confrontation argumentée organisée entre les élèves en vue d'une compréhension fine des notions relatives au courant électrique vaudra mieux qu'une simple présentation magistrale de ces notions, comme en témoigne la réussite du projet Mates (Methodology And Tools for Experiment Scenarios) du réseau Kaléidoscope. Ces activités pédagogiques jugées plus efficaces que d'autres pour l'apprentissage, se transmettent entre les générations d'enseignants sous la forme de "recettes" qui, lorsqu'elles sont correctement appliquées, favorisent l'apprentissage des élèves. Le procédé La Martinière [2], largement et depuis longtemps utilisé dans l'enseignement primaire, en est un exemple. Il est réputé permettre de développer l'activité de calcul mental et faciliter l'acquisition des notions de base en calcul. Il est à ce titre cité comme un des procédés de mise en oeuvre du calcul mental par les instructions officielles de 2007 (BO n°10 2007).

Sans vouloir discuter ici de l'efficacité supposée de ces "recettes", c'est leur existence en tant que telle et le fait que cette existence est reconnue par un grand nombre d'enseignants qui permet à l'ingénieur pédagogique de s'appuyer sur elles pour proposer à ces mêmes enseignants des scénarios pédagogiques dont ils peuvent facilement comprendre l'intérêt et la mise en oeuvre dans des activités susceptibles de faire progresser leurs élèves.

5.2.2. *Définition des PP*

Un procédé pédagogique est un scénario particulier qui n'est pas lié à une discipline précise. Il se distingue des autres scénarios par son caractère relativement codifié, son degré de partage dans la communauté enseignante et par le fait que cette communauté reconnaît sa capacité à mener à un apprentissage effectif. Il est défini par un ensemble d'*instructions* destinées aux futurs *participants* de l'activité pédagogique, indiquant ce qu'ils ont à faire. Ces instructions implique des *artefacts* que ces

2. Il s'agit du procédé qui consiste pour l'enseignant à énoncer un calcul, pour les élèves à effectuer mentalement ce calcul, à inscrire le résultat sur une ardoise et à lever cette ardoise en direction de l'enseignant pour lui montrer ce résultat.

participants vont transformer tout au long du scénario. Les instructions explicitent les transformations et peuvent être regroupées en *phases*.

Par exemple, le procédé pédagogique largement utilisé dans les dispositifs de formation qui consiste à demander à un élève de faire un exposé peut se décrire, selon la définition précédente, de la manière presentée dans la figure 2.

Phases	**Participant**	**Instructions**	**Artefact : *exemple***
Sélection	Elève	Choisir le sujet	Le sujet : *les OGM*
Recherche	Elève	Collecter les informations	Les informations : *textes de loi, documents scientifiques*
	Elève	Analyser les informations	L'analyse : *problèmes, risques*
Production	Elève	Construire la présentation	La présentation : *schémas, croquis, définitions*
Présentation	Elève	Exposer le travail réalisé	L'exposé : *Public, questions, temps*
Evaluation	Enseignant	Evaluer le travail réalisé	La note : *12, 11*

Figure 2. *Description du procédé pédagogique "Exposé"*

De nombreux procédés pédagogiques, plus ou moins codifiés, sont repérables dans les situations d'enseignement pratiquées en formation (Villiot-Leclercq, 2007). Il est impossible d'en établir une liste exhaustive et, ce, d'autant moins que ces procédés sont régulièrement remaniés et modernisés (voir par exemple la description du procédé La Martinière de 2007 (BO n°10 2007) versus celle de 1951 (Rossignol, 1951)). Pour des raisons méthodologiques et pratiques et en accord avec les ingénieurs pédagogiques, nous en avons retenu huit (Martel *et al.,* 2008), qui ont été formalisés et viendront augmenter l'entrepôt de scénarios mis à leur disposition via l'IDE.

5.2.3. *Le méta-modèle des PP*

La figure 3 présente le méta-modèle des PP. Ce méta-modèle définit les concepts composant un PP et leurs relations. Il a été construit à partir de la définition d'un PP, elle-même élaborée et raffinée au fur et à mesure de la modélisation des premiers PPs sélectionnés. Il introduit notamment les *partitions*(au sens musical du terme, dont la traduction est "score" en anglais), concept qui s'est imposé lors du travail de coconception des PPs menés avec les ingénieurs pédagogiques.

Un PP est considéré comme une oeuvre musicale dans laquelle sont impliqués plusieurs *participants* (l'élève, l'enseignant, le tuteur, etc.). Chaque participant contribue à l'oeuvre globale en "jouant" sa propre *partition*. Le PP est donc défini par l'ensemble des partitions. Nous avons pris ici la métaphore de l'orchestre qui s'est avérée facilitatrice lors des séances de travail avec les ingénieurs pédagogiques. Une partition, en reprenant cette métaphore, peut se décomposer en plusieurs "mouvements", que nous

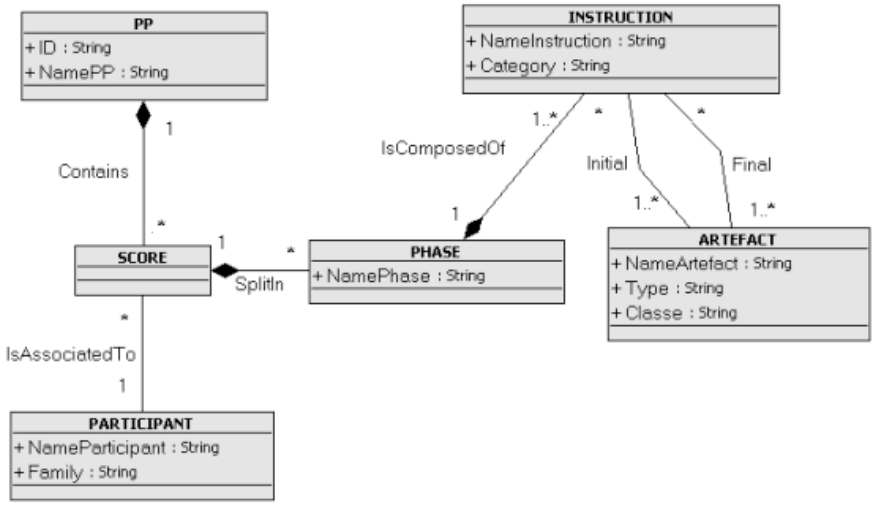


Figure 3. *Le méta-modèle des PP (en version épurée pour les attributs).*

avons appelés des *phases* (une sélection, une recherche, etc.), conformément au souhait des ingénieurs pédagogiques. A une phase correspond au moins une *instruction* (Collecter les informations, Analyser les informations, etc.). Une instruction n'est ni plus ni moins que la transformation d'un artefact en un autre, comme les mesures se transforment en une (jolie) musique. Une première catégorisation des instructions a été entreprise avec l'aide des ingénieurs pédagogiques. Il s'agit par exemple de distinguer les instructions relatives à une production (définir, construire, etc.), de celles relatives à une recherche (collecter, analyser, etc.). Nous avons ainsi défini plus d'une vingtaine de catégories. L'introduction de ces catégories a pour but principal de faciliter la sélection des instructions lors de la construction d'un nouveau PP. De même, nous avons identifié un premier ensemble de classes d'artefacts : les artefacts sur lesquels portent le procédé (une notion, un cas, un sujet, etc.), ceux qui accompagnent et qui cadrent (un objectif pédagogique, une méthodologie, des règles, etc.), les artefacts ressources (une source, un indice, une information, une solution, etc.), ceux qui sont produits (synthèse, présentation, cartographie, etc...). Un artefact possède par ailleurs un type qui désigne sa nature (un document, une image, un exercice, etc...).

Ce meta-modèle est la base sur laquelle le développement de l'éditeur de PPs s'est appuyé.

5.3. *PPDesigner : un éditeur pour les PP*

Les hypothèses de représentation des procédés pédagogiques prennent en compte les différentes entités énumérées précédemment (les participants, les instructions, les phases et les artefacts) et leurs relations. L'ingénieur pédagogique va devoir construire un procédé pédagogique à partir de ces seules entités en proposant de les lier les unes aux autres et en explicitant les actions par lesquelles il les combine entre elles.

PPdesigner a été conçu dans le cadre d'une démarche de co-design impliquant des ingénieurs pédagogiques. Nous avons eu recours à l'IDM à la fois pour la spécification de son interface et pour son développement. En ce qui concerne l'interface, nous nous sommes appuyés sur la méthode de spécification introduite par (Sottet *et al.,* 2005).

En ce qui concerne le développement, nous avons travaillé avec l'environnement EMF/GMF d'eclipse (Arboleda *et al.,* 2007). Le développement d'un éditeur dans GMF se fait en plusieurs étapes impliquant chacune la définition d'un modèle spécifique, avant de pouvoir générer le code correspondant. Nous avons produit les modèles demandés par GMF à savoir les trois modèles de base permettant de décrire le DSL et l'éditeur graphique (le modèle du domaine, le modèle de la définition graphique et modèle de la palette graphique) et le modèle de liaison mettant en relation ces trois modèles de base.

PPdesigner a été conçu et réalisé tout en respectant les souhaits des futurs utilisateurs. Nous avons travaillé conjointement avec eux tout au long de la phase de conception et de réalisation de l'éditeur. Le fait d'avoir opté pour une démarche IDM nous a facilité la tâche, les modifications portant sur les modèles. Modifier directement le code informatique eût été plus long, plus complexe et plus coûteux. La figure 4 présente une copie d'écran de l'éditeur réalisé.

5.4. *Spécification des Transformations des PP vers LDL*

Le DSL des procédés pédagogiques et l'éditeur associé ont été introduits, nous l'avons vu, dans l'objectif de rendre accessible et simple à des ingénieurs pédagogiques la tâche de modélisation d'une activité d'apprentissage (un éditeur simple d'utilisation et des concepts pour modéliser qu'ils peuvent s'approprier facilement). Le DSL des PPs n'est en effet pas un langage computationnel (au sens de Botturi et al. (Botturi *et al.,* 2006)) contrairement à LDL qui a été fait pour cela. Il convient donc de définir des règles pour transformer un PP décrit selon le méta-modèle des PPs en un scénario décrivant la même activité, exprimé en LDL. Cela revient à établir, dans la mesure du possible, les correspondances existant entre les entités du domaine des procédés et celles du domaine des scénarios.

Par correspondance, il faut comprendre la traduction d'une entité d'un domaine dans une ou plusieurs entités de l'autre domaine, par exemple la traduction d'une *instruction* en entités du domaine de la scénarisation. Le langage de scénarisation LDL s'appuie sur un méta-modèle dans lequel les *scénarios* sont décrits comme une suite

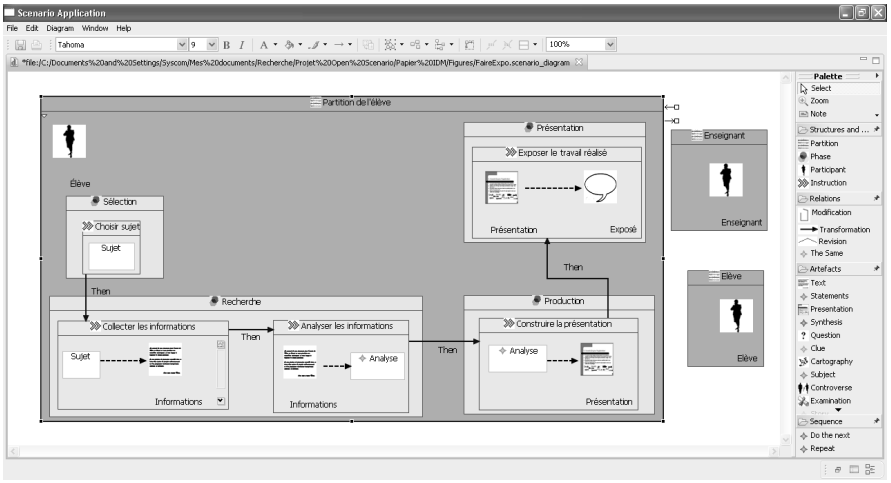


Figure 4. *Edition de la partition "élève" du PP "Exposé"*

d' *interactions* (appelée *structure*) entre des *rôles* au sein de différentes *enceintes* (les lieux des interactions : des services et contenus) soumises à des *règles* dont l'exécution dépend des observations réalisées dans l'environnement (les *observables*) et des *positions* exprimées par les acteurs au cours du déroulement. La figure 5 montre un exemple de scénario LDL partiel, représenté conformément à la notation graphique définie dans (Ferraris *et al.*, 2008). Il est composé d'une structure séquentielle comportant deux interactions ("demander de choisir un sujet" et "choisir un sujet") se déroulant dans 2 enceintes différentes (la "liste de sujets" et "le sujet"). Ces interactions impliquent chacune deux acteurs représentés par leurs rôles : un acteur qui agit (placé du côté gauche de la flèche) et un acteur à qui est destiné l'action (en bout de flèche).

De ce point de vue, une *instruction* n'est rien d'autre qu'une structure LDL articulant deux interactions successives, l'une considérée comme initiatrice de l'échange dans laquelle un acteur (un rôle) donne l'ordre de faire quelque chose à un autre acteur, et la suivante considérée comme réactive dans laquelle l'acteur sollicité produit la réponse attendue par l'ordonnateur.

Cette décomposition ou plus exactement cette transposition, conforme à l'idée générale qu'il est possible de se faire d'un acte de langage par lequel s'exprime un ordre (ou une instruction, forme moins impérative), montre la voie d'une traduction possible d'un domaine dans un autre. Il en va ainsi de la correspondance entre les *artefacts* et les *enceintes*, de l'analogie des *phases* et des *structures*, de l'attribution des *rôles* aux *participants*. D'une manière générale, et après l'avoir vérifié sur les procédés sélectionnés, le premier palier de traduction des procédés en scénarios peut être atteint sans

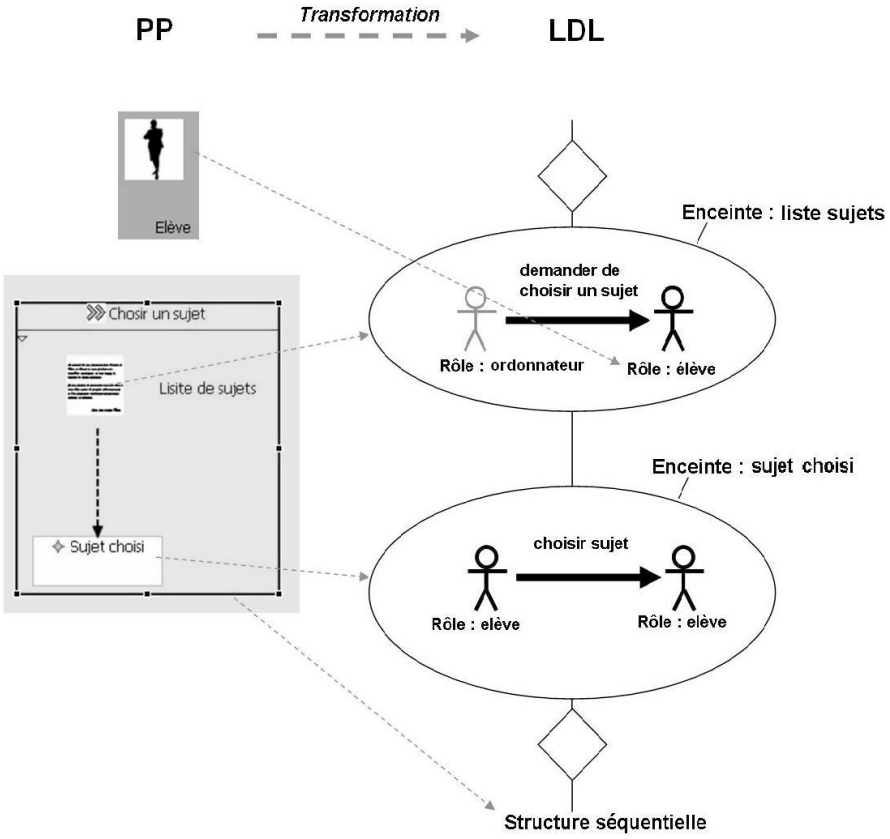


Figure 5. *Du DSL des PPs vers LDL : premier palier de transformation.*

trop de difficultés. Nous travaillons en ce moment à leur formalisation en essayant d'utiliser ATL.

Les étapes suivantes s'annoncent beaucoup plus difficiles à franchir. Le scénario obtenu au premier palier est une sous-spécification d'un véritable scénario LDL. LDL fournit en effet des informations quant à l'activité qui sont absentes du méta-modèle des PPs. Il n'y a par exemple aucun concept pour modéliser les circonstances dans lesquelles une phase ou une instruction débute ou se termine ; en LDL, cela est exprimé par des règles sur les structures et les interactions, règles qui testent par ailleurs les positions des participants. Rien dans le méta-modèle des PPs ne permet par ailleurs de spécifier des positions. De même, rien ne permet de définir la synchronisation entre partitions dans un PP alors que cela est inhérent aux concepts de *structure* et d'*interaction* dans LDL.

Nous réfléchissons en ce moment au moyen de franchir ces paliers. Cela nécessite l'intervention de l'ingénieur pédagogique pour fournir des informations qui n'existent pas dans les PPs. Il faut pour cela étendre PPdesigner de manière à pouvoir passer progressivement d'un éditeur de PP à un éditeur de scénario LDL en cachant la complexité des concepts de LDL derrière des interfaces que l'ingénieur pédagogique pourra appréhender facilement. Le problème reste pour le moment entier aussi bien au niveau des interfaces à produire que des modèles sous-jacents.

6. Conclusion

La conception est désormais au centre du processus de production des e-formations. Cette démarche qu'empruntent de nombreux ingénieurs pédagogiques est prometteuse dans le sens où elle va accentuer la réutilisabilité et le partage des scénarios pédagogiques entre les acteurs de la formation et accroître considérablement l'offre. Dans le même temps, elle va permettre de faire participer plus clairement les ingénieurs pédagogiques à la conception des outils destinés à la modélisation des activités pédagogiques. Dans cet article, nous avons présenté les premiers éléments d'un environnement intégré de support à l'activité de conception pédagogique, dont la réalisation s'appuie sur une démarche IDM. Nous avons fait le pari qu'une telle approche permettra d'atteindre plus facilement et plus rapidement nos objectifs. L'enjeu n'est pas mince puisqu'il conditionne probablement la possibilité pour les enseignants eux-mêmes d'utiliser ces outils et de les faire évoluer selon leurs voeux.

7. Bibliographie

Adam J., Lejeune A., Michelet S., David J.-P., Martel C., « Setting up on-line learning experiments : the LearningLab platform », *The 8th IEEE International Conference on Advanced Learning Technologies*, 2008.

Arboleda H., Casallas R., Royer J., « Implementing an MDA Approach for Managing Variability in Product Line Construction Using the GMF and GME Frameworks », *5th Nordic Workshop on Model Driven Software Engineering. August*, p. 27-29, 2007.

Bézivin J., « Sur les principes de base de l'ingénierie des modèles », *Revue RSTI-L'Objet*p. 145-156, 2004.

Botturi L., Derntl M.and Boot E., Figl K., « A Classification Framework for Educational Modeling Languages in Instructional Design », *IEEE ICALT*, p. 1216-1220, 2006.

Botturi L., Stubbs T., *Handbook of Visual Languages for Instructional Design : Theories and Practices*, Hershey PA : IGI Global, 2008.

Dalziel J., « Using LAMS Version 2 for a game-based Learning Design », *Journal of Interactive Media in Education, Special issue on Comparing Educational Modelling Languages on the "Planet Game" Case Study*, 2008.

Ferraris C., Martel C., Vignollet L., « LDL for Collaborative Activities », *in* , L. Botturi, , T. Stubbs (eds), *Handbook of Visual Languages for Instructional Design : Theories and Practices*, Hershey PA : IGI Global, chapter 12, p. 226-253, 2008.

Griffiths D., et al., « IMS-LD Authoring, Livrable du projet TenCompetence, IST-2005-027087 », Janvier, 2008, disponible sur Internet : http ://hdl.handle.net/1820/1149 (consulté le 08/04/08).

Henri F., Teja I., Lundgren K., Ruelland D., Maina M., Basque J., Cano J., « Pratique de design pédagogique et objets d'apprentissage », mars, 2007, Actes du colloque Initiatives 2005, Débat thématique, disponible sur Internet : http ://www.initiatives.refer.org/Initiatives-2005/document.php ?id=143. (consulté le 08/04/08).

Hernandez-Leo D., Bote-Lorenzo M., Asensio-Perez J., Gomez-Sanchez E., Villasclaras-Fernandez E., Jorrin-Abellan I., Dimitriadis Y., « Free and Open-Source Software for a Course on Network Management : Authoring and Enactment of Scripts Based on Collaborative Learning Strategies », *IEEE Transactions on Education*, vol. 50, n° 4, p. 292-301, 2007.

IMSLD, « IMS Learning Design Information Model, version 1.0. IMS Global Learning Consortium Inc. », 2003, disponible sur internet : http ://www.imsglobal.org/learningdesign/index.html (consulté le 08/04/08).

Kelly S., Tolvanen J.-P., *Domain Specific Modeling : Enabling Full Code Generation*, Wiley-IEEE Computer Society Press, 2008.

Koper R., Tattersall C., *Learning Design : A Handbook on Modelling and Delivering Networked Education and Training*, Springer, 2005.

Lejeune A., David J., Martel C., Michelet S., Vezian N., « To set up pedagogical experiments in a virtual lab : methodology and first results », *Proceedings of the International Conference of "Interactive computer aided learning" ICL2007 : EPortofolio and Quality in e-Learning*, 2007.

Martel C., Vignollet L., Ferraris C., « Modeling the Case Study with LDL and Implementing it with LDI », *IEEE ICALT*, p. 1158-1159, 2006a.

Martel C., Vignollet L., Ferraris C., « Une Ingénierie des Environnements Informatiques pour l'Apprentissage Humain basée sur un modèle de l'activité », *IDM 07*, 2007.

Martel C., Vignollet L., Ferraris C., David J., Lejeune A., « Modeling collaborative learning activities on e-learning platforms », *IEEE ICALT*, p. 707-709, 2006b.

Martel C., Vignollet L., Ferraris C., Durand G., « LDL : a Language to Model Collaborative Learning Activities », *EDMEDIA 2006, World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 2006c.

Martel C., Villiot-Leclercq E., Vignollet L., Despont A., Ferraris C., « Vers un outil d'édition de scénarios pédagogiques », *TICE 2008*, 2008.

Milligan C., Beauvoir P., Sharples P., « The Reload Learning Design Tools », *Journal of Interactive Media in Education*, 2005.

Paquette G., *L'ingénierie pédagogique : pour construire l'apprentissage en réseau*, Puq, 2002.

Rossignol A., *Le procédé La Martinière*, Imp. Fabregue, 1951.

Sottet J., Calvary G., Favre J., « Ingénierie de l'Interaction Homme-Machine Dirigée par les Modèles », *IDM'05 Premières Journées sur l'Ingénierie Dirigée par les Modèles*, 2005.

Villiot-Leclercq E., Modèle de soutien pour l'élaboration et la réutilisation de scénarios pédagogiques, PhD thesis, Université Joseph Fourier/Université de Montréal, 2007. 350 p. disponible sur http ://tel. archives-ouvertes. fr/tel-00156604.

Industrial-strength Rule Interoperability using Model Driven Engineering

Marcos Didonet Del Fabro* - Patrick Albert* - Jean Bézivin Frédéric Jouault****

**ILOG, an IBM Company*
9, rue de Verdun
94253 Gentilly Cedex
{mddfabro,albert}@ilog.fr

***AtlanMod (INRIA & EMN)*
4 rue Alfred Kastler
44307 Nantes Cedex 3
{jean.bezivin, frederic.jouault}@inria.fr

ABSTRACT. *Business Rule Management Systems (BRMS) aim at enabling business users automating their business policies. There are several BRMS supporting different languages and capabilities, and almost no available interoperability facility. So far, migration typically follows a manual procedure. In this paper, we present a generic approach based on Model Driven Engineering (MDE) techniques for bridging Business Rules languages; the solution has been fully implemented and tested on different industrial-strength BRMS. The development of such bridges presents many challenges, such as parsing sophisticated languages with different abstraction levels and supporting a large number of artefacts as input and as output. We describe the overall MDE architecture and components as well the as the issues we encountered. In addition, we present the lessons we learned on applying MDE techniques to an industrial project.*

KEY WORDS: *model transformations, business rules, BRMS, interoperability.*

1. Introduction

Business rules are mostly an evolution of production rules (Brownston *et al.* 1985) – an established A.I. technique – used to enable business users to automate policies. The rules are written in Domain Specific Declarative Languages as close as possible to the application domains. The technical details of the rules and application are hidden by a so-called "business-level" layer, enabling the policies owners to create and manage the rules by themselves, with little to no support from software developers – the IT bottleneck is skipped. Being largely independent for IT, the business users gain agility, while a set of tools and process support the rules lifecycle controlling the evolution of the application.

BRMS, or Business Rule Management Systems (Owen 2004), are the products enabling such behaviour. They are made of a number of tools supporting various activities such as rule authoring, validation, documentation, versioning and execution. Well-formalized processes guarantee that new or modified rules are well managed and have little to no risk of breaking the application logic.

A number of BRMS products compete on the market: ILOG JRules (JRules 2008), JBoss Drools (Drool 2008), Yasu QuickRules (Yasu 2008), and though they share common roots, all these systems have heterogeneous set of capabilities and specific languages. Consequently, it is hard to migrate from one system to another.

Rule Interchange Format (RIF) (RIF 2008) and Production Rules Representation (PRR) (PRR 2007) are two standard proposals respectively developed at the W3C and at the Object Management Group (OMG 2008) aiming at providing a level of standardization to the domain. But these necessary initiatives are either not ready or only covering a share of the BRMS languages, thus the interoperability problem is still a problem and will remain for a moment.

In this paper, we present a rule interoperability solution based on Model Driven Engineering (MDE) technologies. Our solution allows creating bridges between rule languages from different Business Rules Management products; it is currently applied to bridging ILOG JRules and JBoss Drools.

We define two bridges: a (reasonably) simple one from DRL – Drools Rule Language (Drools 2007) – to IRL – ILOG Rule Language (JRules 2008), and a much more complex bridge that takes IRL rules as input and generates Business Rules written in the ILOG "Business Action Language" (BAL) (ILOG Language 2008). Such bridges are a composition of several transformations.

Since the BAL contains expressions similar to natural-language, the bridge requires complementary models about the ontology, the terminology and the type of the expressions. The development of such bridges raises several challenges: mapping existing industrial languages with different abstraction levels and complexity; managing the presence of several models as input and as output; and eventually the packaged production of executable and ready-to-use projects.

In both cases, the resulting rules are successfully executed using the same objects sharing regular POJO (Plain Old Java Objects) definitions in Drools and JRules.

The major contributions of this paper are the following. We show how we applied a complex MDE architecture involving a large number of input or output models and transformations - no less than twenty - for developing bridges between two industrial BRMS. Then, we identify some hard issues when developing such bridges, and we explain the adopted solutions. We focus on the specification of the transformations and on the coordination of their execution. This scheme is based on the capture of the abstract syntax by metamodels, on the systematic use of chains of model transformations and on the mappings between concrete and abstract syntaxes.

This paper is organized as follows. Section 2 presents the general architecture of the components of two BRMS. Section 3 presents our rule interoperability solution.

Section 4 describes the applications and the lessons learned. Section 5 presents the related work. Section 6 concludes.

2. General architecture

A BRMS is composed of several components supporting the definition, management, and execution of business rules. We present business rules and the base concepts of BRMS. We then zoom into the two Business Rules Languages studied and the artifacts involved.

2.1. Rules

Rules are more and more used to help business organizations automating their policies, providing properties such as reliability, consistency, traceability and scalability. Rule languages are mostly made of *If-Then* statements (as shown in Figure 1) involving predicates and actions about sets of business objects. Rules variables are bound to objects of a certain type; when the condition part recognizes that a set of objects satisfies the condition predicates, the action part is triggered.

```
rule "approve"
if
  ?C:Customer.status = “Gold”
Then
  ?C.applyDiscount(20%)
end
```

Figure 1. *A simple Production Rule*

2.2. Business rules

Compared to earlier rules languages such as the seminal OPS5 (Forgy 1981) intended to be used by software developers or knowledge engineers, the business rules approach is an attempt to bring the power of rules programming to business users willing to automate business policies. A trained business is able to evolve the rules implementing its policies without having to ask for IT support.

The main difference is that the rules are expressed in a language close to natural language that can be understood and managed by “business analysts”. These Business-level languages are typically compiled into lower-level technical languages; the simple ‘production rules’ semantics remains unchanged.

The “Business Layer” is composed of additional models supporting the definition of the rules (see Figure 2).

- A Business Object Model (BOM) defines the classes representing the objects of the application domain. A BOM is in fact a simplified ontology defining classes with their associated attributes, predicates and actions.

- A Vocabulary model (VOC) defines the words or phrases used to reference the BOM entities.

- A Business Action Language (BAL) Rule, close to natural language, is typically an If-Then statement that composes the elements of the VOC to build user understandable rules, such as "**If** the age of a human is greater than 18 **Then** the human is major". This rule relates the attributes "age" – a Positive Integer – and Boolean attribute "major" of the class "Human".

The business object model and vocabulary are defined in an early stage of the project, and their projection on an executable layer – the Execution Object Model (XOM) and technical rule language – is implemented by a group of IT specialists. The technical-level layer is used to execute the business rules on the application data within the application architecture.

Figure 2 illustrates how the BOM is implemented by the XOM (typically a set of Java or C# classes), and how the BAL is compiled to the Technical Rules language. The technical rules are applied to the XOM objects.

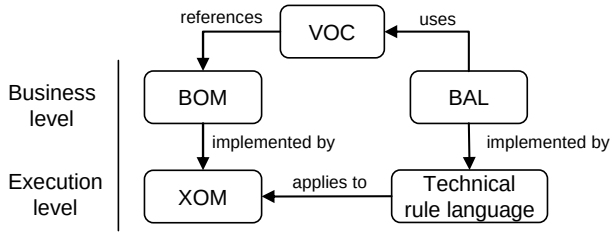


Figure. 2. *Different layers of production rules*

2.3. Business Rule Management Systems

A Business Rule Management System is composed of a large set of components dedicated to enable Business Users managing the whole life cycle of the business rules. We present two BRMS in the following sections, focusing on the rule languages supported by these systems.

2.4 Drools

Drools is an open-source BRMS part of the JBoss foundation. Its rules language is called DRL, for Drools Rule Language (Drools 2003). Though Drools has recently introduced some sort of macros for defining business friendly rules, we have focused our work on the technical rules language, as illustrated in Figure 3.

```
rule "approve"
when
   not Rejection()
   $policy : Policy( approved == false )
then
   System.out.println("approve: " + $policy.getPrice());
   $policy.setApproved(true);
end
```

Figure 3. *Example of rule in DRL*

2.5. ILOG JRules

JRules is the product developed and marketed by ILOG. It supports the two different levels of a full-fledged BRMS: the technical level targeted at software developers and the business action language targeted at business users.

2.5.1. Technical rules

The technical rules written in IRL – see Figure 4 – are directly executed by the rule engine. The complete specification of IRL might be found at (IRL 2008).

Consider the rule illustrated in Figure 4. The rule has the same semantics as the rule from Figure 3, and the rules are quite similar.

```
rule approve {
when {
  not Rejection();
  ?policy : Policy(approved == false);}
then {
  System.out.println("approve: " + ?policy.getPrice());
  ?policy.setApproved(true);
}
```

Figure 4. *Example of rule in IRL*

2.5.2. Business Action Language

The Business Action Language (BAL) hides implementation details, allowing business analyst concentrating on the business logic. The equivalent rule for the previous examples is shown in Figure 7.

```
definitions
  set 'policy' to a policy;
if
  there is no rejection and 'policy' is not approved
then
  print "approve: " + the insurance price of 'policy';
  make it true that 'policy' is approved;
```

Figure 5. *Example of a rule in BAL*

As we can see, as a BAL rule is composed of natural language fragments; it is close enough to natural language that any business analyst can read and modify it. The specification of BAL is found at (ILOG Rule Language 2008).

2.5.2.1. Business Object Model

The Business Object Model (BOM) contains all the concepts manipulated by the business rules; it describes the ontology of the application domain. It is mapped to a so-called execution object model (XOM) that will support the actual application data. In the simplest cases, the BOM can be automatically extracted from the classes definitions defined in Java, C# or XML-Schema. For instance, if the execution objects are Java objects, the business model will define one BOM *class* per Java class. We illustrate in Figure 6 the BOM model of the *Policy* concept.

```
public class Policy {
    public boolean approved;
    public double basePrice;
    public int discountPercent;
    public double insurancePrice;
    public Policy () ;
}
```

Figure 6. *BOM for the policy business object*

2.5.2.2. Vocabulary

The vocabulary (VOC) describes the mapping between the business concepts and the terminology used to write BAL rules. It defines a controlled vocabulary that "closes" the rules syntax on a fixed set of allowed words and fragments.

For instance, the attribute *insurancePrice* is mapped to *"insurance price of {this}"*. The *{this}* token indicates the calling object. An excerpt of the vocabulary of policy is shown in Figure 7. The verbalization of class *Policy* is *policy*.

This closed vocabulary is used by the business rules' editor to propose the possible inputs while a business user creates or modifies a rule.

```
Policy#concept.label = policy
Policy.approved#phrase.action =
      make it {approved} that {this} is approved
Policy.approved#phrase.navigation= {this} is approved
Policy.insurancePrice#phrase.action=
      set the insurance price of {this} to {insurance price}
Policy.insurancePrice#phrase.navigation= {insurance price} of {this}
```

Figure 7. *Vocabulary with the terminology of the Policy class*

2.5.2.3. B2X and project files

The "Business To eXecution" model (B2X) describes how the logical elements represented in the BOM are actually mapped to physical data structures. Although generic BOM mappings are defined towards Java, C# and XML, an application developer might add new B2X constructs to specify the way new BOM elements that have no corresponding XOM construct are implemented with the target programming language.

The B2X illustrated in Figure 8 contains a mapping for a virtual BOM member *InsertAction.* When the corresponding verbalization is used in a BAL rule, the engine calls the expression of the mapping (*insert (object)*).

```
    <method>
      <name>InsertAction</name>
      <parameter type = "java.lang.Object"/>
      <body language = "irl"><![CDATA[insert (object);]]> </body>
    </method>
```

Figure 8. *B2X mapping*

Eventually, in order to produce a ready to use ILOG JRules application, a JRules project has to be created. The *.ruleproject* (RP) file specifies the project parameters, the folder where the rules, the BOM and the vocabulary are stored, a URL, and the output folder. The BAL rules are wrapped in XML files (*.brl* files) that contain the rule properties, URL, folder info and the code of the rules.

3. Rule Interoperability

We present our solution that applies MDE techniques to achieve interoperability across BRMS. As already stated, the central concept in BRMS is the rule language. For that reason, the main goal is to transform a set of rules of a source BRMS, into an executable set of rules (and associated files) in a target BRMS. The

interoperability between BRMS is intended to ease the comparison of rule engines, and to automate the migration from one product to another.

In our scenario, we have two complementary objectives: translating a set of rules available in DRL into IRL (interoperability between technical languages) and translating IRL rules into BAL rules (interoperability between a technical and a business language). We first produce IRL files from DRL files, and then we produce the BAL-level project from the IRL files. In order to have a ready-to-run JRules application, it is also necessary to produce the vocabulary, BOM and project files.

We present below the general architecture. Then, we describe the different steps of the bridge, the challenges encountered, and how we handle them.

3.1. General architecture

The architecture follows the usual MDE pattern: **inject**, **transform, extract**, and relies on four core MDE practices and technologies: Domain Specific Languages (DSLs) (Kurtev *et al.* 2006), metamodeling (Kurtev *et al.* 2006), model transformations (Jouault *et al. 2005*) and projections across technical spaces (i.e., injections and extractions) (Kurteve *et al.* 2002). DSLs and metamodeling are used to design adapted rule languages; model transformations are used to produce a set of output models from a set of input models; injections allows translating the input rules (and related artifacts) into models (e.g., text-to-model translation); and extractions allows translating models into the output artifacts (e.g., model-to-text translation). Though the current implementation only supports Drools and JRules, the architecture is modularized in order to easily accept other rules languages.

We rely on the tools developed at the AtlanMod Inria project:

- TCS (Textual Concrete Syntax) (Jouault *et al.* 2006). TCS is bidirectional, i.e., it provides text-to-model and model-to-text translations.

- KM3 (Jouault *et al.* 2006), a simple metamodel specification language, is used to define the metamodels.

- ATL (Jouault *et al.* 2006) is used to implement the many model transformations. ATL has a development environment integrated into Eclipse (EMP 2008), such as textual editor, and debugger.

- AM3 (Allilaire *et al.* 2006), the model management scripting environment, enabling us to execute chains of “load, transform and save” units.

3.2. DRL to IRL

This bridge produces an ILOG JRules project including rules written in the IRL language from a set of files written in DRL.

3.2.1. KM3 and TCS definition

The KM3 metamodels are based on the online specification of Drools and JRules (respectively expressed as XSD schemas and EBNF notation). For the sake of maintainability, we could have designed more clever KM3 models, we have chosen to create model elements sticking to the specifications.

Still, in the Drools case, we had to refactor the KM3 to take into account the nested nature of XSD schemas. For instance, <*choice*> elements in the XML schemas have been transposed into trees in KM3. In both cases The TCS models directly map the KM3 metamodels to the syntax.

3.2.2. Transformation specification

A transformation rule, expressed in ATL, has the following signature: `create OUT : IRL from IN : DRL;` It takes one DRL model as input and produces one IRL model as output. The transformation is relatively simple, since both languages are similar. There are a few DRL expressions that are not natively supported by IRL (the opposite is also true, but in this paper we concentrate on one direction). For example, DRL conditions might be connected by an *OR* predicate – such as in "`not Rejection() OR Policy(approved == false)`" – which is not possible in IRL. To transpose the "OR" semantics, we first run an endogenous refactoring transformation that transforms every *OR* predicate into two rules, with the same conclusions.

3.2.3 Chaining and parameterization

A model management script chains the different operations, executing the whole bridge in one automated step; it executes the following sequence of actions:

1. **Injection:** DRL textual syntax into DRL models.
2. **Refactoring:** DRL models are transformed into new DRL models
3. **Transformation:** the DRL models are transformed into the IRL models.
4. **Extraction:** IRL models are projected in the IRL textual syntax.

3.3. IRL to BAL

The central transformation produces an ILOG JRules Project including rules written in the BAL syntax. As this bridge is far more complex than the former – it

includes additional input and output models, such as the BOM, the VOC and the B2X – it has been designed as a composition of a large number of transformations.

3.3.1. KM3 and TCS definition

We have created five KM3 metamodels, for the BOM, VOC, BAL, B2X and project files, and three TCS schemas, because only the BOM, VOC and BAL have textual syntax. Though they lack a well documented formal specification, the BOM and VOC models are relatively simple; we have been able to create the metamodels by testing successive alternatives in the editor provided by JRules.

As BAL is not context-free, and TCS only supports context-free expressions, the creation of the TCS for BAL has been quite challenging. A BAL rules is essentially a composition of small fragments of free text. It is thus possible in BAL to write literally any kind of expression with very few restrictions. In order to use TCS, we support a large subset of BAL and its associated KM3 metamodel. Consider the two expressions below:

```
set '<variable>' to <value>
set the insurance price to 300,00
```

The first expression defines a variable, while the second assigns a value to a field. To make thinks more complex, any kind of terminology is supported by JRules, such as *set's* and *to's* within sets, commas, *if*s, etc.

Another challenge is the specification of operators. The operators are textual operators, such as "*is bigger than"*, "*is smaller than"*, and they may be composed as in: "*is bigger than* X *and smaller than* Y". To be able to parse these multi-word operators, we have created a class for each: *IsBiggerThan*, *IsSmallerThan*, etc. Though this approach proved to be practical, is has required writing much more code on the IRL to BAL transformation, and requires more modifications to create a TCS for another language (e.g., French).

3.3.2. BOM, VOC, B2X and RP refactoring

The JRules environment can automatically generate the initial BOM and the default vocabulary from the Java or C# XOM classes. However, as IRL provides more constructs than BAL, as for example *insert* or *retract* actions and *for*, *while*, *if* and *try-catch* statements, we had to add specific vocabulary to express them at the BAL level, and specific translations from BOM to XOM.

The solution has been to generate B2X mappings as needed for all non-supported expressions occurring in the rules. This has translated into the definition of three elementary transformations:

- **Extending** the BOM to add virtual methods, for instance an *InsertAction* method;

- **Extending** the VOC to add verbalizations, for instance "*Policy.InsertAction = add policy*;"

- **Extending** the initial B2X mapping to add the new mapping expressions.

3.3.3. Transformation specification

The IRL2BAL transformation has different issues. The first one is to find the correct verbalization (terminology) for every IRL expression. The transformation must search the corresponding expressions in the BOM and in the vocabulary. We create a set of ATL helpers that navigate through the IRL expressions from left to right, and produce an expression from right to left. This means, for example, that *policyInsurance.cost* is translated as *"the cost of the insurance policy"*. The vocabulary is navigated recursively to find the correct verbalizations.

BAL has also pre-defined patterns that are not in the BOM. Operations over *Strings* are the most common patterns. For instance, expressions such as *str1.indexOf("str") != -1* must be transformed into (*str1 contains "str"*). We had to do an inventory of all the patterns supported by the BAL and to implement them in the transformation rules.

This transformation has different complex issues, and it deserves itself a separated study. However, in this paper we focus on the overall complexity of the transformations and metamodels.

3.3.4. Project files

A rule project in JRules has additional files storing project-relative properties. These files are not at the core of the bridge, but they are necessary to obtain a ready-to-use project. For instance, we define a helper transformation that wraps the BAL rules into an XML model (BRL), which is later extracted into an XML file that is directly used by JRules.

4. Application

As shown in Figure 9, the solution has reached a level of complexity in terms of the number and types of operations performed. It may be considered as a significant deployment of an MDE application in a real environment to solve a practical problem. As a side consideration, we can observe here the need to get tools to handle complex networks of transformations. The inventory of the transformations required, as well as their correct chaining is one of the central challenges for producing such a bridge.

The bridge described executes *twenty four* operations, some of which are complex in nature and involve a large number of metamodel classes, grammars or transformation rules. They can be categorized as follows:

- **Generation:** of BOM and VOC by JRules from the Java Classes definitions.

- **Injection**: input files for DRL, BOM, VOC and optionally IRL.
- **Transformations**:
  - **Refactoring**: endogenous model transformation for DRL and IRL
  - **Augmentation:** growing endogenous model transformation for BOM, VOC, B2X and Rule Project.
  - **Traduction**: exogenous model transformation for DRL2IRL, IRL2BAL and BAL2BRL.
  - **XML-ification:** a special kind of exogenous model transformation toward XML, for B2X, BRL and Rule Project.
- **Extraction**: toward ad-hoc files for IRL, BOM, VOC and BAL, and toward XML for B2X, BRL and Rule Project.

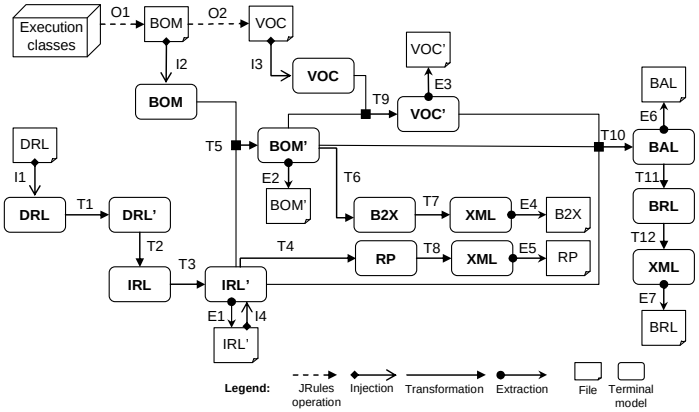


Figure 9. *DRL → IRL → BAL complete process*

Table 1 below provides some quantitative information about some of the metamodels, TCS models and transformations.

	TCS size	**KM3 size**	**Dev. time**
BOM	21 templates	21 classes	3 days
VOC	16 templates	18 classes	2 days
DRL	98 templates	102 classes	1.5 weeks
IRL	179 templates	185 classes	1.5 weeks
BAL	191 templates	194 classes	4 weeks

Table 1. *Size of KM3 and TCS of the main DSLs*

Table 2 below provides the size of the two main transformations, one refactoring and two augmentation transformations.

	Transformation size	**Dev. time**
DRL2IRL	1005 lines, 61 rules	2 weeks
DRLRefactor	1412 lines, 81 rules	
IRL2BAL	2396 lines, 90 rules	3 weeks
BOM augmentation	677 lines, 32 rules	2 weeks
VOC augmentation	778 lines, 31 rules	1 week

Table 2. *Size of the transformations*

Both bridges have been tested on the two standard business rules benchmarks: *manners* and *waltz* (Academic Benchmarks 2007), on an "insurance claim" demonstration (Drools Examples 2008), and on the translation of a banking application with more than 100 rules.

The rules are executed over the Java objects provided with the examples. We have run the *manners* benchmark with settings for 16, 32, 64, 128, 256 and 512 objects. In all cases, the results are the same, and the same number of rules is fired in the three implementations: DRL, IRL and BAL. The correctness of the transformations are empirically validated (by extensive testing), we did not apply any formal validation technique. The generated rules correctly preserve the operational semantics of their original models.

4.1. Lessons learned

Following this experimentation, we conclude that MDE tools have reached a reasonable level of maturity allowing their use in the context of industrial projects. They show good performances, little bugs, and are available as Eclipse open source projects. However, installing and using these tools may be rather complicated for a non specialist. The major advantage of MDE is the possibility to concentrate on the problem specification and apply a declarative and modular approach using a small set of principles and tools.

One of the major conclusions is that the correct chaining of transformations is a key factor of success, because the bridge must be easy to configure and to run, acting over several input and output models. A transformation cannot be fired until all its parameters are loaded. Consequently, the dependency relations shown in the schema of Figure 9 must be respected during the execution.

Coupling the scripts and of the parameterization of the execution has proved very important. However, an integrated repository, using graphical interfaces integrated within Eclipse would help a lot. We can observe here the need to get tools to handle complex networks of transformations. This is a field where MDE has still to provide new solutions.

KM3 enables the definition of the metamodels in a practical way. On the syntax side, thought we could eventually reach our objective, we found more difficulties with TCS, because of its context-free limitation. Such a limitation has introduced an unwanted level of complexity in our models. This means we have to do compromises between supporting all the expressions and the complexity of the development.

Declarative model transformations are a concise and practical approach. The ATL transformations are particularly elegant in the DRL to IRL bridge, because the languages have similar semantics. However, the metamodels have specific details that have required considerable development time.

As far as we know, the bridge requiring such a large number of transformation expressions have not been deeply explored before. Some of them -- such as expression patterns – are generic enough to be used in other solutions, with just minor adaptations. However, even by using declarative rules, the transformations produced are still large. Current solutions still need to improve their modularization capabilities. For instance, we would like to have transformations and libraries separated by packages, and with easy ways to navigate through the transformation code.

5. Related work

There are several BRMS in the market, such as ILOG JRules, Drools, Haley, Yasu QuickRules, etc. We are not aware of any industrial solution (at least publicly available) of bridges between different BRMS, and especially using advanced MDE techniques. A first work before this one has been the translation of PRR into IRL (Abouzhara *et al.* 2007). Though our project has a much greater size and complexity, the former has provided us with inspiration and reasonable hope for convergence.

Model transformations (and related concepts) are widely used in solutions of different degree of complexity. Several uses cases of model transformations may be found at (ATL Use cases 2008). There are several other tool suites similar to AMMA (e.g., Epsilon (Kolovos 2008) or oAW (oAW 2009)). The closest one is oAW, which is also available on the Eclipse.org platform. AMMA and oAW share the same model-centric vision and only differ on some implementation choices.

6. Conclusions

In this paper, we have presented a practical approach to provide interoperability between BRMS. The utilization of MDE techniques enabled to successfully develop two bridges amongst rule languages with different degrees of expressiveness.

To the best of our knowledge, this is the first approach implementing a solution for transformations with such a large number of transformations. The discovery, development and chaining of complex transformations has been a challenging task. The presence of several transformations, models and metamodels showed that megamodeling is a crucial issue when dealing with large projects. The scripting language is a first step that helped a lot, together with its parameterization. However, care should be taken to control the inherent complexity of the solution. We think more work can still be done on that area, especially in the specification of generic megamodeling platforms.

On top of their expected practical usefulness, implementing these bridges has revealed both the strength and some limits of available MDE tools. We have greatly appreciated the power of metamodeling associated with declarative programming and have suffered from the main limit relative to parsing of non context-free grammars. We have also faced different transformation problems that have not been explicitly handled in the literature (e.g., the navigation through patterns of expressions).

Our approach has been empirically validated by executing the bridge on a set of well-known benchmarks for business rules, and on a demonstrative example. The bridges produced the expected results.

There are several possibilities for future work, such as the creation of similar bridges amongst different rule languages (e.g., Yasu or Haley). The parsing can be internationalized into different languages, which is a common requirement of industry. Finally, we plan to study how to use MDE techniques to parse context-aware grammars, or even natural language.

Acknowledgements: This work has been partially supported by French ANR IdM++ project.

7. References

Abouzhara A, Barbero M. Implementing two business rule languages: PRR and IRL. Ref. site: http://www.eclipse.org/m2m/atl/usecases/PRR2IRL/. March07

Academic Benchmark performances. Ref. site: http://blogs.ilog.com/brms/2007/10/22/academic-benchmark-performance/, 27-12-2007

Allilaire F, Bézivin J, Brunelière H, Jouault F. Global Model Management. In proc. of eTX Workshop at the ECOOP 2006, Nantes, France

ATL Use Cases. Ref. site: http://www.eclipse.org/m2m/atl/usecases/. April 08

Brownston L, Farrell R, Kant E. Programming Expert Systems in OPS5 Reading, Massachusetts: Addison-Wesley, (1985)

Drools Examples. Ref. site: http://download.jboss.org/drools/release/4.0.4.17825.GA/ drools-4.0.4-examples.zip. 31-03-2008

Eclipse Modeling Project. Ref. site: http://www.eclipse.org/modeling/. April 08

Forgy C. OPS5 User's Manual, Technical Report CMU-CS-81-135 (Carnegie Mellon University), 1981

ILOG Rule Languages. Ref. site: http://www.ilog.com/products/jrules/documentation/jrules67/. April 08

ILOG JRules. Ref. site: http://www.ilog.com/products/jrules/index.cfm. April 08

JBoss Drools. Ref. site: http://www.jboss.org/drools/. April 08

Jouault F, Bézivin J. KM3: a DSL for Metamodel Specification. In proc. of 8th FMOODS, LNCS 4037, Bologna, Italy, 2006, pp 171-185

Jouault F, Bézivin J, Kurtev I. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In proc. of GPCE'06, Portland, Oregon, USA, pp 249-254

Jouault F, Kurtev I. Transforming Models with ATL. In proc. of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica, pp 128-138

Kolovos D, Paige R, Polack F. A Framework for Composing Modular and Interoperable Model Management Tasks, In proc. of Workshop on Model Driven Tool and Process Integration (MDTPI), EC-MDA 2008, Berlin, Germany

Kurtev I, Bézivin J, Aksit M. Technological Spaces: An Initial Appraisal. In proc. of CoopIS, DOA'2002 Federated Conferences, Industrial track, 2002, Irvine, California, USA

Kurtev I, Bézivin J, Jouault F, Valduriez P. Model-based DSL Frameworks. In proc of. Companion of OOPSLA 2006, October 22-26, 2006, Portland, OR, USA, pp 602-616. 2006

OpenArchitectureWare (oAW). Ref. site: http://www.openarchitectureware.com/. Jan. 2009

OMG. Object Management Group. Ref. site: http://www.omg.org. April 08

Owen J. Business rules management systems. Extract business rules from applications, and business analysts can make changes without IT breaking a sweat. Infoworld, 25-06-2004

Production Rule Representation (PRR), Beta 1, Document Number: dtc/2007-11-04 Ref. site: http://www.omg.org/spec/PRR/1.0/

Rule Interchange Format (Working Group). Ref. site: http://www.w3.org/2005/rules/wiki/RIF_Working_Group. April 08

Yasu Technologies. Ref. site : http://www.yasutech.com/. April 08

Model-based DSL Frameworks: A Simple Graphical Telecommunications Specific Modeling Language

Vanea Chiprianov[*,] — Yvon Kermarrec[*,**]**

[*] *Institut Télécom, Télécom Bretagne, UMR CNRS 3192 Lab-STICC*
Technopôle Brest Iroise CS 83818 29238 Brest Cedex 3, France
{vanea.chiprianov, yvon.kermarrec}@telecom-bretagne.eu

[**] *Université européenne de Bretagne, France*

ABSTRACT. *Since 1989, when it was first formulated, the problem of service and/or service feature interaction in Telecommunications has been widely addressed. We investigate the use of Model Driven Engineering (MDE) to provide an approach for the design of large-scale distributed systems that enables validation of system properties (e.g.; absence of undesirable service interactions). The first step to this design method is the definition of a modeling language which enables formal definition of services and their interactions. We describe here the experience we gained with MDE through the definition of a simple graphical Telecommunications specific modeling language (SGTSML), developed in the context of a master thesis.*

RÉSUMÉ. *Depuis 1989, date de sa première formulation, le problème de l'interaction entre les services et/ou les caractéristiques des services dans les télécommunications a été largement abordé. Nous étudions l'emploi de l'Ingénierie Dirigée par les Modèles (IDM) dans l'élaboration d'une approche de conception de systèmes distribues complexes et qui permet la validation de propriétés (e.g.; absence d'interactions indésirables entre les services). La première étape de cette méthode de conception est la définition d'un langage de modélisation pour la spécification formelle des services et de leurs interactions. Nous décrivons ici l'expérience acquise en IDM par la définition d'un langage graphique simple de modélisation pour les télécommunications, développé au cours d'un stage de master.*

KEYWORDS: *MDE, DSL, Telecommunications, model-based DSL frameworks.*

MOTS-CLÉS : *IDM, DSL, Télécommunications, frameworks dirigés par les modèles pour les DSL*

1. Introduction

Today's Telecommunications systems are very large and complex. They are constituted of services and service features. Between different services and/or service features there are numerous interactions. Because of them, the introduction of a new service poses a great problem, as it impacts the services with which it will interact, both directly and indirectly. Moreover, undesired interactions may appear. This constitutes the "Feature Interaction Problem", first described by (BOW 89). Many solutions that tackle this problem have been proposed. A classification of them is proposed by (KEC 98). Following this classification, we investigate the use of MDE (briefly presented in section 2) in the devise of a preventive, structural approach for the design of large-scale distributed systems. This approach enables validation of system properties (e.g.; undesirable service feature interactions) and provides a means to clearly specify the system structure. This means is a model driven, design oriented method, for the services and/or service features.

Our work takes place in the context of an applied research project in the domain of Telecommunications. This project aims at bridging the communication gaps between the different domain experts involved in the top-to-bottom process of defining a new service, and concentrates on team work, reuse and multi-vendor support (i.e.; integration of information coming from different hardware platform providers).

The model driven design method we investigate proposes to service designers a Domain Specific Language (DSL) that enables them to define the domain specific entities they will use to specify a service. These entities are implemented as models, which makes this DSL a modeling language. This language will have a static model view (like UML class diagrams), a process model view (like BPEL) and a constraints model view (like OCL). The development of a DSL is a complex task, following several phases, as presented in section 3. In this work, we concentrate on the static model view and investigate the use of model-based technologies for the definition of a language for this view. For our investigation to be pertinent, we apply the process of definition of a DSL using MDE technologies (section 4) in the context of Telecommunications. We start with a simplified domain description, the result being the language described in section 5. The language produced in this work could be envisioned as a DSL embedded (MER 05) in the future modeling language. We capitalize on the lessons learned from this approach (section 6) and indicate the future work that needs to be done.

2. Model Driven Engineering

MDE is a software engineering approach concerned with bridging the conceptual gap between the problem and implementation domains by using as primary artifacts of development abstract models that describe the system from multiple viewpoints and by providing automated support for analyzing models and transforming them to

concrete implementations. The basic principle of MDE is "everything is a model" (BEZ 04).

"*A model represents reality for the given purpose; the model is an **abstraction** of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a **simplified** manner, avoiding the complexity, danger and irreversibility of reality.*" (ROT 89)

MDE main challenges have been recently surveyed by (FRA 07). In preparation for these challenges:

– We decided for our project that the **modeling language** will be a DSL (as opposed to an extensible general purpose modeling language like UML, because we need as much expressive power as possible) and the formalism used will be the revisited MDA organization, presented by (BEZ 01).

– We address the issue of **separation of concerns** by using a fixed set of viewpoints as a starting point, because it is a simple approach, and, after the concern specific viewpoints will have been identified, we will switch to using them.

– In what concerns the **model manipulation and management**, we will begin by interesting ourselves into abstraction and refactoring of the platform model.

3. Domain Specific Languages

"*A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, **expressive power** focused on, and usually restricted to, a **particular problem domain**.*"(DEU 00)

The DSL development methodology has been extensively presented by (MER 05). It consists of five phases, for each of which several choices are possible:

– In the context of our project, the **decision** to implement a DSL was quite straightforward, as there is a need of domain-specific and domain-level analysis, verification, optimization and transformation, the services can be modeled as families of products and there is a need for complex data structure representation and access.

– For the **analysis** of the Telecommunications services domain the method that seems to be the most promising and worthy of investigation is Family-Oriented Abstractions, Specification, and Translation (FAST) (COP 98), as it applies the product line architecture principles and, as stated by (DEU 02), "*one can also view a DSL as a means to describe and generate members of a family of programs in the domain*".

– In what concerns the **design** of the modeling language for our project, we chose to investigate and propose a new language that responds exactly to the needs of service designers, as opposed to exploit an already existing language. We also chose to specify this design informally, following time constraints related to rapid prototyping.

– We decided to investigate the model-based DSL frameworks approach (presented in section 4) for the **implementation** phase, the results of this investigation being presented in this document.

– Like for any new programing language, we expect to have **deployment** issues related to the degree of adaptation to the users, the adoption by the users, the maturity and completeness of the development environment, the expressive power of the DSL.

4. Model-based DSL frameworks

On one hand, being a programming language, the definition process of a DSL has to be consistent with the definition process of any programming language. Consequently, an abstract syntax (AS), a concrete syntax (CS) and a semantics have to be defined. On the other hand, (KUR 06) consider that "*a DSL is a set of coordinated models*". The two points of view are not at all contradictory. The AS of a DSL can be modeled as the Domain Definition Meta-Model (DDMM), the CS can be represented as a "display surface" meta-model and the semantics can be obtained through a transformation model between the DDMM and, either the meta-model (MM) of a DSL with a precise execution, or the MM of a general purpose programming language (GPL) (e.g.; Smalltalk). In this way, a DSL is implemented using model driven technologies.

Besides model driven technologies, model-based tools are needed for the definition of a DSL. TOPCASED (FAR 06) is an integrated system engineering toolkit, based on Eclipse, for critical and embedded applications. It is strongly model oriented, providing model editors, checkers, transformations and a tool to automatically generate graphical editors for DSLs based on their MM. Consequently, *TOPCASED can be used as an AS and a concrete (graphical) syntax definition tool*. OpenArchitectureWare (OAW) (FEA 07) is a modular MDE generator framework implemented in Java and based on Eclipse. It supports a language family to check and transform models as well as generate code based on them. Consequently, *OAW can be used as a tool to define the execution semantics*. We chose to use TOPCASED and OAW because they have some of the most stable implementations of MDE concepts and template based code generation, respectively.

In our work, we apply the methodology for DSL implementation using MDE presented by (KUR 06), but with different tools, to a different problem, in a different context, as presented in the next section.

5. Simple graphical Telecommunications specific modeling language (SGTSML)

Using the process presented in section 4, we defined a simple DSL, with graphical syntax, for the modeling of Telecommunications. We started with the definition of the **abstract syntax**, presented in figure 1. Because we intend to use an iterative approach for the definition of our language, this first DDMM is a very simple one, focusing

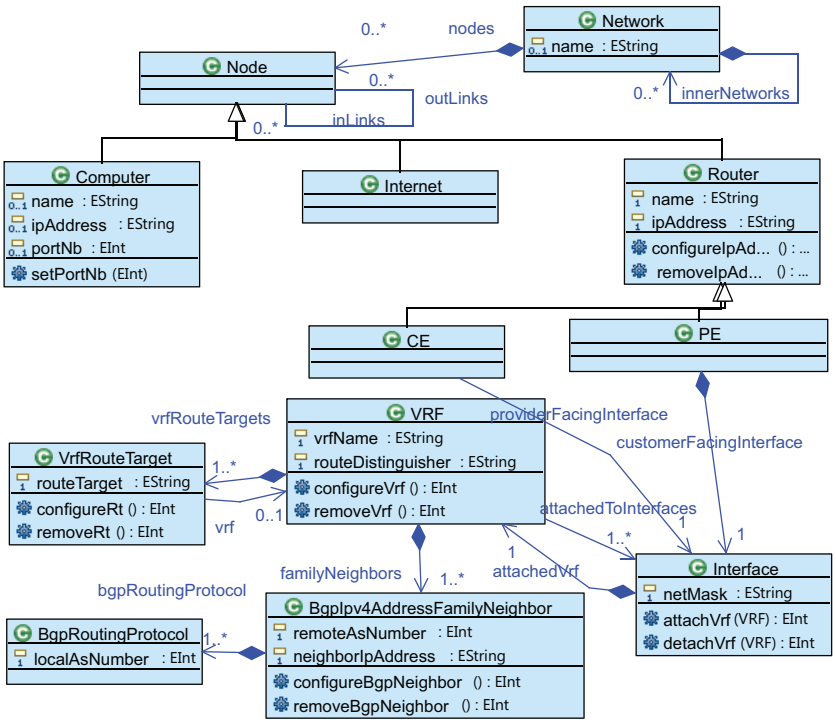


Figure 1: Abstract syntax of SGTSML

on the virtual private network (VPN) specifics. It consists of a *Network*, which may contain several inner networks and several *Node*s. The nodes are either *Computer*s, *Internet* or *Router*s; they are connected by links which constitute for the source nodes outlinks, and for the target nodes inlinks. The routers can be either customer edge routers (*CE*) or provider edge routers (*PE*). Each PE and CE has an *Interface*, which contains a virtual routing and forwarding (*VRF*) table containing the *VrfRouteTargets* and information about the neighboring PEs (*BgpIpv4AddressFamilyNeighbor*s). PEs use the Border Gateway Protocol (*BgpRoutingProtocol*) to communicate with each other. It is also possible to enrich the DDMM with validation rules, such as that presented in listing 1, which validates that the IP address is different for all PEs, thus enabling domain level validation.

The next step is the definition of the **concrete syntax**, presented in figure 2. We represent here a VPN with three *PE*s, interconnected by *BgpRoutingProtocol* green links. Each *PE* is connected with a *CE* from each of the three customer networks, which also include another two regular *Computer*s each. The fact that we use the same icons for *CE*s, *PE*s and *Computer*s is due to the difficulty of defining custom

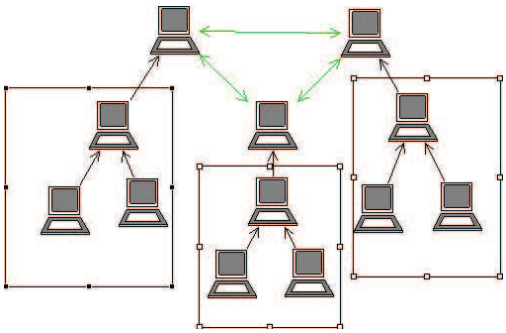


Figure 2: Concrete syntax of SGTSML

```
context PE
inv: self.attachedVrf.familyNeighbors->forAll(
    n: BgpIpv4AddressFamilyNeighbor |
    n.neighborIpAddress = self.ipAddress)
```

Listing 1: OCL rules

icons with TOPCASED. We even tried to develop the extension points, as indicated in documentation, but did not succeed, which takes us to the conclusion that defining custom icons in TOPCASED is not simple for the regular user. Hopefully, this current limitation will soon be overcome. We decided that our language should have a graphical syntax, because its intended users are service designers, who are familiar with representing services as drawings. To make the connection between the AS and the CS we define a transformation model in the form of a direct mapping. For the definition of the AS, the CS and the mapping between the two, we used TOPCASED.

The last step consists in the definition of **semantics**. We chose to define a transformation model between the AS of our SGTSML and the MM of a GPL. We decided to implement this transformation using templates, because it is a mature and well-documented technique. We used OAW as a tool for defining templates and chose Smalltalk as the GPL because we needed an object oriented language.

The modeling language for our project, when mature, due to its use of domain specific concepts, will be easy to use, thus increasing the performance of service designers, and will constitute a good means of domain knowledge conservation and reuse. The use of model driven techniques and tools enabled us to rapidly design and implement our SGTSML and moreover allows us to iteratively evolve and maintain it without any additional cost. The ability to iteratively develop the language definition (i.e.; the DDMM) enables a closer cooperation with the future users (i.e.; service designers), which increases the chances of a good acceptance of the language in the community.

Even with all the help of model-based tools, the definition of our SGTSML remains difficult, because it requires knowledge about several domains: Telecommunications, programming languages and MDE. The use of model-based tools introduces the need for MDE specific knowledge, but greatly reduces the knowledge needed about programming languages (e.g.; no need for compiler techniques). Because the semantics of our SGTSML is implemented using automated translation towards Smalltalk, the quality and performance of the generated code remain important issues.

6. Lessons learned

We discovered that, despite its young age, MDE already provides *good tools*, which allow for complete, flexible, iterative solutions, mainly due to the power of the 3-level model paradigm. The fact that many of these tools are open source is also very useful, because we can include them in our future environment. However, there are some aspects in these tools which are not as straightforward as they should be (e.g.; the definition of the mapping between the AS and the CS with TOPCASED is quite laborious and difficult to understand). A good point is the possibility to define a *graphical syntax* for one's language without doing any extra miles. In what concerns the capabilities TOPCASED has for graphical syntax definition, it would be useful to exist a way to parametrize the way links between entities are represented.

We found that approaching the concept of model-based DSL frameworks only from a theoretical point of view is quite difficult, as it implies a paradigm shift towards models, with all the attached new concepts and unavoidable learning curve, and a deep understanding of DSLs' purpose and scope. However, experimenting with existing tools made it easier to understand and finally we have found that the process is quite *simple to apply to a real world problem*. The model-based tools enabled us to easily define and implement our SGTSML. This allowed us to *rapidly prototype*, giving us the opportunity to consult with service designers early in the design process and use their help in the development of the DDMM. In this manner, we also *involve the future users* in the definition process, which will hopefully provide us with a better acceptance rate. Consequently, we can say that *the the concept of model-based DSL frameworks allows an iterative, incremental, gradual definition process.*

7. Conclusion and future work

In this work, we presented our experience in using MDE technologies and tools to implement a simple graphical modeling DSL for Telecommunications. We found that it is actually easy to use the approach of model-based DSL frameworks. We started with a simplified DDMM and concentrated on the process of using model driven technologies and tools to implement our SGTSML. In the future we will develop the DDMM incrementally and investigate the use of model transformations between the MM of our SGTSML and the MM of an existing GPL for the implementation of the

semantics of our SGTSML, the final purpose being to use the experience gained in defining our SGTSML to define the modeling language for our project.

Acknowledgements

This work has been performed as a master project and Vanea would like to thank the entire team for their valuable contribution.

References

[BEZ 01] BEZIVIN J., "From Object Composition to Model Transformation with the MDA", *TOOLS '01*, 2001, Page 350.

[BEZ 04] BEZIVIN J., "In Search of a Basic Principle for Model Driven Engineering", *Novatica Journal*, vol. 2, 2004, p. 21-24.

[BOW 89] BOWEN T., DWORACK F., CHOW C., GRIFFETH N., HERMAN G., LIN Y., "The feature interaction problem in telecommunications systems", *SETSS*, 1989.

[COP 98] COPLIEN J., HOFFMAN D., WEISS D., "Commonality and Variability in Software Engineering", *IEEE Softw.*, vol. 15, 1998, p. 37–45.

[DEU 00] DEURSEN A. V., KLINT P., VISSER J., "Domain-specific languages: an annotated bibliography", *SIGPLAN Not.*, vol. 35, 2000, p. 26–36.

[DEU 02] DEURSEN A., KLINT P., "Domain-specific language design requires feature descriptions", *J. of Comp. and Inf. Tech.*, vol. 10, 2002, Page 2002.

[FAR 06] FARAIL P., GAUFILLET P., CANALS A., LE CAMUS C., SCIAMMA D., MICHEL P., CREGUT X., PANTEL M., "The TOPCASED project: a Toolkit in Open source for Critical Aeronautic Systems Design", *ERTS*, 2006.

[FEA 07] FEATURES C., "openArchitectureWare 4.2", report , 2007, Eclipse.

[FRA 07] FRANCE R., RUMPE B., "Model-driven Development of Complex Software: A Research Roadmap", *FOSE '07*, 2007, p. 37–54.

[KEC 98] KECK D., KUEHN P., "The feature and service interaction problem in telecommunications systems: a survey", *Soft. Eng., IEEE Trans. on*, 1998.

[KUR 06] KURTEV I., BEZIVIN J., JOUAULT F., VALDURIEZ P., "Model-based DSL frameworks", *OOPSLA '06:*, 2006, p. 602–616.

[MER 05] MERNIK M., HEERING J., SLOANE A. M., "When and how to develop domain-specific languages", *ACM Comput. Surv.*, vol. 37, 2005, p. 316-344.

[ROT 89] ROTHENBERG J., "The Nature of Modeling", *Artificial Intelligence, Simulation, and Modeling*, 1989.

Contrôle guidé par l'IDM des évolutions d'un système de recherche d'informations clinique

Valéry Lopes, Éric Leclercq, Marie-Noëlle Terrasse
Laboratoire LE2I - UMR CNRS 5158
Université de Bourgogne,
21000 Dijon
prénom.nom@u-bourgogne.fr

Résumé. Dans le contexte des essais cliniques, un travail conséquent d'expérimentation est réalisé sur la base de protocoles de recherche définis avec précision. En effet, un essai clinique ne peut aboutir qu'après la validation successive de chacune de ses phases. L'échec d'une phase peut donc conduire à l'évolution de tout ou partie de l'essai clinique. Du point de vue du système d'information, les données, les traitements sur les données, les applications de manipulation ainsi que les *workflows* reflétant les protocoles d'études sont soumis à de fréquentes évolutions. Par conséquent, une gestion fine des développements est nécessaire pour maîtriser les évolutions et maintenir l'accès aux données acquises.

L'approche que nous proposons pour le contrôle de l'évolution vise non seulement à maintenir un accès aux données acquises, mais également à garder la trace des évolutions de modèles au niveau *workflows*, données et applications. Notre démarche repose sur trois éléments clés :

1) la définition d'un ensemble de modèles de haut niveau : ces modèles suivent pour les uns un point de vue centré métier et pour les autres un point de vue centré données.

2) la définition de points d'extension au niveau de ces modèles : cette notion de points d'extension est basée sur l'hypothèse que dans un domaine donné certains éléments des modèles centrés métier concentrent les évolutions.

3) la définition de *mappings* dédiés au maintien de l'accès aux données : les *mappings* sont définis à partir de modèles centrés données et visent à assurer la flexibilité nécessaire par rapport à l'évolution des données.

Nous avons développé un prototype de systèmes de recherche d'informations spécialisé pour les données cliniques, avec le soutien des partenaires de ce projet (la Fondation Clément-Drevon, la Fondation Transplantation et la société FORENAP). Nous utilisons les technologies J2EE et la bibliothèque Apache Lucene à laquelle nous avons ajouté des modèles de méta-données pour unifier les requêtes sur les différentes sources d'information.

Cette approche uniformisée, centrée modèle, et basée sur des transformations, nous permet d'envisager d'une part un canevas de transformations métier et d'autre part des transformations opérationnelles sous forme de *mapping* au niveau données. De plus, cette approche assure la persistance d'un lien traçable (aux niveaux données, applications et sémantique) entre ces modèles.

Index des auteurs

Albert, Patrick , 163
Alff, Patrick , 179
Bernard, Alain , 101
Bézivin, Jean , 33, 163
Cancila, Daniela , 73
Charfi, Anis, 17
Chiprianov, Vanea , 179
Cointe, Pierre , 33
Dupuy-Chessa, Sophie , 109
Ferraris, Christine , 147
Front, Agnès , 109
Garcés, Kelly , 33
Hamid, Brahim , 65
Joffroy, Cedric , 131
Jouault, Frédéric , 163
Kermarrec, Yvon , 179
Martel, Christian , 147
Mévellec, Pierre , 101
Occello, Audrey , 131
Ouari, Salim , 147
Renevier, Philippe , 131
Rieu, Dominique , 109
Vignollet, Laurence , 147
Woods, Martin , 179

Adedjouma, Morayo , 73
Arnaud, Nicolas , 109

Barbier, Franck, 1
Beaudoux, Olivier , 115
Belloir, Nicolas, 1
Blouin, Arnaud , 115
Boukhenoufa, Mohamed Lamine, 65
Bouquet, Fabrice , 49

Cariou, Eric, 1
Chevallereau, Benjamin , 101

Debricon, Stéphane , 49
Dery, Anne Marie , 131
Didonet Del Fabro, Marcos , 163
Dubois, Hubert , 73

Fleurquin, Régis , 95

Grisvard, Olivier , 79

Hammoudi, Slimane , 115

Jouault, Frédéric , 33

Kermarrec, Yvon , 79

Lanusse, Agnès , 65
Le Gloahec, Vincent , 95
Le Pors, Eric , 79
Leclercq, Eric, 187
Legeard, Bruno , 49
Lopes, Valery , 187

Mueller, Heiko , 17

Radermacher, A., 65
Roth, Andreas , 17

Sadou, Salah , 95
Spriestersbach, Axel , 17

Terrasse, Marie-Noëlle, 187