De la modélisation objet des logiciels à la metamodélisation des langages informatiques

Pierre-Alain Muller

▶ **To cite this version:**

Pierre-Alain Muller. De la modélisation objet des logiciels à la metamodélisation des langages informatiques. Génie logiciel [cs.SE]. Université Rennes 1, 2006. tel-00538525

HAL Id: tel-00538525
https://theses.hal.science/tel-00538525

Submitted on 22 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Habilitation à diriger des recherches

présentée devant

L'Université de Rennes 1

par

Pierre-Alain Muller

De la modélisation objet des logiciels

à la

metamodélisation des langages informatiques

Soutenue le 20 novembre 2006 devant le jury composé de :

M. :		PRESIDENT
MM. :	MOKRANE BOUZEGHOUB	RAPPORTEURS
	JACKY ESTUBLIER	
	JEAN-MARC GEIB	
MM. :	PIERRE AMBS	EXAMINATEURS
	JEAN-MARC JEZEQUEL	
	PAUL LE GUERNIC	

Remerciements

Tout d'abord un grand merci aux membres du jury, et en particulier à Mokrane Bouzeghoub, Jean-Marc Geib et Jacky Estublier qui ont accepté d'être rapporteurs pour cette HDR.

Je voudrais également remercier Jean-Marc Jezequel qui m'a accueilli dans le projet Triskell, durant ma période de délégation à l'INRIA (une formule fantastique pour laquelle je remercie d'ailleurs l'INRIA dans son ensemble). Merci à Pierre Ambs pour m'avoir toujours soutenu dans mes activités de recherche, et merci à Paul Le Guernic qui m'a fait découvrir la facette du montage de projets à l'INRIA.

Je voudrais également remercier mes collègues de l'UHA qui ont soutenu ou participé à mes projets passés, notamment la création d'ObjeXion Software.

Plus en amont, je tiens à exprimer ma gratitude à tous les collègues, confrères, doctorants et étudiants avec lesquels j'ai interagi et qui m'ont apporté la matière première de mes travaux.

Une pensée particulière pour mes parents qui m'ont donné l'envie d'étudier et la curiosité de faire de la recherche, et un merci tout spécial à ma compagne Malika et à mes enfants Jonathan, Roxane et Lara.

Pierre-Alain

Table des matières

Introduction 7

1.1 Contexte général de nos travaux 7

1.2 Activités scientifiques 8

1.3 Organisation du mémoire 9

2 Présentation de l'ingénierie dirigée par les modèles 11

2.1 Contexte – le génie logiciel et les méthodes à objets 11

2.2 Zoom sur l'IDM 16

2.3 Les approches de développement basées sur les modèles 18

2.3.1 Model-Integrated Computing (MIC) 18

2.3.2 Model-Driven Architecture (MDA) 19

2.3.3 Les usines logiciels (Software Factories) 19

2.4 Modèles, metamodèles et meta-metamodèles 20

2.4.1 Modèles et Systèmes 21

2.4.2 Metamodèle: langage de modélisation 21

2.4.3 Meta-metamodèle: langage de metamodélisation 21

2.5 Expression de contraintes et utilisation d'OCL 22

2.6 Transformation de modèles 23

2.6.1 Outils génériques 24

2.6.2 Outils intégrés aux AGLs 24

2.6.3 Langages spécifiques 25
2.6.4 Outils de metamodélisation 26
2.7 Conclusion sur l'IDM 27
2.8 Positionnement de mes contributions / l'état de l'art de l'IDM 27
3 Résumé de mes activités de recherche 31
3.1 Modélisation objet 31
3.1.1 Modélisation objet de lois de commande de robotique sous-marine 32
3.1.2 Mise en œuvre de la modélisation objet 34
3.1.3 Emergence d'UML 36
3.1.4 Des modèles aux metamodèles 37
3.1.4.1 Une expérimentation de metamodèle de savoir-faire métier 38
3.1.4.2 Une expérimentation de metamodèle de savoir-faire technique 41
3.2 L'opérationnalisation des modèles 49
3.2.1 Vers des techniques pour le prototypage de modèles UML 49
3.2.2 Réalisation de ponts vers le monde XML 50
3.2.3 A la recherche d'une expérience significative 52
3.2.4 Retour à la vie universitaire 55
3.3 Modèles et ingénierie des langages 59
3.3.1 Initiative TopModL 59
3.3.2 Le système de metamodélisation Kermeta 61

3.3.2.1 Principales limitations des solutions actuelles de metamodélisation 63
3.3.2.2 Vers la conception d'un nouveau metalangage.....65
3.3.2.3 Fabrication de Kermeta.....67
3.3.3 Metamodèle de la syntaxe concrète 71
3.3.4 Exemples d'application de Kermeta et leçons apprises 76
3.3.4.1 Transformation de modèles.....76
3.3.4.2 Aspects pour la modélisation 77
3.3.4.3 Implantation d'un langage spécifique pour l'ingénierie des exigences.....78
3.3.4.4 Leçons apprises 79
4 Perspectives de recherche 83
4.1 Verrous scientifiques.....83
4.2 Travaux à court terme - La plateforme RNTL OpenEmbeDD.....85
4.3 Travaux à moyen terme.....86
4.3.1 Projection vers des modèles formels et remontée de diagnostics 86
4.3.2 Test de transformations de modèles.....87
4.3.3 Modélisation par aspects 88
4.3.4 IDM, automatique et génie informatique 88
4.3.5 Modélisation de l'intention.....89
4.3.6 Cartographie de l'ingénierie des logiciels 90
5 Conclusion.....91

Introduction

1.1 Contexte général de nos travaux

Nos travaux ont pour contexte le génie logiciel, que l'on peut définir comme l'art et la manière de développer du logiciel. Comme son nom l'indique, le génie logiciel est une discipline d'ingénierie. Son objectif est la maîtrise technologique et économique des activités de développement et de maintenance des applications informatiques. Les principales variables d'ajustement sont : les fonctionnalités, les délais, les coûts et la qualité.

Le domaine du génie logiciel est caractérisé tant par une interaction forte entre les mondes académique et industriel, que par une industrie capable de réactions rapides, notamment en termes de normalisation de technologie. Nos travaux de recherche accompagnent ces efforts industriels ; tantôt nos avancées les précèdent, tantôt nos analyses les critique. En résumé, nos travaux de recherche font avancer l'état de l'art, mais nos contributions ne peuvent ignorer l'état des pratiques.

Les pratiques de développement de logiciel ont fortement évolué ces vingt dernières années. Dans les années 80, un des défis majeurs du génie logiciel était de trouver comment organiser et synchroniser le travail de grandes équipes de programmeurs (typiquement plus de 100 personnes), qui développaient de concert une application complexe, comme un système de commande-contrôle de frégate ou un logiciel de pilotage d'autocommutateur. Aujourd'hui les équipes se sont réduites, et la difficulté est souvent de trouver la manière la plus judicieuse d'identifier puis d'assembler des composants existants, souvent hétérogènes, au sein d'environnements en évolution rapide.

En parallèle à cette mutation, un fossé s'est creusé entre la réalité des enjeux de l'informatisation, et la perception du métier par le grand public. Alors que la dépendance informatique de nos sociétés ne cesse de croître, la banalisation de l'informatique a réduit considérablement l'attrait du métier auprès des plus jeunes qui s'orientent plutôt vers les sciences de la vie ou les nanotechnologies [1].

Il en résulte un grand défi pour le génie logiciel qui doit apporter de nouvelles techniques et outils pour répondre notamment aux questions suivantes. Comment allons-nous faire pour développer plus de logiciels avec moins d'informaticiens ? Comment allons-nous continuer de construire des systèmes efficaces, mais intriqués et complexes, et en plus dans un contexte de consommation de masse ? Comment allons-nous nous maintenir en capacité de performance collective, alors que le niveau d'exigence individuel baisse ?

Je postule que l'ingénierie dirigée par les modèles a le potentiel pour répondre à ces questions. Elle permet de capturer le savoir-faire des experts (tant métier que technique), puis de le mettre à disposition des praticiens. Encore faut-il qu'elle délivre des gains de productivité significatifs, quantifiables et indiscutables.

1.2 Activités scientifiques

Mes travaux de recherche concernent la modélisation opérationnelle des systèmes à informatique prépondérante, dans le double but de capitaliser les savoir-faire et d'automatiser les réalisations.

La modélisation est l'utilisation d'une représentation, en lieu et place d'une chose du monde réel, dans un but cognitif. Un modèle se substitue à un système dans un contexte donné, à moindre coût, plus simplement, plus rapidement et sans les risques ou dangers inhérents à une manipulation du système réel. Une modélisation devient opérationnelle dès lors qu'une implantation informatique peut en être dérivée de manière systématique.

L'ensemble de mes travaux a pour but *in fine* de réduire le fossé entre les processus métier et les techniques d'informatisation. Je m'intéresse à la fois à la formalisation des savoir-faire métier et informatique. Ma démarche scientifique mélange des aspects théoriques et des validations expérimentales. Elle procède par fertilisation croisée, en reposant tout à la fois sur l'analyse de l'état de l'art, la participation à des groupes de travail ou de normalisation, et à la confrontation avec des problématiques industrielles. Les idées que j'ai approfondies dans mes travaux ont émergées dans différentes communautés (telles celles des grammaires, des bases de données, de la gestion des documents, des méthodes formelles…).

Je procède en considérant la modélisation comme une alternative à la programmation. C'est

une posture certes un peu radicale – car la modélisation et la programmation peuvent très bien se complémenter – mais je l'ai choisi afin d'amener les techniques de modélisation à leur extrême.

Mes travaux peuvent se résumer de la façon suivante :

- Des contributions méthodologiques pour la mise en œuvre de la notation UML et pour l'intégration de l'IDM et des méthodes agiles.
- Des contributions pour la capitalisation du savoir-faire de modélisation des systèmes de commande et contrôle.
- Des contributions pour la modélisation opérationnelle des systèmes d'information Web.
- Des contributions pour l'application de la modélisation à l'ingénierie des langages.

Ces espaces de contributions sont détaillés dans le chapitre 3 de ce mémoire.

1.3 Organisation du mémoire

Le mémoire est organisé en cinq sections :

- Une introduction.
- Une présentation de l'ingénierie dirigée par les modèles dans le but d'une part de définir les notions abordées dans ce mémoire, et d'autre part de situer mes activités de recherche par rapport à l'évolution du domaine.
- Un résumé chronologique de mes activités de recherche organisé en trois grands thèmes (la modélisation objet, l'opérationnalisation des modèles, les modèles et l'ingénierie des langages).
- Une présentation des perspectives de recherche que j'entrevois pour les prochaines années.
- Une conclusion.

Par ailleurs, j'ai choisi de mettre en annexe quatre publications qui sont représentatives de mes travaux : la première sur la modélisation objet (période allant de 1993-1998), la deuxième sur l'opérationnalisation des modèles (période allant de 1999 à 2002) et enfin deux sur des metamodèles pour l'ingénierie des langages (depuis 2003).

2 Présentation de l'ingénierie dirigée par les modèles

Cette section se présente sous la forme d'un manifeste de l'IDM, dans le but de présenter la discipline et de définir la terminologie. Elle se termine par une critique et un positionnement de mes contributions par rapport à l'évolution de cette discipline.

2.1 Contexte – le génie logiciel et les méthodes à objets

Je reprends ici quelques extraits de l'introduction de mon ouvrage « Modélisation objet avec UML ». [2]

L'informatique s'est glissée imperceptiblement dans quasiment toutes nos activités quotidiennes. Des machines à laver aux lecteurs de disques compacts, en passant par les distributeurs de billets et les téléphones, nombreuses sont les appareils qui utilisent des applications logicielles et, plus le temps passe, plus ces applications deviennent complexes et coûteuses.

La demande de logiciels sophistiqués alourdit considérablement les contraintes imposées aux équipes de développement. Les informaticiens sont confrontés à une complexité croissante, à la fois du fait de la nature des applications, des environnements distribués et hétérogènes, de la taille des logiciels, de la composition des équipes de développement, et des attentes des utilisateurs en matière d'ergonomie. Pour surmonter ces difficultés, les informaticiens doivent apprendre à faire, à expliquer, et à comprendre. C'est pour ces raisons qu'ils ont et auront toujours plus besoin de méthodes.

Une méthode définit une démarche reproductible qui fournit des résultats fiables. Tous les domaines de la connaissance utilisent des méthodes plus ou moins sophistiquées et plus ou moins formalisées. Les cuisiniers parlent de recettes de cuisine, les pilotes déroulent des *check-lists* avant le décollage, les architectes dessinent des plans et les musiciens suivent des règles de composition. Une méthode d'élaboration de logiciels décrit comment construire des systèmes logiciels de manière fiable et reproductible.

De manière générale, les méthodes permettent de construire des modèles à partir d'éléments de modélisation qui constituent des concepts fondamentaux pour la représentation de systèmes ou de phénomènes. Les notes reportées sur les partitions sont des éléments de modélisation pour la musique. L'approche objet propose l'équivalent des notes, à savoir les

objets, pour la représentation des logiciels.

Les méthodes définissent également une représentation, souvent graphique, qui permet d'une part de manipuler aisément les modèles, et d'autre part de communiquer et d'échanger l'information entre les différents intervenants. Une bonne représentation offre un équilibre entre la densité d'information et la lisibilité.

En plus des éléments de modélisation et de leurs représentations graphiques, une méthode définit des règles de mise en œuvre qui décrivent l'articulation des différents points de vue, l'enchaînement des actions, l'ordonnancement des tâches et la répartition des responsabilités. Ces règles définissent un processus qui assure l'harmonie au sein d'un ensemble d'éléments coopératifs et qui explique comment il convient de se servir de la méthode. Avec le temps, les utilisateurs d'une méthode développent un savoir-faire lié à la mise en œuvre de ladite méthode. Ce savoir-faire, également appelé expérience, n'est pas toujours formulé clairement, ni aisément transmissible.

Les méthodes structurées et fonctionnelles se sont imposées les premières, bien que les méthodes objet aient leurs racines solidement ancrées dans les années 60. Ce n'est guère surprenant, car les méthodes fonctionnelles s'inspirent directement de l'architecture des ordinateurs, c'est-à-dire d'un domaine éprouvé et bien connu des informaticiens. La séparation entre les données et le code, telle qu'elle existe physiquement dans le matériel, a été transposée vers les méthodes ; c'est ainsi que les informaticiens ont pris l'habitude de raisonner en termes de fonctions du système. Cette démarche est naturelle lorsqu'elle est replacée dans son contexte historique ; aujourd'hui, elle est devenue quasiment anachronique. Il n'y a en effet aucune raison d'inscrire des réponses matérielles dans des solutions logicielles. Le logiciel s'exécute sur du matériel qui doit être à son service et non sur du matériel qui lui impose des contraintes d'architecture.

Plus récemment, vers le début des années 80, les méthodes objet ont commencé à émerger. L'Histoire, avec un grand H, est un éternel recommencement. La petite histoire des méthodes se répète elle aussi. Le cheminement des méthodes fonctionnelles et des méthodes objet est similaire. Au début existait la programmation, avec dans un cas le sous-programme, et dans l'autre, l'objet comme élément structurant de base. Quelques années plus tard, les

informaticiens poussent le concept structurant vers la conception et inventent la conception structurée dans un cas et la conception objet dans l'autre. Plus tard encore, ils opèrent une progression vers l'analyse, toujours en exploitant le même paradigme, soit fonctionnel, soit objet. Chaque approche peut donc proposer une démarche complète, sur l'ensemble du cycle de vie du logiciel.

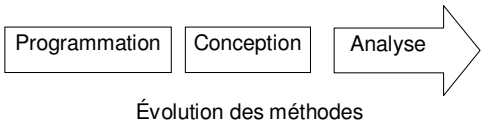

Figure 1 : ***L'évolution des méthodes se fait de la programmation vers l'analyse.***

Dans les faits, la situation est légèrement plus complexe. En effet, souvent les méthodes ne couvrent pas l'ensemble du cycle de vie. Cela se traduit alors par la juxtaposition de méthodes : une méthode A pour l'analyse, suivie d'une méthode C pour la conception. Cette approche parcellaire, tant qu'elle est confinée dans un des paradigmes – l'approche fonctionnelle ou l'approche objet – reste raisonnable. En revanche, le mélange de paradigmes est nettement moins raisonnable, bien que compréhensible. Vers le milieu des années 80, les bienfaits de la programmation objet commencent à être largement reconnus, et la conception objet semble une approche raisonnable pour qui veut mettre en œuvre un langage de programmation objet comme Smalltalk. Du côté de l'analyse par contre, la notion d'analyse objet n'est que vapeur et supputation. Les entreprises ont développé à cette époque une solide connaissance des méthodes d'analyse fonctionnelle et de modélisation sémantique des données ; les informaticiens s'emploient donc tout naturellement à juxtaposer une phase de conception objet à une phase d'analyse fonctionnelle. Cette manière de procéder présente de nombreux inconvénients liés au changement de paradigme. Le passage de l'analyse fonctionnelle à la conception objet nécessite une traduction des éléments de modélisation fonctionnelle en des éléments de modélisation objet, ce qui est loin d'être commode et naturel. En effet, il n'y a pas de bijection entre les deux ensembles, de sorte qu'il faut casser les éléments de modélisation d'une des approches pour construire des fragments d'éléments de modélisation de l'autre approche. Le résultat net de ce changement d'état d'esprit en cours de développement est de limiter considérablement la navigation entre l'énoncé des besoins en amont de l'analyse et la satisfaction de ces besoins en aval de la conception. D'autre part, une

conception objet obtenue après traduction manque souvent d'abstraction et se limite à l'encapsulation des objets de bas niveaux, disponibles dans les environnements de réalisation et d'exécution. Tout cela implique beaucoup d'efforts pour des résultats somme toute bien peu satisfaisants.

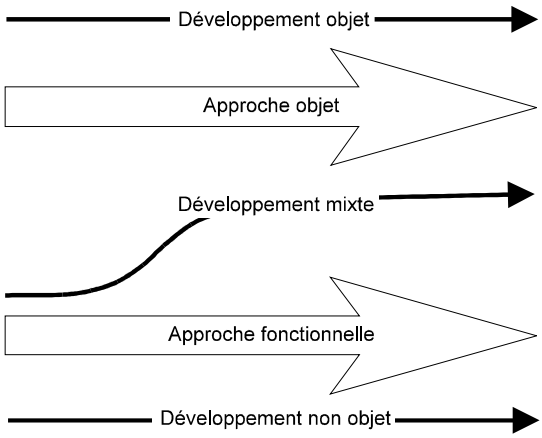

Figure 2 : La combinaison d'une approche fonctionnelle pour l'analyse et d'une approche objet pour la conception et la réalisation n'a plus lieu d'être aujourd'hui car les méthodes objet actuelles couvrent l'ensemble du cycle de vie du logiciel.

A partir de 1995, des applications objet – depuis l'analyse des besoins, jusqu'à la réalisation – ont été développées dans tous les secteurs d'applications. L'expérience acquise sur ces projets permet de savoir comment enchaîner les différentes activités selon une approche totalement objet. Au début des années 90, l'évolution des pratiques n'était pas totale et il existait encore des adeptes de la mixité, prisonniers du poids de leurs habitudes.

La première moitié des années 90 a vu fleurir une cinquantaine de méthodes objet. Cette prolifération est le signe de la grande vitalité de l'objet, mais aussi le fruit d'une multitude d'interprétations de ce qui est objet, de ce qui l'est moins et de ce qui ne l'est pas du tout [3]. Le travers de cette effervescence méthodologique est d'encourager la confusion, de sorte que les utilisateurs se placent dans une situation d'attentisme qui limite les progrès des méthodes. La meilleure validation reste la mise en œuvre ; les méthodes ne sont pas figées, elles évoluent en réponse aux commentaires des utilisateurs.

Fort heureusement, l'examen des méthodes dominantes a permis de dégager un consensus autour d'idées communes. Les grands traits, repris par de nombreuses méthodes à objets,

s'articulent autour des notions de classe, d'association et de partition en sous-systèmes.

Constatant que les différences entre les méthodes s'amenuisent et que la guerre des méthodes ne fait plus progresser la technologie objet, Jim Rumbaugh et Grady Booch décident fin 94 d'unifier leurs travaux pour proposer une méthode unique : la méthode unifiée (*Unified Method*). Une année plus tard, ils sont rejoints par Ivar Jacobson, le créateur des cas d'utilisation (*use cases*), une technique pour la détermination des exigences.

Booch, Jacobson et Rumbaugh se fixent quatre objectifs :

- représenter des systèmes entiers (au-delà du seul logiciel) par des concepts objet ;
- établir un couplage explicite entre les concepts et les artefacts exécutables ;
- prendre en compte les facteurs d'échelle inhérents aux systèmes complexes et critiques ;
- créer un langage de modélisation utilisable à la fois par les humains et les machines.

Les auteurs de la méthode unifiée atteignent rapidement un consensus sur les concepts fondamentaux de l'objet. En revanche, ils s'accordent plus difficilement sur les éléments de notation ; la représentation graphique retenue pour les différents éléments de modélisation connaîtra plusieurs modifications.

La première version de la description de la méthode unifiée est présentée en octobre 1995, dans un document intitulé *Unified Method V0.8*. Ce document est largement diffusé et les auteurs recueillent plus d'un millier de commentaires détaillés de la part de la communauté des utilisateurs. Ces commentaires sont pris en compte dans la version 0.9 parue en juin 1996, mais c'est surtout la version 0.91, disponible depuis octobre 1996, qui rend compte de l'évolution de la méthode unifiée.

La principale modification consiste en la réorientation de la portée de l'effort d'unification, d'abord vers la définition d'un langage universel pour la modélisation objet, et accessoirement vers la standardisation du processus de développement objet. La méthode unifiée (*Unified Method*) se transforme en UML (*Unified Modeling Language for Object-Oriented Development*).

En 1996, il apparaît clairement qu'UML est perçu comme un élément de base dans la stratégie de plusieurs grandes entreprises. C'est ainsi que se crée un consortium de partenaires pour travailler à la définition de la version 1.0 d'UML ; il regroupe notamment DEC, HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI et Unisys.

De cette collaboration naît la description d'UML version 1.0 remise à l'OMG en janvier 1997. La version 1.1 soumise à l'OMG en septembre 1997 est acceptée à l'unanimité deux mois plus tard, en novembre. La version 1.1 devient de ce fait un standard. Après une petite dizaine d'années d'évolution, UML en est actuellement à la version 2.

A partir des années 2000, les besoins en termes de méthodes s'expriment différemment, et cela se traduit notamment par l'émergence (dans le sillage d'UML) de langages normalisés pour la metamodèlisation. Les langages de modélisation se spécialisent (pour le temps-réel, les composants, pour la gestion, pour la démarche…) et le génie logiciel objet n'apparaît plus comme la solution universelle.

Si l'objet reste bien implanté pour la modélisation à petite échelle, de nouvelles approches voient le jour pour prendre en compte le passage à l'échelle. L'ingénierie dirigée par les modèles apporte un cadre unificateur, pour penser et pour réaliser cette évolution.

2.2 Zoom sur l'IDM

Au sens large, l'ingénierie dirigée par les modèles [4] est la discipline qui étudie l'usage des modèles dans les processus d'ingénierie. Un modèle est une représentation simplifiée d'un aspect d'un système (le système pouvant d'ailleurs être un modèle) qui peut être utilisée en lieu et place du système modélisé, par exemple pour réduire les coûts ou diminuer un risque.

Les approches décrites dans ce mémoire se préoccupent plus spécifiquement de développement de logiciels, un champ d'application particulièrement intéressant du fait que les modèles y sont de la même nature que les éléments modélisés. Cette homogénéité permet notamment la génération d'artefacts de génie logiciel à partir de modèles et réciproquement de modèles à partir d'artefacts.

L'ingénierie dirigée par les modèles (IDM) propose des pistes pour permettre aux organisations de surmonter la mutation des exigences du développement de logiciel. Une des raisons majeures de l'apparition des architectures dirigées par les modèles repose sur la volonté de décrire au mieux le savoir-faire ou la connaissance métier d'une organisation dans des modèles abstraits indépendant des plates-formes (PIM - *Platform Independent Models*). Ayant isolé le savoir-faire métier dans des PIM, on a besoin soit de transformer ces modèles en d'autres PIM (besoin d'interopérabilité), soit de produire ou de créer des modèles (PSM - Platform Specific Models) qui leur correspondent pour une plate-forme d'exécution spécifique (pour améliorer la portabilité et augmenter la productivité).

L'IDM est donc une forme d'ingénierie générative, par laquelle tout ou partie d'une application informatique est générée à partir de modèles. Les idées de base de cette approche sont voisines de celles de nombreuses autres approches du génie logiciel, comme la programmation générative, les langages spécifiques de domaines (DSL pour *domain-specific language*) [5] [6], le MIC (Model Integrated Computing) [7] [8], les usines à logiciels (Software Factories) [9], etc.

Dans cette nouvelle perspective, les modèles occupent une place de premier plan parmi les artefacts de développement des systèmes, et doivent en contrepartie être suffisamment précis afin de pouvoir être interprétés ou transformés par des machines. Le processus de développement des systèmes peut alors être vu comme un ensemble de transformations de modèles partiellement ordonné, chaque transformation prenant des modèles en entrée et produisant des modèles en sortie, jusqu'à obtention d'artefacts exécutables.

Pour donner aux modèles cette dimension opérationnelle, il est essentiel de spécifier leurs points de variations sémantiques, et donc aussi de décrire de manière précise les langages utilisés pour les représenter. On parle alors de metamodélisation.

L'intérêt pour l'ingénierie dirigée par les modèles a été fortement amplifié, en novembre 2000, lorsque l'OMG (*Object Management Group*) a rendu publique son initiative MDA [10] qui vise à la définition d'un cadre normatif pour l'IDM. Il existe cependant bien d'autres alternatives technologiques aux normes de l'OMG, par exemple Ecore dans la sphère Eclipse, mais aussi les grammaires, les schémas de bases de données, les schémas XML.

2.3 Les approches de développement basées sur les modèles

Dans le courant des langages spécifiques et de la programmation générative, un certain nombre d'approches basées sur les modèles se développent depuis une décennie. Parmi ces approches, et par ordre chronologique d'apparition, on peut citer le Model-Integrated Computing (MIC), le Model-Driven Architecture (MDA) et les Software Factories. Les sous-sections qui suivent présentent les grandes lignes de chacune de ces approches.

2.3.1 Model-Integrated Computing (MIC)

Les travaux autour du Model-Integated Computing (MIC) [7, 8] proposent, dès le milieu des années 90, une vision du génie logiciel dans laquelle les artefacts de base sont des modèles spécifiques au domaine d'application considéré. Initialement, le MIC est conçu pour le développement de systèmes embarqués complexes [11]. Dans [12] les auteurs discutent de l'implantation et de l'outillage des idées du MIC.

La méthodologie proposée décompose le développement logiciel en deux grandes phases. La première phase est réalisée par des ingénieurs logiciels et systèmes. Elle consiste en une analyse du domaine d'application ayant pour but de produire un environnement de modélisation spécifique à ce domaine. Au cours de cette phase, il faut choisir les paradigmes de modélisation et définir formellement les langages de modélisation qui seront utilisés. A partir de ces éléments, un ensemble de metaoutils sont utilisés pour générer un environnement spécifique au domaine. Cet environnement a l'avantage d'être utilisable directement par des ingénieurs du domaine. La seconde phase consiste à modéliser l'application à réaliser en utilisant l'environnement généré. Cette phase est réalisée uniquement par des ingénieurs du domaine et permet de synthétiser automatiquement le logiciel.

En pratique, les idées du MIC sont implantées dans la suite d'outils (GME) [13] (The Generic Modeling Environment). Dans sa dernière version (5.0), GME s'intègre à l'environnement de développement Visual Studio .NET 2003 de Microsoft et propose des outils pour la création de langages de modélisation et de modèles. La définition de la sémantique des langages de modélisation, et donc la sémantique des modèles manipulés, n'est pas supportée directement: l'interprétation sémantique des modèles doit être réalisée dans une phase avale.

2.3.2 Model-Driven Architecture (MDA)

Le MDA (*Model-Driven Architecture*) [14] est une démarche de développement à l'initiative de l'OMG (*Object Management Group*) rendue publique fin 2000. L'idée de base du MDA est de séparer les spécifications fonctionnelles d'un système des spécifications de son implémentation sur une plate-forme donnée. Pour cela, le MDA définit une architecture de spécification structurée en modèles indépendants des plates-formes (*Platform-Independant Models*, PIM) et en modèles spécifiques (*Platform-Specific Models*, PSM).

L'approche MDA permet de déployer le même modèle sur plusieurs plates-formes grâce à des projections standardisées. Elle permet aux applications d'interopérer en reliant leurs modèles et supporte l'évolution des plates-formes et des techniques. La mise en oeuvre du MDA est entièrement basée sur les modèles et leurs transformations.

L'initiative MDA de l'OMG a donné lieu à une importante standardisation des technologies pour la modélisation. La proposition initiale était d'utiliser le langage UML (Unified Modeling Langage) et ses différentes vues comme unique langage de modélisation. Cependant, il a fallu rapidement ajouter la possibilité d'étendre le langage UML, par exemple en créant des profils, afin de permettre d'exprimer de nouvelles notions. Ces extensions devenant de plus en plus importantes, la communauté MDA a progressivement délaissé UML au profit d'approches utilisant des langages spécifiques. Le langage de metamodélisation MOF, et plus récemment MOF 2.0, a été créé pour permettre la spécification de metamodèles, c'est-à-dire de langages de modélisation. Il est intéressant de noter que, par exemple, UML est construit à partir de MOF.

2.3.3 Les usines logiciels (Software Factories)

Les usines logiciels (*Software Factories*) [9] sont la vision de Microsoft de l'ingénierie des modèles. Le terme d'usine logiciel fait référence à une industrialisation du développement logiciel et s'explique par l'analogie entre le processus de développement proposé et une chaîne de montage industrielle classique. En effet, dans l'industrie classique:

- Une usine fabrique une seule famille de produits. Si l'on considère par exemple l'industrie automobile, une chaîne de montage ne fabrique généralement qu'un seul

type de voiture avec différentes combinaisons d'options.

- Les ouvriers sont relativement spécialisés. Il n'est pas rare de trouver des ouvriers ayant des compétences sur plusieurs postes de la chaîne de montage mais il est très rare qu'un ouvrier ait toutes les compétences depuis l'assemblage à la peinture des véhicules.

- Les outils utilisés sont très spécialisés et fortement automatisés. Les outils utilisés sur une chaîne de montage sont conçus uniquement pour cette chaîne de montage ce qui permet d'atteindre des degrés d'automatisation importants.

- Toutes les pièces assemblées ne sont pas fabriquées sur place. Une chaîne de montage automobile ne fait généralement qu'un assemblage de pièces normalisées ou préfabriquées à un autre endroit.

L'idée des usines logicielles est d'adapter ces caractéristiques aux développements logiciels, qui ont fait leur preuve pour la production de masse de familles de produits matériels. Les deux premiers points correspondent à la spécialisation des fournisseurs de logiciels et des développeurs à l'intérieur des équipes de développement. Le troisième point correspond à l'utilisation d'outils de génie logiciel spécifiques au domaine d'application, c'est-à-dire de langages, d'assistants et transformations spécifiques. Le dernier point correspond à la réutilisation de composants logiciels sur étagères. L'environnement de développement Visual Studio .NET 2005 de Microsoft à été conçu autour de ces idées et propose un environnement générique extensible pouvant initialement être configuré pour un ensemble de domaine d'application prédéfinis.

2.4 Modèles, metamodèles et meta-metamodèles

Cette section définit les concepts de base de l'ingénierie des modèles et les relations qui existent entre ces concepts. Les définitions données dans cette section reprennent les résultats de l'action spécifique MDA du CNRS [4]. La suite de ce document s'appuie fortement sur ces définitions.

2.4.1 Modèles et Systèmes

La première notion importante est la notion de modèle. Quel que soit la discipline scientifique considérée, un modèle est une abstraction d'un système construite dans un but précis. On dit alors que le modèle représente le système. Un modèle est une abstraction dans la mesure où il contient un ensemble restreint d'informations sur un système. Il est construit dans un but précis et les informations qu'il contient sont choisies pour être pertinentes vis-à-vis de l'utilisation qui sera faite du modèle.

A titre d'exemple, une carte routière est un modèle d'une zone géographique, conçu pour circuler en voiture dans cette zone. Pour être utilisable, une carte routière ne doit inclure qu'une information très synthétique sur la zone cartographiée: une carte à l'échelle 1, bien que très précise, ne serait d'aucune utilité.

2.4.2 Metamodèle: langage de modélisation

Afin de rendre un modèle utilisable il est nécessaire de préciser le langage de modélisation dans lequel il est exprimé. On utilise pour cela un metamodèle. Ce metamodèle représente les concepts du langage de metamodélisation utilisé et la sémantique qui leur est associée. En d'autres termes, le metamodèle décrit le lien existant entre un modèle et le système qu'il représente. Dans le cas d'une carte routière, le metamodèle utilisé est donné par la légende qui précise, entre autres, l'échelle de la carte et la signification des différents symboles utilisés.

On dit qu'un modèle est conforme à un metamodèle si l'ensemble des éléments du modèle est défini par le metamodèle. Cette notion de conformité est essentielle à l'ingénierie des modèle mais n'est pas nouvelle: un texte est conforme à une orthographe et une grammaire, un programme JAVA est conforme au langage JAVA et un document XML est conforme à sa DTD.

2.4.3 Meta-metamodèle: langage de metamodélisation

De la même manière qu'il est nécessaire d'avoir un metamodèle pour interpréter un modèle, pour pouvoir interpréter un metamodèle il faut disposer d'une description du langage dans lequel il est écrit: un metamodèle pour les metamodèles. C'est naturellement que l'on désigne

ce metamodèle particulier par le terme de meta-metamodèle. En pratique, un meta-metamodèle détermine le paradigme utilisé dans les modèles qui sont construit à partir de lui. De ce fait, le choix d'un meta-metamodèle est un choix important et l'utilisation de différents meta-metamodèles conduit à des approches très différentes du point de vue théorique et du point de vue technique.

A titre d'exemple, on peut citer deux approches différentes. Tout d'abord, EBNF qui permet des définir des grammaires qui jouent le rôle de metamodèles et dont les mots jouent le rôle de modèles. Ensuite, XML pour lequel la DTD joue le rôle de metamodèle pour des documents XML pouvant alors être considérés comme des modèles. Le formalisme retenu dans les approches à base de modèles récentes, tel que MIC, MDA ou les usines logicielles, est basé sur le paradigme objet. Dans la suite de ce document nous ne considérons que les langages de metamodélisation à objets.

Afin de se soustraire au problème de la définition des meta-metamodèles (et ainsi éviter d'avoir à définir un meta-meta-metamodèle), l'idée généralement retenue est de concevoir les meta-metamodèles de sorte qu'ils soient auto-descriptifs, c'est-à-dire qui se définissent eux-mêmes. Dans le cas des grammaires, on spécifie ainsi EBNF par une grammaire et dans le cas d'XML, c'est une DTD XML pour les DTD qui joue le rôle de meta-metamodèle.

2.5 Expression de contraintes et utilisation d'OCL

Un metamodèle permet de définir (et donc finalement de contraindre) la structure de modèles. Par exemple, les multiplicités des propriétés spécifient les nombres minimum et maximum d'objets qui peuvent participer à chaque association. Cependant, les constructions structurelles des langages de metamodélisation ne permettent pas d'exprimer toutes les contraintes relatives à un metamodèle, notamment lorsque qu'une contrainte porte sur plusieurs associations.

En pratique il est fréquent que des contraintes supplémentaires doivent être ajoutées à un metamodèle pour assurer la cohérence des modèles. Ces contraintes sont alors exprimées dans un langage externe, soit informellement sous simple forme textuelle, soit plus finement au moyen de langages à la sémantique connue.

Une première approche hybride consiste à se projeter dans le domaine d'un langage de

programmation (par exemple Java), en traduisant les éléments de modélisation en constructions du langage, et en utilisant des programmes pour décrire les contraintes. Cette approche a le mérite de la simplicité, par contre elle souffre de problèmes liés à l'alignement partiel des langages de modélisation et de programmation (notamment en ce qui concerne les types de base et les associations).

Une approche plus intégrée consiste à utiliser un langage spécifique, destiné à l'expression de contraintes dans un environnement à base de modèles, comme le langage OCL (*Object Constraint Language*) [15]. OCL a été initialement conçu pour ajouter des contraintes sur les modèles UML sous la forme d'invariants de classes, de pré-conditions et post-conditions pour les opérations. Les éléments de modélisation des diagrammes de classes d'UML et les metamodèles tels MOF étant des formalismes très proches, OCL peut être directement réutilisé pour définir des invariants sur des classes et des associations, et des pré-conditions et post-conditions pour les opérations d'un metamodèle.

Le langage OCL inclut un ensemble de constructions permettant de naviguer parmi les objets d'un modèle afin de vérifier des contraintes. Les constructions d'OCL ne permettent pas de créer, détruire ou modifier les objets d'un modèle: la vérification des contraintes se fait sans effet de bord. En plus de la définition de contraintes, OCL peut être utilisé pour spécifier (de manière déclarative) le comportement de propriétés dérivées et d'opérations, sans toutefois pouvoir exprimer impérativement des modifications de modèles. OCL fournit donc une bonne solution pour exprimer des contraintes sur les modèles mais pas pour décrire complètement le comportement ou la sémantique des modèles. Un des objectifs du langage Kermeta, que nous décrivons plus loin, est de permettre à la fois la définition de contraintes et la spécification du corps des opérations et propriétés dérivées.

2.6 Transformation de modèles

Les deux principaux artefacts de l'ingénierie des modèles sont les modèles et les transformations de modèles. D'un point de vue général, on appelle transformation de modèles tout programme dont les entrées et les sorties sont des modèles. Cette section propose un rapide tour d'horizon des techniques de transformation de modèles existantes. De nombreux outils, tant commerciaux que dans le monde de l'open-source sont aujourd'hui disponibles

pour faire la transformation de modèles. On peut grossièrement distinguer quatre catégories d'outils:

- les outils de transformation génériques qui peuvent être utilisés pour faire de la transformation de modèles ;
- les facilités de type langages de scripts intégrés à la plupart des ateliers de génie logiciel ;
- les outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement classiques ;
- les outils de metamodélisation "pur jus" dans lesquels la transformation de modèles revient à l'exécution d'un metaprogramme.

2.6.1 Outils génériques

Dans la première catégorie on trouve notamment d'une part les outils de la famille XML, comme XSLT [16] ou Xquery [17], et d'autre part les outils de transformation de graphes (la plupart du temps issus du monde académique) [18]. Les premiers ont l'avantage d'être déjà largement utilisés dans le monde XML, ce qui leur a permis d'atteindre un certain niveau de maturité. En revanche, notre expérience (nous avons notamment essayé de réaliser des générateurs de code par transformation XSLT) montre que ce type de langage est assez mal adapté pour des transformations de modèles complexes (c'est-à-dire allant au-delà des problématiques de transcodage syntaxique). En effet, des transformations XSLT ne permettent pas de travailler au niveau de la sémantique des modèles manipulés mais seulement à celui d'un arbre couvrant le graphe de la syntaxe abstraite du modèle, ce qui impose de nombreuses contorsions qui rendent rapidement ce type de transformation de modèles complexes à élaborer, à valider et surtout à maintenir sur de longues périodes.

2.6.2 Outils intégrés aux AGLs

Dans la seconde catégorie, on va trouver une famille d'outils de transformation de modèles proposés par des vendeurs d'ateliers de génie logiciel. Par exemple, l'outil Arcstyler [19] de

Interactive Objects propose la notion de MDA-Cartridge qui encapsule une transformation de modèles écrite en JPython (langage de script construit à l'aide de Python et de Java). Ces MDA-Cartridges peuvent être configurées et combinées avant d'être exécutées par un interpréteur dédié. L'outil propose aussi une interface graphique pour définir de nouvelles MDA-Cartridges. Dans cette catégorie on trouve aussi l'outil Objecteering [20] de Objecteering Software (filiale de Softeam), qui propose un langage de script pour la transformation de modèles appelé J, ou encore OptimalJ [21] de Compuware qui utilise le langage TPL, et bien d'autres encore, y compris dans le monde de l'open-source avec des outils comme Fujaba [22] (From UML to Java and Back Again).

L'intérêt de cette catégorie d'outils de transformation de modèles et d'une part leur relative maturité (car certains d'entre eux comme le J d'Objecteering sont développés depuis plus d'une décennie) et d'autres pas leur excellente intégration dans l'atelier de génie logiciel qui les héberge. Leur principal inconvénient est le revers de la médaille de cette intégration poussée : il s'agit la plupart du temps de langages de transformation de modèles propriétaires sur lesquels il peut être risqué de miser sur le long terme. De plus, historiquement ces langages de transformation de modèles ont été conçus comme des ajouts au départ marginaux à l'atelier de génie logiciel qui les héberge. Même s'ils ont aujourd'hui pris de l'importance, ils ne sont toujours pas vus comme les outils centraux qu'ils devraient être dans une véritable ingénierie dirigée par les modèles. De nouveau ces langages montrent leur limitation lorsque les transformations de modèles deviennent complexes et qu'il faut les gérer, les faire évoluer et les maintenir sur de longues périodes.

2.6.3 Langages spécifiques

Dans la troisième catégorie, c'est-à-dire les outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standards, on va trouver par exemple Mia-Transformation [23] de Mia-Software qui est un outil qui exécute des transformations de modèles prenant en charge différents formats d'entrée et de sortie (XMI, tout autre format de fichiers, API, dépôt). Les transformations sont exprimées comme des règles d'inférence semi-déclaratives (dans le sens où il n'y a pas de mécanisme de type de programmation logique), qui peuvent être enrichies en utilisant des scripts Java pour des services additionnels tels que la manipulation

de chaînes. PathMATE [24] de Pathfinder Solutions est un autre environnement configurable de transformation de modèles qui s'intègre particulièrement bien avec les ateliers de modélisation UML dominant le marché. Dans le monde académique on va trouver de nombreux projets open-source s'inscrivant dans cette approche: les outils ATL [25] et MTL [26] de l'INRIA, AndroMDA [27], BOTL [28] (Bidirectional Object oriented Transformation Language), Coral [29] (Toolkit to create/edit/transform new models/modeling languages at run-time), ModTransf [30] (XML and ruled based transformation language), QVTEclipse [31] (une implantation préliminaire de quelques unes des idées du standard QVT dans Eclipse) ou encore MT-QVT [32] (UML Model Transformation Tool).

2.6.4 Outils de metamodélisation

La dernière catégorie d'outils de transformation de modèles est celle des outils de metamodélisation dans lesquels la transformation de modèles revient à l'exécution d'un metaprogramme.

Le plus ancien et le plus connu de ces outils est certainement MetaEdit+ [33, 34] de MetaCase. Celui-ci permet de définir explicitement un metamodèle, et au même niveau de programmer tous les outils nécessaires, allant des éditeurs graphiques aux générateurs de code en passant par des transformations de modèles. Dans une veine similaire, XMF-Mosaic [35] de Xactium est un environnement complet de définition de langage. Au coeur de XMF-Mosaic on trouve un noyau exécutable de définition de langage (qui est d'ailleurs auto-définissant). Tout est modélisé sur cette base, que ce soit les langages (e.g. UML), les outils, les GUI, parseurs et autres analyseurs XML... Le langage de metamodélisation de base est un langage orienté objet qui a toute la puissance d'un langage de programmation complet ce qui en fait un outil particulièrement puissant pour la transformation de modèles.

Dans le monde de l'*open-source* on trouvera dans cette catégorie l'environnement Kermeta développé à l'INRIA dont nous parlerons largement plus loin dans ce mémoire. Kermeta peut être décrit comme l'ajout d'un aspect d'exécutabilité dans le MOF par un mécanisme astucieux inspiré du tissage d'aspects ; ceci lui permet de garder une compatibilité totale avec les environnements MOF standard (par exemple EMF).

2.7 Conclusion sur l'IDM

Il est encore un peu tôt pour savoir si l'IDM parviendra à relever le défi de la rénovation du génie logiciel. Les techniques à base de modèles semblent prometteuses mais il nous appartient pour le moment de les analyser, de classifier leurs artefacts et de comprendre quand et comment les utiliser.

2.8 Positionnement de mes contributions / l'état de l'art de l'IDM

Les paragraphes précédents ont présenté l'évolution du domaine de la modélisation objet et de l'IDM sur une quinzaine d'années.

J'ai accompagné cette évolution, en réagissant au fur et à mesure aux points faibles du moment. Je vais resituer dans les paragraphes suivants mes contributions par rapport à cette évolution. Le chapitre 3 présentera ces contributions plus en détail.

UML n'est pas connu avant 1997. Au début des années 90, il existe certes des travaux sur des approches de modélisation (Booch, Classes-Relations, HOOD, OMT…), mais ces travaux sont loin de remporter l'adhésion de l'industrie qui va caractériser le déploiement d'UML. Chaque technique propose des notions plus ou moins similaires à celles des autres, mais il est très difficile de comparer ces techniques en l'absence d'un système de référence commun. L'absence de formalisation précise rend difficile à la fois l'utilisation des concepts et l'échange des modèles (du fait des variations d'interprétation). J'axe alors mes travaux sur la définition des types de relations dans les modèles [36], plus particulièrement dans le contexte des méthodes de Booch et OMT.

A partir de 1997, la première version d'UML est disponible, mais elle se présente essentiellement sous la forme d'une boîte à outils, d'une collection de concepts sans mode d'emploi. Par ailleurs, la définition de ces concepts reste imprécise, largement basée sur des textes en langage naturel, ou alors sur des diagrammes eux-mêmes exprimés en UML. Il en résulte deux sévères limitations :

- pour les utilisateurs, l'absence de méthode pour les guider lors de la réalisation des modèles,

- pour les outilleurs, l'absence de sémantique précise, et donc une multiplicité de points de variations d'interprétation des modèles.

Du fait de ces deux grandes faiblesses, UML 1.0 ne permet pas vraiment de faire des modèles productifs, et reste cantonné à une technique de dessin. La réutilisation des modèles UML est quasiment impossible hors de leur contexte d'élaboration.

Mes contributions de la fin des années 90 portent essentiellement sur les points d'utilisation et les problèmes méthodologiques avec UML, notamment dans une perspective de réutilisation de modèles appliqués au domaine de la commande et du contrôle [37] [38] [39]. Je m'intéresse en particulier à comprendre comment représenter les spécificités des domaines avec un langage de modélisation généraliste comme UML. Je détaille ces activités de recherche dans le chapitre 3.1 (Modélisation objet).

Toute l'expérience de la modélisation acquise sur cette décennie sera consolidée dans la seconde édition de mon ouvrage sur la modélisation objet avec UML [40]. Je contribue également à la définition de la sémantique des extrémités d'associations d'UML en soumettant des rapports d'anomalies à l'OMG.

A partir du début des années 2000, et notamment du fait de l'émergence de la notion de lignes de produits [41], des travaux sur la programmation générative [42], et de l'amélioration du niveau de formalisme d'UML, il devient envisageable de rendre les modèles UML plus productifs et de s'attaquer à la transformation systématique des modèles en artéfacts exécutables. La validité industrielle de cette approche n'est cependant pas établie.

Je fais de cette validité industrielle une priorité et je choisi le domaine des systèmes d'informations Web comme champs d'application (car je suis persuadé que ce domaine à beaucoup à gagner du génie logiciel, qui n'y est que très peu présent). Mes contributions sur l'opérationnalisation des modèles se déclinent d'une part sous la forme de techniques et outils pour l'ingénierie des systèmes d'information sur Internet [43], et d'autre part d'avancées méthodologiques, notamment pour l'intégration de l'IDM avec les méthodes agiles [44]. Je rapporte ces travaux dans le chapitre 3.2 (Opérationnalisation des modèles).

L'arrivée du MDA en 2000 entraîne la promotion d'une architecture de metamodélisation

basée sur le MOF (Meta-Object Facility) et met en lumière que l'ingénierie dirigée par les modèles est plus centrée sur les metamodèles que sur les modèles. UML perd sa position centrale, au profit de MOF, et se retrouve a égalité avec d'autres langages de modélisation comme CWM, SPEM, AADL, SysML…

Ce glissement vers les metamodèles a aussi pour conséquence la création de nombreux nouveaux langages spécifiques (par exemple pour l'expression de contraintes, d'actions, de transformations, de sérialisation…). Il en résulte une sévère problématique d'intégration de ces langages qui tendent à disperser les concepts de manipulation de modèles. Il faut doter l'IDM de moyens pour contrôler la définition et la fusion de ces langages, et notamment pour résoudre les problèmes durs de recouvrement de domaines sémantiques.

A partir de 2002, mes contributions portent sur l'application des modèles à l'ingénierie des langages. La démarche retenue a été de définir tout d'abord un noyau commun aux différents langages spécifiques de la mouvance MDA, puis de promouvoir ce noyau en un système de metamodélisation, compatible avec EMOF [45]. Mes travaux les plus récents portent sur la modélisation des syntaxes concrètes et sur la liaison avec le monde des grammaires [46]. Ce thème est traité en détail dans le chapitre 3.3 (Modèles et ingénierie des langages).

3 Résumé de mes activités de recherche

Cette section présente mes travaux de recherche de manière chronologique, pour la période 1993-2006. Elle comporte trois thèmes principaux : la modélisation objet, l'opérationnalisation des modèles et l'application des modèles à l'ingénierie des langages.

Le premier thème a été traité en collaboration avec les deux doctorants que j'ai co-encadré au sein du laboratoire MIPS, en partenariat avec Rational Software et Nipson Printing System.

Le deuxième thème a été mis en œuvre au sein de la startup ObjeXion Software, notamment dans le cadre de la réalisation de l'AGL Netsilon, pour la modélisation opérationnelle de systèmes d'information Web.

Le troisième thème correspond à mes préoccupations actuelles au sein du projet Triskell, à l'IRISA à Rennes. Dans ce cadre j'ai notamment co-encadré la thèse de Franck Fleurey et coordonné le montage du projet RNTL OpenEmbeDD.

3.1 Modélisation objet

Dans ma thèse de doctorat, *L'intégration dans les systèmes logiciels complexes, modélisation et architecture*, soutenue en décembre 1993, j'ai proposé des modèles et des architectures à base d'objets distribués pour favoriser l'intégration de composants hétérogènes au sein d'applications informatiques de grande ampleur. Cette thèse a été réalisée dans le cadre d'une convention CIFRE entre l'ANRT, l'Université de Haute-Alsace et la société Rational Software. J'ai partagé mon temps à parité entre le laboratoire et l'entreprise, et je garde un excellent souvenir de cette période, notamment car j'ai eu l'occasion de rencontrer un très grand nombre de praticiens de la modélisation, qui travaillaient sur des exemples concrets, dans des situations opérationnelles.

En plus de mon mémoire de thèse, ce thème a donné lieu à la communication *Integrating OSF/MOTIF User Interfaces with Ada Applications* à la conférence internationale GL'92 [47] dans laquelle je discute notamment de la cohabitation d'espaces techniques différents (par exemple les langages Ada et C/C++) et à deux articles dans *Rational Technical Newsletter*, respectivement *RMI, a « motified » version of RXI* [48] et *Integration, CASE tools and software engineering* [49].

Nos travaux ont trouvé une application pratique lors du « revamping » de l'interface utilisateur de l'environnement Ada de Rational, qui a été basée sur les mécanismes que j'ai mis au point. Cette approche a notamment permis à Rational d'améliorer relativement facilement son image, en passant d'une solution entièrement propriétaire (sur matériel dédié) à une solution plus ouverte, hétérogène, avec la partie présentation de ses éditeurs directement exécutables sur stations de travail.

Avec le recul, et en opérant une reformulation orientée IDM, il est intéressant de voir comment j'ai utilisé la technique des valeurs actives [50] pour réaliser une forme d'adaptation d'impédance entre deux meta-modèles. Une valeur active peut être vue comme un point de transformation de modèles, qui permet de faire interagir deux programmes (deux modèles) qui sont conformes à deux espaces techniques différents. L'appel de la transformation est implicite ; il est réalisé par déclenchement des méthodes *get* et *set* (qui encapsulent le code de transformation) chaque fois que la valeur active est accédée, respectivement en lecture ou en écriture.

De manière beaucoup plus générale, le principal enseignement de ce travail sur l'intégration et la cohabitation d'éléments logiciels est que le purisme technique ne sert pas à grand-chose, car le plus important c'est la valeur perçue par l'utilisateur. Une adaptation technique modeste, mais bien conçue, peut avoir beaucoup plus de valeur qu'une transformation en profondeur d'un logiciel. La valeur intrinsèque du logiciel est très difficile à déterminer, et ne réside pas dans ce que fait un logiciel, mais plutôt dans ce qu'il permet de faire.

Après ma thèse, je me suis appliqué à bien comprendre les techniques de modélisation objet existantes, tant dans leur portée que dans leur utilisation. Dans ce contexte, en partenariat avec Rational Software, j'ai assuré de nombreuses actions de transfert de connaissance vers l'industrie que je considère comme un champ d'expérimentation privilégié lorsque l'on se préoccupe de génie logiciel.

3.1.1 Modélisation objet de lois de commande de robotique sous-marine

En 1993, j'ai participé au projet Vortex de l'Ifremer à Toulon. Le Vortex est un sous-marin de poche, conçu spécialement pour la mise au point de lois de commande pour la robotique sous-marine. Le Vortex est équipé de plusieurs moteurs électriques pour assurer ses déplacements,

ainsi que de nombreux capteurs et effecteurs ; il est relié en permanence à une station de contrôle en surface sur le navire qui déploie le sous-marin. Un cordon ombilical assure l'alimentation électrique du sous-marin, ainsi que la communication entre les différents processeurs (au fond et en surface).

La problématique scientifique de ce projet à la croisée de la robotique et de l'informatique était liée à l'incompatibilité entre les approches de composition de modèles en automatique (la composition de fonctions de transfert) et en informatique (le produit d'automates) [51]. Au cours de ce projet, nous avons notamment étudié et mis en place une architecture de modélisation et d'intégration pour assurer la collaboration entre les équipes d'informaticiens chargées de la réalisation du logiciel (qui développaient du C++) et les spécialistes de robotique sous-marine, qui étaient en charge du développement des lois de commandes (au moyen d'outils comme *Matlab* ou *Simulink*). Nous reviendrons plus tard sur cette problématique d'informatisation de lois de commande, notamment au cours de la présentation des travaux de thèse de Nathalie Gaertner sur les micro-architectures pour la commande et la supervision de processus (voir infra).

J'ai particulièrement apprécié dans ce projet d'œuvrer à l'interface entre deux communautés, avec des préoccupations, un vocabulaire et des habitudes de travail très différentes. Un résumé de ce projet mêlant informatique et automatique a été présenté à la conférence internationale GL'93 sous le titre *Une expérience de développement orienté-objet d'un logiciel de contrôle-commande de robot sous-marin : le projet VORTEX* [52].

A nouveau, il est intéressant d'opérer une reformulation des concepts. A l'époque nous parlions de bibliothèques pour réaliser l'interface entre les informaticiens et les automaticiens. Aujourd'hui nous parlerions plutôt de la définition d'un langage spécifique de domaine, au travers d'un metamodèle dédié à la commande. Il s'agit d'ailleurs ici d'une problématique tout à fait récurrente en ingénierie dirigée par les modèles. Une bibliothèque peut toujours être vue comme la première étape de la réalisation d'un DSL [53]. Une fois que des notions spécifiques ont été mises au point et sont regroupées en bibliothèques, il est possible de promouvoir ces notions en concepts d'un langage spécifique, propre au domaine concerné. Cette deuxième étape n'est vraiment pertinente que si elle permet à une autre population d'utilisateurs (par exemple sans formation informatique) de développer (ou plus souvent de

personnaliser) une application informatique. Face à une population de développeurs, il n'est généralement pas nécessaire d'aller au-delà de l'étape de mise en bibliothèques.

3.1.2 Mise en œuvre de la modélisation objet

Lors de mes contacts avec le monde industriel, j'ai été frappé par la sous-utilisation des techniques de modélisation objet en informatique, à la fois comme élément de documentation, mais surtout comme artefacts opérationnels.

Au début des années 90, la pertinence de l'utilisation de modèles objet dans un processus d'ingénierie logiciel n'est pas établie. L'approche objet est en concurrence avec la démarche de décomposition fonctionnelle. Le déploiement de méthodes objet, telles Booch ou OMT, est freiné par de multiples facteurs comme l'absence de consensus sur les notions de base, le manque de précision des formalismes, et l'immaturité des outils.

Pour faire avancer les choses, je décide de porter mon effort sur les aspects méthodologiques, en accompagnant des projets industriels en partenariat avec Rational Software.

J'ai recensé les grandes classes de difficultés rencontrées par les ingénieurs pour s'approprier les usages des différents types d'éléments de modélisation objet et en ai fait état au congrès biennal de l'AFCET en 1995 dans la communication *Rapport d'expérience sur l'enseignement de l'approche orientée-objets* [54]. Dans les grandes lignes, et du point de vue du langage de modélisation, les utilisateurs rencontraient de nombreuses difficultés liées à l'usage de l'héritage et de la liaison dynamique, principalement du fait des difficultés inhérentes aux classifications, en particulier dans les cas à nomenclatures multiples. Une deuxième catégorie de difficultés était plus liée à la démarche de développement, par exemple lors du passage d'une approche fonctionnelle, à une approche objet.

J'ai proposé une démarche de formation pour surmonter les difficultés décrites précédemment à la conférence Object-Expo à Paris en décembre 1995, dans la communication *Un exemple d'enseignement global de l'approche orientée-objets* [55]. J'ai par la suite présenté une heuristique pour identifier le type des relations dans la communication *Comment identifier les différents types de relations dans les modèles orientés-objets, application à la méthode de Booch* [36] lors des journées de synthèse sur les technologies objets de l'AFCET en janvier

1996.

De 1994 à 1996 je me suis également intéressé à l'architecture des modèles, c'est-à-dire à l'équilibre entre les différents éléments de modélisation, et à leur mise en perspective dans un modèle. Je me suis attaché à faire ressortir les correspondances entre les 4+1 vues [56] définies par Philippe Kruchten avec qui j'ai travaillé comme consultant au sein de Rational Software, et j'ai montré comment le processus de modélisation pouvait fonctionner dans un mode de décomposition pyramidal.

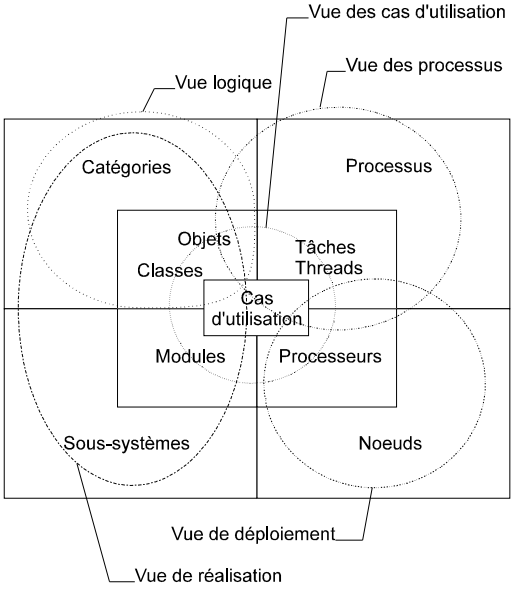

Figure 3 : Décomposition pyramidale d'une architecture de modèles (vue du dessus).

Ce thème a donné lieu à deux communications de workshops *Architecture objet*, Séminaire EEA, Mulhouse, mars 95 [57] et *Expression des vues d'architecture avec UML*, AFCET, mars 97 [58] ; ainsi qu'à une communication à la conférence internationale GL'97, intitulée *Architecture avec UML* [59] puis à une version étendue dans la revue nationale Génie Logiciel N. 46 *Représentation des vues d'architecture avec UML* [60]. J'aborde également largement ce thème dans mon ouvrage *Modélisation Objet avec UML* [2].

Aujourd'hui, il me semblerait pertinent de repenser ce thème des vues d'architecture sous l'angle du tissage d'aspects dans les modèles.

3.1.3 Emergence d'UML

L'année 1997 est marquée par la naissance de la norme de modélisation UML, disponible dans sa version 1.0 le 17 janvier 1997. Cette norme est issue d'un effort d'origine plus industriel qu'académique et répond à la volonté des éditeurs de logiciels de de-segmenter le marché des outils de modélisation. En effet, avant UML le marché de ces outils peinait à décoller, notamment du fait du grand nombre de notations et méthodes de modélisation qui favorisaient l'attentisme des utilisateurs potentiels.

La normalisation d'UML a cependant engendré plus de problèmes qu'elle n'en a résolu. Nous avons bien une norme de modélisation, mais cette norme est floue, repose sur des concepts peu ou mal définis, et rien n'est dit sur la manière d'utiliser ces concepts ! Il nous semble évident qu'UML doit dépasser les *petits dessins,* devenir plus précis, et qu'il s'agit probablement là d'une occasion unique de réconcilier le génie logiciel avec les méthodes formelles.

Dans cette optique, je me préoccupe alors tout particulièrement de la formalisation de la démarche de développement avec UML. Je présente mes contributions à plusieurs conférences :

- *Les derniers développements d'UML 1.0* à SSD'97 [61]
- *Démarche de modélisation d'un système d'information avec UML,* conférence invitée, Inforsid'98 [62]
- *Mise en oeuvre d'UML, une méthode de développement universelle,* SSD'98 [63]
- *Solution cherche problème,* conférence invitée, Journée UML'2000 [64]

Ainsi que dans deux articles de revue nationale:

- *Introduction à UML,* Point DBF N 96, P15-17, Janvier 1999 [65]
- *Mise en oeuvre d'UML, vers une démarche de développement universelle,* Point DBF N 96, Janvier 1999 [66]

3.1.4 Des modèles aux metamodèles

UML est un langage de modélisation objet, certes généraliste mais pas neutre. Il propose un ensemble d'éléments de modélisation, largement basés sur la technique des objets [67]. Les éléments du système en cours de modélisation doivent être appréhendés au moyen des éléments de modélisation définis par UML, au risque parfois de devoir tordre une réalité pour l'exprimer en UML.

Le choix d'un langage de modélisation est donc très important et doit être en bonne adéquation avec la finalité de la modélisation et la complexité du système à modéliser. Dans un contexte de modélisation spécifique, la pertinence d'un langage généraliste comme UML n'est pas garantie.

UML propose les profiles, une forme de personnalisation du langage, réalisée par dérivation des concepts existants. Ce mécanisme, considéré par certains comme une technique de metamodélisation au rabais, fait l'objet de nombreuses critiques dans la littérature [68]. Le MOF est une alternative aux profiles UML ; il permet la spécification de nouveaux metamodèles sans liens avec UML. Une troisième voie est apportée par UML 2, qui sépare la spécification d'UML en un niveau d'infrastructure (globalement des composants réutilisables pour fabriquer des metamodèles) et la superstructure (qui est le langage UML proprement dit). L'infrastructure pouvant s'auto-décrire, il en résulte une concurrence avec le MOF, ce qui vient encore compliquer l'espace technologique.

Tout cela créé un verrou méthodologique. Comment doit-on utiliser UML pour modéliser les spécificités d'un domaine ? En considérant qu'il existe plusieurs manières de modéliser les artefacts d'un domaine, quand doit-on :

- fabriquer des bibliothèques à partir des éléments de modélisation définis UML ?
- étendre les concepts d'UML pour représenter les concepts du domaine ?
- abandonner UML pour définir un nouveau metamodèle ?

A l'époque, nous n'avons pas trouvé de réponse très tranchée à ces questions. Notre position a évolué progressivement, passant d'une recherche de l'utilisation optimale d'UML, à la

définition de bibliothèques et à l'intégration avec des techniques de réutilisation, comme les patterns et frameworks. Nous avons mené des expériences significatives, en collaboration avec deux doctorants que j'ai co-encadré, respectivement Marie-Christine Roch et Nathalie Gaertner. Nous avons travaillé d'une part sur la metamodélisation de savoir-faire métier (en l'occurrence, le domaine de l'impression à haute vitesse) et d'autre part sur la metamodélisation de savoir-faire technique (la commande et la supervision de processus).

Notre position actuelle va au-delà de nos choix de l'époque. Nous privilégions maintenant la fabrication de langages de modélisation dédiés. Dans le contexte des deux travaux de thèses décrits ci-dessous, nous n'avions pas encore tous les éléments de compréhension dont nous disposons à présent (en particulier du fait de nos travaux sur les modèles appliqués à l'ingénierie des langages, présentés plus loin dans ce mémoire).

3.1.4.1 Une expérimentation de metamodèle de savoir-faire métier

Un cas typique de besoin de metamodèle spécifique à un métier, nous a été apporté par NIPSON Printing System, leader mondial de l'impression magnéto-graphique, qui était confronté à la multiplication des versions de ses logiciels de commande d'imprimantes. Ces logiciels écrits en C sont à fois identiques dans leurs grandes lignes et en même temps suffisamment différents dans leurs détails, pour que les équipes de développement se soient retrouvées face à un cauchemar en phase de maintenance.

La société Nipson avait initialement pris contact avec le laboratoire MIPS dans le but de trouver une solution à son problème dans le monde de la programmation. Les équipes de développement pensaient qu'une transition vers C++, imaginée indolore pour des spécialistes du C, résoudrait tous leurs problèmes de variabilité de code. Le problème était évidemment bien plus large !

La problématique de Nipson était avant tout une problématique de réutilisation de savoir-faire métier, et au premier chef d'expression de ce savoir-faire, selon un formalisme indépendant des détails de chaque réalisation.

Pour résoudre ce problème, nous avons collaboré avec la société Nipson pendant environ 4 années, notamment dans le cadre de la thèse CIFRE de Marie-Christine Roch (dont j'ai assuré

le co-encadrement à 80 %). Cette thèse est intitulée *Reformulation orienté-objet des chaînes d'impression informatique à haute vitesse.*

Dans ces travaux nous avons étudié comment la modélisation d'un processus industriel existant pouvait conduire d'une part à la reformulation du processus lui-même, et d'autre part à la définition d'éléments de modélisation spécifiques ; en d'autres termes à la définition d'un metamodèle dédié.

Nous nous sommes intéressés tout d'abord à l'étude du métier de l'impression informatique en vigueur dans les imprimeries actuelles [69]. L'impression informatique se distingue des techniques d'impression traditionnelles par le fait qu'elle traite de gros volumes d'informations variables. Les documents concernés peuvent représenter jusqu'à 100 000 pages différentes, imprimées à une vitesse supérieure à 1000 pages à la minute. Cette caractéristique conduit à un volume considérable d'informations à traiter lors du processus de composition, ce qui impose l'automatisation complète des travaux de composition et d'impression, contrairement au pré-presse traditionnel.

Une partie importante du métier à trait à la composition automatique de documents qui s'appuie sur des techniques traditionnelles d'imprimerie - certaines sont le fruit de cinq siècles d'expérience - et sur des technologies informatiques beaucoup plus récentes [70].

Au coeur du métier se trouve la notion de document électronique [71]. En effet, si le but d'un atelier d'impression est bien de produire un document imprimé, le document existe sous une forme électronique avant son impression. Il est soit créé spécialement dans un but d'impression, soit issu d'un système de gestion électronique de document (GED) [72]. Nous avons étudié les principales techniques qui ont permis le passage du document papier au document électronique, notamment les formats des documents électroniques, les normes et les standards de fait existants, les outils qui les gèrent et les concepts qu'ils mettent en oeuvre.

Une part importante du travail a été consacrée à la structuration des documents et aux avantages qu'il est possible d'en tirer pour la composition automatique. Nous nous sommes appuyés sur les normes SGML ISO 8879 (description du contenu logique), SPDL ISO 10180 (enrichissement) et DSSSL ISO 10179 (restitution).

La démarche retenue a mis en oeuvre la technique des cas d'utilisation [73] pour délimiter le système, à la fois dédié à la création et à la composition automatique de document. L'aspect comportemental du système a été décrit à l'aide d'objets et de scenarii. Une classification des objets a conduit à la structure statique du système exprimée sous la forme de diagrammes de classes. La classification, qui offre une possibilité de garantir la cohérence d'un ensemble de documents, introduit la notion de modèle de document. La construction de bibliothèques de composants pour les documents propose une solution pour promouvoir la réutilisabilité et la portabilité des impressions. Les types de composants sont choisis dans l'optique de favoriser la capitalisation de compétences métiers.

Nous avons proposé une modélisation objet (en UML) du document et de sa composition (une approche que nous reprendrons et formaliserons plus tard dans nos travaux sur la synthèse de syntaxe concrète, notamment pour la synthèse de texte HTML dans le contexte de l'outil Netsilon présenté plus loin).

Cette modélisation objet, associée à l'identification des exigences d'évolution continuelle du métier, a conduit naturellement à un glissement vers une metamodélisation prenant en compte l'intégration des différents points de vue (relatifs aux différents métiers) qui en résulte.

En particulier, il met en évidence plusieurs niveaux d'objets métiers. Les classes du modèle représentent les objets des métiers des informaticiens et des imprimeurs alors que la réutilisation de portions de modèles permet la création d'une base d'objets métiers relatifs aux types d'informations véhiculées par les documents produits.

Une mise en œuvre du metamodèle avec une technologie à objets communicants [74], a montré qu'il était possible de garantir l'interoperabilité des applications et la distribution de l'atelier d'impression sur plusieurs plateformes.

Les résultats de cette approche de reformulation ont été présentés à la conférence Object-Expo'97 sous le titre *Mise en oeuvre d'UML pour la modélisation objet de publipostages complexes* [75] et à la conférence internationale GL'97 sous le titre *Modélisation d'une application de composition automatique de documents avec* UML [76]. Une version étendue de cette communication a également été publiée dans la revue nationale Génie Logiciel N. 46 [76]. Une dernière communication sur le sujet, plus axée sur les gains de flexibilité

opérationnels a été présentée à la conférence internationale CESA'98 sous le titre *Improved flexibility of a document production line through object-oriented remodelling* [37].

Après sa thèse, Marie-Christine Roch a choisi de participer à la mise en application de ses travaux de recherche au sein des équipes de développement de NIPSON Printing System. Elle occupe actuellement une fonction d'audit des pratiques internes, elle est directement rattachée à la direction générale.

Avec le recul, je vois dans ce travail une bonne illustration des limites de la réutilisation de code. Les seules techniques de programmation objet ne sont en fait pas suffisantes, comme nous le savons aujourd'hui, avec l'émergence de la notion de lignes de produits depuis la deuxième moitié des années 90.

Ce cadre de lignes d'impression serait certainement un bon champ d'application pour valider l'IDM dans le contexte de lignes de produits.

3.1.4.2 Une expérimentation de metamodèle de savoir-faire technique

Les travaux présentés dans cette section ont été réalisés dans le cadre de la thèse MNRT de Nathalie Gaertner (dont j'ai assuré le co-encadrement à 50 %) intitulée : *Patterns métier et architectures génériques pour la commande et la supervision de processus.*

Ce travail de recherche à la frontière entre l'informatique et l'automatique participe d'une volonté de rapprocher les groupes (automaticiens, mécaniciens et informaticiens) qui composent le laboratoire MIPS. Le M de MIPS signifie Modélisation, toutefois il existe de nombreux types de modélisation, et un des objectifs de cette thèse à été d'établir des ponts entre la modélisation informatique par objets, patterns et frameworks, et la modélisation automatique, par processus, équations différentielles et fonctions de transfert.

Le point dur est de trouver le bon niveau d'interaction entre les deux approches, et cela d'autant plus que le niveau de granularité des modèles n'est pas le même.

Du fait du domaine interdisciplinaire de ce travail, il a d'abord fallu comprendre puis modéliser le domaine de l'automaticien et la problématique de la commande des systèmes dynamiques discrets. Dans un deuxième temps, des solutions de contrôle-commande

existantes ont été analysées, avant de préciser les concepts et les qualités apportés par la technologie des objets. Un état de l'art sur les artefacts de réutilisation (restreint aux patterns et frameworks, nous n'étions pas sensibilisés aux composants à l'époque) a ensuite permis de préciser les concepts de patterns (patterns d'analyse, de conception ou métier) et de frameworks, ainsi que leurs avantages respectifs et complémentaires.

Le systèmes dynamiques possèdent un ou plusieurs composants qui varient avec le temps [77]. Les frontières d'un système dynamique définissent et encapsulent un état et un comportement. Pour un automaticien, un système – les automaticiens parlent plutôt de processus – est un ensemble de relations causales entre des grandeurs d'entrées (les causes) et des grandeurs de sorties (les effets) [78]. Les entrées influencent l'état du système alors que les sorties reflètent l'état ou le comportement. Un tel système peut être modélisé par une fonction $y = f(u, x, t)$ avec u le vecteur d'entrée, y le vecteur de sortie, x l'état du système et t le temps.

Les processus dynamiques se décomposent globalement en trois familles : les processus continus, les processus discrets et les systèmes hybrides qui évoluent à la fois de manière continue et en fonction de l'occurrence d'événements [79].

Les automaticiens définissent les termes suivants :

- **Contrôle** : tâche qui a pour but de maîtriser un processus ou de maintenir des objectifs en exerçant des commandes.

- **Commande** : la commande permet d'agir sur un système. En général elle utilise des effecteurs pour réaliser sa tâche.

- **Instrumentation** : l'instrumentation d'un processus consiste à ajouter des capteurs pour pouvoir observer des états de ce processus.

- **Asservissement** : contrôle d'un système automatique dont le fonctionnement tend à annuler l'écart entre une grandeur commandée et une grandeur de commande (consigne).

- **Régulation** : mode de fonctionnement d'un système asservi dans lequel la grandeur

réglée tend à se rapprocher d'une grandeur de référence tout en compensant d'éventuelles perturbations.

- **Supervision** : action qui surveille et qui révise une commande.

- **Planification** : la planification permet d'organiser et de régler la commande selon un plan, une tactique ou une stratégie. Elle définit ainsi une ou plusieurs séquences d'actions qui s'exécutent en séquence ou en parallèle.

- **Maintenance** : la maintenance définit un ensemble de mesures qui ont pour but de détecter et de corriger les dégradations du système.

Nous utiliserons dans ce qui suit les termes de contrôle et de contrôleur dans un sens un peu plus large, en y englobant les actions de contrôle-commande incluant la supervision, la détermination de la loi de commande et la conduite de procédés

Un contrôleur (ou correcteur) transforme des informations du système à contrôler en signal de commande, en fonction de lois de contrôle et d'un état désiré du système [80]. Le but de l'automaticien est de fournir la détermination de la commande du processus pour que les sorties du système prennent les valeurs voulues, tout en satisfaisant certains critères (stabilité, temps de réponse, précision, etc.) [81].

Le rebouclage du système – boucle fermée par opposition à boucle ouverte – apporte certains avantages tels que :

- stabilisation d'un système instable en boucle ouverte,

- compensations de perturbations externes,

- poursuite d'une consigne,

- accélération des réponses du système.

Pour définir ce rebouclage, il faut observer le système au niveau de ses sorties et selon la stratégie du correcteur, définir les entrées à injecter dans le système. Les entrées du correcteur sont des observations faites sur le système et sont représentées par un ou des vecteurs

d'observation, alors que les sorties du contrôleur sont données sous la forme de vecteur de commandes. Ainsi, du point de vue du correcteur, l'information de sortie est déterminée par une combinaison d'informations en entrée et de l'état courant du système [82]. Cette approche permet à un correcteur de faire de la régulation, de l'asservissement, de la planification et de la supervision de processus.

Il existe deux grandes familles au sein du contrôle-commande : le contrôle-commande continu, et le contrôle-commande discret. Nous nous sommes intéressés exclusivement à cette deuxième famille que l'on désigne également par commande numérique. L'avantage du numérique est de pouvoir garder en mémoire les commandes effectuées et les états observés précédemment. Le contrôle-commande par calculateur permet ainsi de définir des lois de commandes de plus en plus sophistiquées et facilement ajustables afin d'améliorer les performances globales. Son développement est croissant dans de nombreux secteurs d'activité ; seules les applications qui nécessitent de grandes vitesses de traitements ne lui sont pas adaptées [83].

Pour effectuer une commande numérique, le processus à contrôler doit être discret, ou être discrétisé. Nous avons étudié les types de contrôleurs discrets les plus courants. De manière générale, les régulateurs numériques utilisent un certains nombres de termes mémorisés et de coefficients de pondération r_i t_j s_l pour atteindre les performances imposées [83]. Le contrôle est défini par l'équation récurrente suivante :

$$U(k) = \sum_i r_i.u(k-i) + \sum_j t_j.y_c(k-j) - \sum_l s_l.y(k-l)$$

avec

u(k)	: signal de commande envoyé par le calculateur
u(k – 1)	: commande précédente
y(k), y(k – 1)	: sorties
$y_c(k)$, $y_c(k-j)$	: trajectoires de consigne

Ainsi l'avantage du contrôle numérique est de permettre la réalisation de structures nouvelles

très générales dont les correcteurs PID (Proportionnel, Intégral, Dérivé) par exemple ne sont que des cas particuliers. Mais le contrôle numérique ne se cantonne pas uniquement à la définition d'algorithmes numériques. Nous nous sommes ainsi intéressés à un cas particulier de systèmes experts qui se base sur un ensemble de données floues pour réaliser des contrôleurs par la logique floue [84, 85]. L'intelligence artificielle et les techniques qui lui sont associées permettent de définir des commandes pour des processus évoluant dans un environnement imprévisible, ou pour un processus difficile à décrire avec des équations mathématiques (soit parce qu'on ne possède pas une connaissance suffisante du système, soit parce que le système est trop complexe). Un contrôle basé sur un système expert permet de prendre des décisions de manière plus ou moins autonomes pour un contrôle de haut niveau [84].

La nécessité de modéliser apparaît lorsqu'on l'on a besoin de contrôler, quantifier et comparer. Un modèle est une abstraction du monde réel qui permet de maîtriser des systèmes complexes. C'est une entité qui sert d'objet d'imitation pour faire ou reproduire quelque chose. En automatique, les modèles de processus sont de deux types : les modèles de connaissance et les modèles de représentation. Le premier type est basé sur des relations mathématiques déduites des lois de la physique qui régissent le fonctionnement du processus. Le second type est utilisé quand il n'y a pas d'analyse interne du processus (car trop complexe, impossible ou inaccessible). Il se base alors sur des mesures expérimentales pour établir la relation mathématique qui correspond le mieux au système. Le processus à contrôler peut être modélisé par : un bloc diagramme, des équations différentielles, des équations d'état linéaires ou non, des fonctions de transfert…

Le but de nos travaux a été de rapprocher l'automaticien et l'informaticien en proposant des abstractions logicielles réutilisables, flexibles et fiables pour le contrôle et la commande de processus en exploitant la technologie des objets. L'automaticien a besoin de modéliser, d'identifier, de simuler, de linéariser, de contrôler, de gérer et de documenter des projets de plus en plus complexes. L'approche objet a le potentiel pour faciliter certaines de ces tâches en intégrant le savoir-faire des concepteurs logiciels dans la définition et le développement d'outils pour l'automatique. Les principaux objectifs que nous avons poursuivis sont :

- Donner aux utilisateurs des outils simples, réutilisables et facilement adaptables pour

commander des systèmes.

- Ne laisser visible que des abstractions qui peuvent être manipulées facilement par des non-informaticiens.

- Pouvoir réaliser plusieurs contrôleurs complexes et les synchroniser.

- Fournir des services qui permettent d'ajouter des contraintes de sécurité, de faire de l'archivage, de la simulation, etc.

En partant de la problématique des systèmes et de leur commande, la recherche, l'identification et la définition d'abstractions génériques à objets et leurs relations ont permis de proposer une modélisation objet des mécanismes de comportements et des propriétés de ces systèmes. Des patterns de contrôle-commande sont le résultat de cette recherche. Ils permettent de véhiculer des solutions de conception, de l'expertise et du savoir-faire en matière de commande de processus, tout en introduisant les avantages de ces artefacts de réutilisation dans le domaine du contrôle-commande. Nous avons ainsi proposé des patterns pour l'abstraction des processus à commander, des patterns pour la commande classique ou floue et pour la commande à événements discrets. Finalement la définition d'une structuration en couches et la création d'applications génériques réutilisables, flexibles et à caractères intégrateurs forme l'aboutissement de nos travaux.

Rappelons que les patterns sont une tentative de formalisation de savoir faire indépendamment des spécificités de réalisation. Les patterns peuvent être vus comme des guides qui véhiculent de l'expertise pour inspirer d'autres analystes, concepteurs ou programmeurs. Ces microarchitectures formées de plusieurs classes ou objet interconnectés offrent des éléments réutilisables plus abstraits que de simples classes ou objets. Les patterns ne sont pas implémentés ce qui permet de laisser une grande liberté d'adaptation au programmeur. Les patterns assurent ainsi un transfert de connaissance, mais finalement ne favorisent pas une réutilisation opérationnelle.

Les frameworks quant à eux sont des applications génériques, des architectures – semi-finies – qui fournissent à la fois une réutilisation architecturale et une réutilisation au niveau de l'implémentation. Ils permettent de créer rapidement des applications ou des sous-systèmes

flexibles, robustes et extensibles par spécialisation. Leur granularité est plus importante que les patterns puisqu'ils permettent l'instanciation d'applications complètes. Toutefois, le revers de la médaille est la complexité de leur architecture qui induit un temps d'apprentissage long. Les patterns peuvent par contre être utilisés pour documenter les mécanismes des frameworks. La figure suivante illustre comment patterns et frameworks se complètent pour supporter un effort de réutilisation.

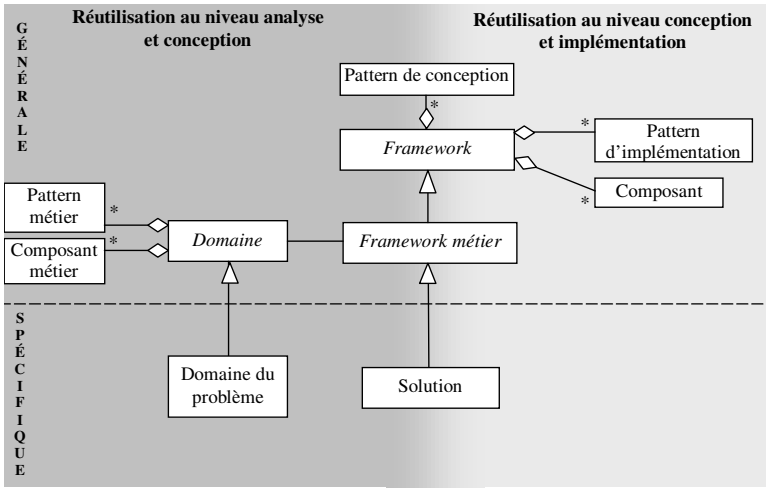

Figure 4 : Metamodélisation des artefacts de réutilisation.

Cette metamodélisation des artefacts de réutilisation intitulé *Idiomes, Patterns, Frameworks, un tour d'horizon* [39] a été présentée aux journées COOSI de l'AFCET en mai 1997. Nathalie Gaertner a également publié des résultats *(A framework for* fuzzy *knowldge based control* [86]*)* dans la revue *Software: Practice and Experience,* avec Bernard Thirion qui co-encadrait la thèse.

Les conclusions de nos travaux sur les patterns et les frameworks ont été reprises dans un chapitre de l'ouvrage *Modélisation Objet avec UML* [2]. La figure suivante est une vue d'ensemble des patterns, des composants et des frameworks. Les trois axes choisis sont la granularité, l'activité et la dépendance vis-à-vis d'un domaine d'expertise particulier.

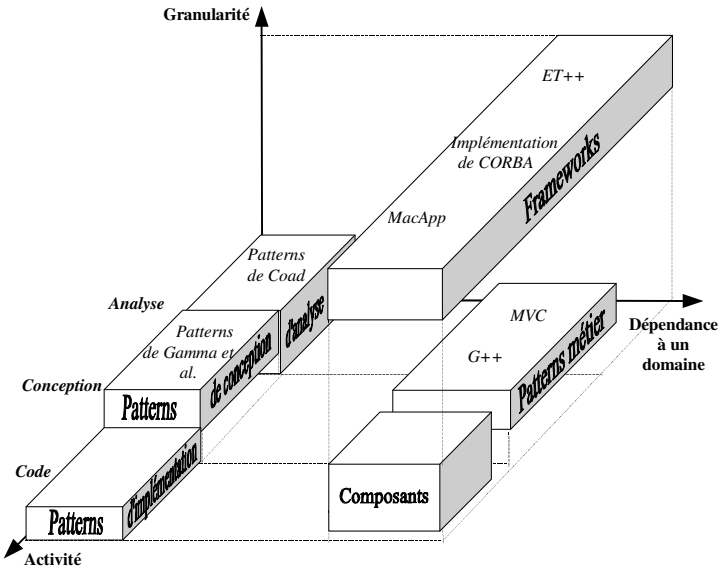

Figure 5 : vue d'ensemble des patterns, des composants et des frameworks

Nathalie Gaertner a été qualifiée aux fonctions de maître de conférences en 27ème et 61ème sections ; elle a préféré rejoindre la division Rational Software de IBM.

3.2 L'opérationnalisation des modèles

L'opérationnalisation des modèles consiste à promouvoir les modèles au rang d'artefacts de réalisation de premier plan. Un modèle opérationnel est un modèle qui contient suffisamment d'information pour qu'une implantation informatique raisonnable (par rapport à une implantation programmée manuellement) puisse en être dérivée de manière systématique.

A partir du début des années 2000, et notamment du fait de l'émergence de la notion de lignes de produits [41], des travaux sur la programmation générative [42], et de l'amélioration de la précision du formalisme d'UML, il devient envisageable de rendre les modèles UML plus productifs et de s'attaquer à la transformation systématique des modèles en artéfacts exécutables.

La validité industrielle de cette approche n'est cependant pas établie ; cela constitue un point bloquant pour la crédibilité et le déploiement de l'IDM. Nous estimons à cette époque que des expérimentations sont nécessaires pour faire avancer la discipline.

Nous allons procéder en trois temps : tout d'abord par une approche outillée pour le prototypage de modèles, puis par la réalisation de ponts entre le monde des modèles et l'espace technologique du Web, et enfin par une expérience en vraie grandeur de déploiement d'une solution d'ingénierie Web entièrement pilotée par des modèles.

3.2.1 Vers des techniques pour le prototypage de modèles UML

Le déploiement d'UML se heurte aux difficultés rencontrées par les utilisateurs. Ces difficultés sont de différentes natures, certaines concernent la démarche proprement dite, d'autres concernent la sémantique des modèles.

Le point dur auquel nous sommes confrontés, est la signification des modèles construits par les utilisateurs. Les utilisateurs fabriquent des modèles, utilisent des éléments de modélisation, mais ne font pas nécessairement le lien entre leurs choix de modélisation (leurs assemblages d'éléments de modélisation) et les implications de ces choix en termes d'implémentation informatique.

Nous participons à cette époque à de nombreuses actions d'audit sur des modèles UML, et très rapidement nous identifions des problèmes récurrents souvent détectables quasiment syntaxiquement. Il nous semble qu'une part importante de ces problèmes pourrait être levée si les utilisateurs avaient dès la modélisation un retour tangible sur les implications de leurs choix de modélisation en termes de réalisation. On retrouve là d'ailleurs un principe cardinal des méthodes agiles : le « feed-back ».

A cet effet, nous allons travailler avec Philippe Studer (à l'époque ingénieur de recherche, aujourd'hui maître de conférences à l'UHA) sur un environnement de prototypage de modèles UML. Il s'agissait en fait d'un atelier de mise au point des diagrammes de classes, sous la forme d'un outil qui permettait de jouer des scénarios sur les instances correspondantes aux classes définies dans les diagrammes de classes. Par la suite, Frédéric Fondement (actuellement en fin de thèse à l'EPFL, j'assure son encadrement à 50 %) ajoutera au prototypeur de modèles un interprète OCL dans le cadre de son stage de DRT (que je co-encadrai avec Michel Hassenforder, professeur à l'UHA).

Des résultats ont été publiés avec Nasser Kettani (consultant chez IBM) dans la revue Rose Architect, sous le titre *Validating UML Models Through Prototyping* [87] et j'ai été invité par la suite à présenter nos travaux lors de la conférence IEEE RSP'2000 *An example of UML model level Prototyping* [88].

Le prototypeur de modèles est notre première contribution significative à l'opérationnalisation des modèles. Paradoxalement, la principale critique qui nous a été faite, était de ne fabriquer que des prototypes et pas des applications réelles à partir des modèles ! Nos travaux ultérieurs seront tous axés sur la génération d'artéfacts opérationnels et incorporables dans des applications réelles.

3.2.2 Réalisation de ponts vers le monde XML

L'absence ou l'immaturité des outils spécifiques pour la manipulation de modèles (éditeurs, référentiels, transformateurs, tisseurs…) freine le déploiement de l'IDM.

Le développement de nouvelles technologies présente un risque important, d'où l'intérêt de réutiliser le plus possible d'éléments existants, quitte à construire des ponts vers d'autres

espaces techniques. Les techniques de transformations d'arbres XML bénéficient a cette époque d'une large diffusion du fait de leur application à l'Internet qui est en plein essor (nous sommes avant l'effondrement de la bulle).

Nous décidons de poursuivre la piste des ponts entre le monde des modèles et l'univers XML. Nous avons commencé à travailler dans cette thématique dès 1998 notamment avec Olivier Burgard, d'abord en stage ingénieur, puis en DEA, et enfin pendant 3 ans sous contrat d'ingénieur. A partir de sérialisations XML de modèles UML (restreints aux aspects statiques) nous avons étudié comment manipuler ces sérialisations à l'aide d'outils disponibles dans l'espace technique XML. Nous avons tout d'abord montré dans la communication *Représentation d'un modèle UML codé en XMI* [89] aux journées GRACQ que la représentation d'UML en XMI (*XML Metadata Interface*) posait un problème non trivial de compatibilité entre les différentes versions d'UML, d'XMI et de MOF (*Meta Object Facility*). Dans ce contexte, nous avons présenté une approche de génération des classes Java pour la sérialisation de modèles à partir de leurs metamodèles à la conférence OCM'2000.

La représentation de modèles UML sous la forme d'un format XML (XMI) nous a également amené à expérimenter la visualisation de modèles dans Internet Explorer, au moyen de feuilles de styles XSLT (un langage de transformation pour XML). Nous avons présenté ces travaux aux journées COOSI de l'AFCET, sous le titre *Représentation d'un modèle UML codé en XMI dans Internet Explorer 5* [90] en septembre 1999. Il existe aujourd'hui des plug-ins SVG pour Internet Explorer, qui nous semblent être une meilleure approche.

Nous avons également appliqué XSLT à la génération de code JSP et PHP depuis un modèle dérivé d'UML, ces travaux ont été présentés aux Journées du CRESPIM'02, sous le titre *Vers une programmation unifiée avec XSLT : applications aux serveurs Web* [91]. A l'usage, XSLT s'est avéré assez mal adapté pour des transformations de modèles complexes (c'est-à-dire allant au-delà des problématiques de transcodage syntaxique) car il ne permet pas de travailler au niveau de la sémantique des modèles manipulés, mais seulement à celui d'un arbre couvrant le graphe de la syntaxe abstraite du modèle.

3.2.3 A la recherche d'une expérience significative

Les physiciens et les chimistes valident leurs théories par des expériences. Les chercheurs en informatique font de même en écrivant des programmes qui deviennent leurs expériences.

Il n'est pas aisé pour qui se préoccupe de génie logiciel de trouver une expérience à sa mesure. En effet, la pertinence des techniques de génie logiciel ne se manifeste que lorsque la taille de l'objet d'étude devient significative. Le passage à l'échelle est une préoccupation constante de nos travaux et il n'est pas facile de trouver un cadre d'expérimentation.

Comme je ne trouve pas de cadre industriel pour expérimenter dans le domaine des modèles opérationnels, car personne ne veut prendre de risques significatifs, je décide de profiter du nouveau statut offert par la nouvelle loi sur l'innovation, et de fabriquer mon expérience sous la forme d'une start-up.

La loi sur l'innovation et la recherche du 12 juillet 1999 favorise, par un ensemble de dispositions, le transfert de technologies de la recherche publique vers l'économie et la création d'entreprises innovantes en se donnant pour but d'instaurer un cadre juridique conciliant à la fois la participation des personnels de la recherche publique à la création et au développement d'entreprises, le fonctionnement régulier des services employant ces fonctionnaires, et la moralité de leurs comportements.

L'Université de Haute-Alsace (UHA) a fait de cette forme de valorisation une priorité (son objectif est de soutenir la création d'une entreprise innovante par an), et m'a encouragé à monter un projet d'entreprise pour valoriser mon expérience de la modélisation objet.

De 1999 à 2002, j'ai été placé en délégation dans la société ObjeXion Software SA dont j'ai assuré la présidence et la direction générale; j'ai encadré 1 spécialiste du marketing (ingénieur + MBA), 1 ingénieur commercial, 7 docteurs ou ingénieurs informaticiens, 1 assistante. La start-up a été créée avec le soutien de l'Université de Haute-Alsace, de l'ANVAR et de la société de capital risque Alsace-Création. L'objectif était d'apporter aux informaticiens un savoir-faire de modélisation sous la forme d'outils (plutôt que de classiques actions de conseil) pour rendre leurs modèles informatiques plus opérationnels, en automatisant la génération de systèmes d'information à partir des modèles.

Dans le contexte industriel de l'époque (avant l'implosion de la bulle Internet), nous avons choisi le Web comme champ d'application et nous nous lançons dans le développement et la commercialisation de technologies d'ingénierie dirigées par les modèles dédiées à la réalisation de systèmes d'information accessibles par Internet au moyen de navigateurs.

Ces applications se décomposent typiquement en plusieurs composants qui s'exécutent simultanément du coté serveur (sur un serveur de base de données, un serveur de fichiers et un serveur http) et du coté client (par l'exécution de scripts dans les navigateurs).

La réalisation d'une application de ce type, demande la collaboration de différents profils d'intervenants : typiquement des spécialistes du marketing, des analystes métiers, des graphistes, des informaticiens et des producteurs de contenu.

Cette variété de profils entraîne une variété de démarches de développements et de techniques, ce qui génère de nombreuses contraintes d'organisation pour le pilotage d'un tel projet. De plus, la jeunesse du domaine (les développements Web), et l'immaturité des technologies alliée à l'absence d'une réelle compréhension des enjeux par les praticiens, conduit également à une forte variabilité des choix technologiques, rendant difficile, voire impossible, toute démarche significative de capitalisation de savoir-faire.

L'ingénierie dirigée par les modèles peut permettre à ces projets Web de retrouver le chemin d'une démarche de génie logiciel, notamment par sa capacité à séparer les représentations métier des choix technologiques, tout en assurant une claire séparation des points de vues, au moyen de metamodèles spécifiques.

Dans ce contexte, la société ObjeXion a développé l'outil Netsilon, un AGL dédié à la réalisation de systèmes d'information Web [92]. Les différents points de vue sont modélisés séparément au moyen de trois metamodèles spécifiques, pour représenter respectivement le modèle métier (avec un sous-ensemble d'UML), la navigation et la composition (avec un metamodèle hypertexte) et la présentation (avec un metamodèle des artefacts Web). Un langage d'action, dénommé Xion, à la syntaxe concrète proche de celles d'OCL et Java, permet de spécifier le corps des méthodes et d'exprimer des conditions, et ainsi de faire le lien entre les trois points de vue de modélisation.

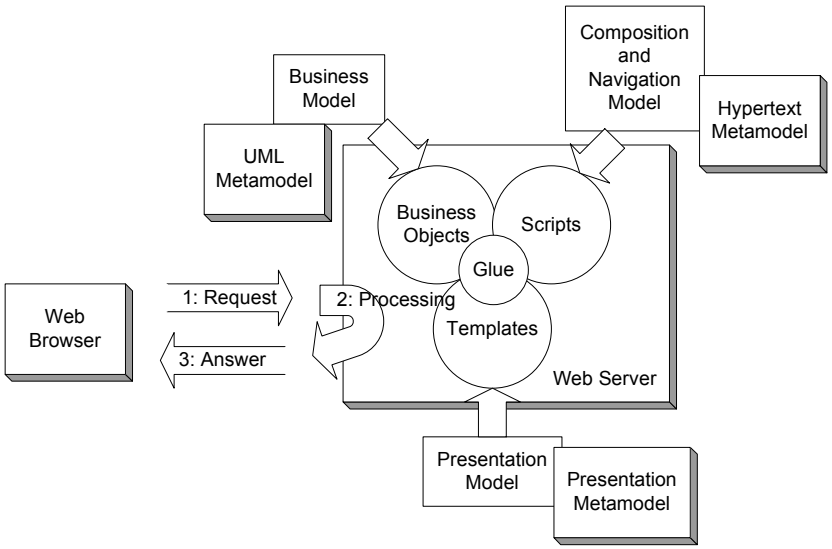

Figure 6 : Modélisation des différents points de vue avec Netsilon

Les spécificités de plateformes, c'est-à-dire le langage cible (Java Servlet ou JSP, ou PHP), le type de base de données (Oracle, MySQL ou PostgreSQL), ainsi que la configuration matérielle (nombre de serveurs, répartition des composants) sont spécifiées dans un modèle de déploiement et prises en compte lors de la phase de génération du code de l'application.

J'ai été amené à mettre en place une démarche qui s'inspirait fortement d'XP (*Extreme Programming*) et dans laquelle je me suis livré à la planification négociée, dans le rôle du client. Avec une équipe réduite (7 informaticiens) nous avons développé en deux ans, plus de deux millions de lignes de code Java parfaitement opérationnel. Autant que faire se peut nous avons essayé de suivre les principes d'XP, sans réussir totalement, notamment sans parvenir à limiter le stress et les dépassements d'horaires. Toutefois, et globalement, le projet a été un succès du point de vue de la réalisation d'un logiciel et la gestion d'une équipe de développement.

II est bien évidemment difficile de résumer une telle aventure en quelques lignes. Je présente une compilation des principaux enseignements dans les paragraphes suivants.

- Il y a une différence fondamentale entre la recherche et l'innovation. La recherche se préoccupe de concepts, l'innovation d'idées. La recherche élabore de nouvelles connaissances, l'innovation optimise les processus existants. Le laboratoire est le lieu

de la recherche, l'entreprise le lieu de l'innovation. J'ai eu tendance à confondre ces lieux, c'était une erreur.

- La recherche de financements est un emploi à temps plein, et il ne faut jamais se trouver en panne au milieu du gué. D'autre part, le montant recherché compte assez peu, ce qui compte c'est le temps et l'énergie à déployer pour trouver les fonds. En conclusion, autant rechercher une grande enveloppe, et si ce n'est pas possible, il faut changer de projet.

- Dans notre discipline, le génie logiciel, les espoirs de gains se trouvent plus dans les démarches que dans les outils.

- L'informatique n'est pas finale. L'informatique est toujours au service d'une autre activité, qui elle peut être finale. Malheureusement, la perception de valeur est liée à la finalité. II en découle que le laboratoire est le seul espace pour qui veut vivre l'informatique comme une finalité.

Je considère cette expérience comme très féconde d'un point de vue humain et scientifique, car elle m'a permit d'expérimenter en vraie grandeur toutes les dimensions de l'ingénierie dirigée par les modèles, dans ses aspects à la fois technique, économique et organisationnel.

3.2.4 Retour à la vie universitaire

Je réintègre mes fonctions de maître de conférences à l'UHA en septembre 2002. Je n'ai que peu de travaux publiés durant ma période de délégation en entreprise car la logique de dépôt de brevet imposée par les investisseurs est incompatible avec la démarche de publication. A partir de 2002, ayant récupéré ma liberté de parole, j'ai présenté diverses facettes de mes contributions sur les modèles opérationnels appliqués au Web, dans plusieurs média :

- Un article très général sur l'utilisation de Netsilon dans la revue française Internet Professionnel *Netsilon : Le développement web sans tracas* [92] en juillet 2002.

- Un retour d'expérience sur la mise en œuvre industrielle de l'IDM, *MDA : Une réalité, un rêve ? Retour d'expérience* [93], devant le groupe de travail de l'OFTA sur l'IDM, en octobre 2002.

- Une communication *Platform Independent Web Application Modeling* [94] au séminaire de Dagshtul sur *Language Engineering for Model-Driven Development*. Pour information, pour un universitaire allemand, la participation à un séminaire de Dagstuhl est une étape significative de la carrière ; elle est un indicateur d'excellence au sein d'une communauté scientifique.

- Une communication à la conférence internationale UML'03 sur la modélisation des applications Web, indépendamment des plateformes *Platform Independent Web Application Modeling* [95].

- Un article de fond sur le même sujet, de manière beaucoup plus détaillée, dans la revue internationale SoSyM, intitulé *Platform Independent Web Application Modeling and Development with Netsilon* [43]. Cet article figure en annexe de ce mémoire.

Avec William ElKaim (Thalès Research and Technology) et Thierry Cros (consultant chez TWAM) nous nous sommes intéressés à l'impact des techniques d'ingénierie dirigée par les modèles sur les habitudes de développement. Près de 50 ans après les débuts de l'informatique, les pratiques industrielles restent centrées sur le code, considéré comme étant le seul représentant fiable du logiciel. Poussé à son extrême (comme dans *l'Extreme Programming*) ce raisonnement conduit à rejeter toute production d'artefacts différents du code. L'opérationnalisation des modèles me semble un bon angle d'attaque pour dépasser ce point de vue. J'aborde ce point de vue dans la préface de l'ouvrage *Maîtriser les projets avec l'Extreme Programming*, de Thierry Cros et Ron Jeffries [96].

Nous avons également présenté une démarche dirigée par les modèles pour générer des lignes de produits à la conférence internationale GL'01 intitulée *MDA Compliant Product Line Methodology, Technology and Tool for Automatic Generation and Deployment of Web Information System [97]*. Une version étendue a ensuite été publiée dans la revue française Génie Logiciel, sous le titre *Outils, Méthodologies et Technologies de produits MDA pour la génération et le déploiement automatique de systèmes d'information Web* [98].

Dans la communication *Model Driven Architecture for Agile Web Information System Engineering [44]* à la conférence internationale OOIS'03 je détaille comment l'utilisation d'un environnement comme Netsilon permet d'injecter de l'agilité dans les développements

de systèmes d'information Web. Je montre notamment que tous les avantages des méthodes agiles peuvent être retrouvés dans un processus dirigé par les modèles, dès lors que des transformations systématiques sont disponibles pour remplacer la production manuelle du code.

En parallèle de cet effort de communication sur les travaux menés au sein de la start-up, et dans la suite des travaux menés dans le contexte de la thèse de Nathalie Gaertner, nous avons travaillé avec Didier Bresch (maître de conférences à l'Université de Haute-Alsace) sur la définition et l'outillage d'un metamodèle pour la réalisation d'application de commande-contrôle, dans un environnement à très faible coût à base de microcontrôleurs. L'idée est d'utiliser une chaîne de transformations qui permet de passer d'une description orientée composants électroniques vers un assemblage automatique de composants programmés préalablement en assembleur ou en langage C. Ces composants ont été utilisés pour de nombreuses réalisations lors de travaux pratiques (asservissement de position d'un télescope, pilotage d'un appareil photo, domotique...). Nous avons présenté l'application de cette approche à l'enseignement à UML 2004 [99] et au congrès CETSIS'05 [100]. Nous poursuivons toujours notre collaboration, nos travaux actuels portent sur la qualification plus précise des caractéristiques d'assemblage des composants décrits précédemment et sur liens avec les langages de description d'architecture (en collaboration avec Olivier Barrais, maître de conférences à Rennes).

3.3 Modèles et ingénierie des langages

L'arrivée du MDA en 2000 entraîne la promotion d'une architecture de metamodélisation basée sur le MOF (Meta-Object Facility) et met en lumière que l'ingénierie dirigée par les modèles est plus centrée sur les metamodèles que sur les modèles. UML perd sa position centrale, au profit de MOF, et se retrouve à égalité avec d'autres langages de modélisation comme CWM, SPEM, AADL, SysML…

On constate aujourd'hui une prolifération de metamodèles, soit pour décrire différentes facettes d'un même système, soit pour dénoter différents niveaux d'abstraction (modèles conceptuels, modèles logiques, modèles physiques…).

Ce glissement vers les metamodèles a aussi pour conséquence la création de nombreux nouveaux langages spécifiques à l'IDM (par exemple OCL pour l'expression de contraintes, Action Semantics pour les actions, QVT pour les transformations, XMI pour les sérialisations…). Il en résulte une sévère problématique d'intégration de ces langages qui tendent à disperser les concepts de manipulation de modèles.

Nous prenons acte de ce changement de perspective et montrons comment les metamodèles, les modèles et les données sont autant de leviers pour piloter le paramétrage des applications métier, dans la communication *A propos des niveaux de modélisation dans l'approche MDA, un exemple de paramétrage d'application* [101]. Avec d'autres collègues français (Jean Bézivin, Sébastien Gérard et Laurent Rioux) nous posons également les bases de la notion de composants MDA (*MDA components: Challenges and Opportunities* [102]). De tels composants définissent des unités de regroupement pour tout artefacts MDA (en fait le travail est généralisable à tout composant IDM) produits dans une démarche reliée au MDA, y compris cette démarche elle-même.

3.3.1 Initiative TopModL

TopModL a été notre première réaction aux problèmes induits par la prolifération des langages, metamodèles et des modèles, et par la diversité technologique (foisonnement de plateformes, de services, de normes…). Les personnes suivantes ont participé aux réunions de réflexion : Sébastien Gérard et Benoît Baudry du CEA-LIST, Laurent Rioux de Thalès, Jean-Marc Jézéquel de l'INRIA, Cédric Dumoulin du LIFL, Jérôme Delatour de l'ESEO, Michel

Hassenforder du MIPS, Thomas Baar et Frédéric Fondement de l'EPFL, Colin Atkinson de l'université de Manheim et Thomas Kuehne de l'université de Darmstadt. J'ai assuré la coordination des travaux du groupe.

Notre objectif a été de contribuer à doter l'IDM de moyens pour contrôler la définition et la fusion de metamodèles, et notamment pour résoudre les problèmes durs de recouvrement de domaines sémantiques. Nous avons porté notre effort sur la définition et l'outillage de techniques rigoureuses pour :

- comparer les metamodèles (et les modèles)
- prouver l'équivalence des metamodèles (ou des modèles),
- transformer les metamodèles (et les modèles) pour passer d'un metamodèle (d'un modèle) à l'autre ou raffiner un même modèle,
- intégrer des modèles pour créer de nouveaux modèles par composition ou par tissage.

TopModL nous a servi de cadre pour réfléchir à ce que pourrait être une plateforme de recherche pour l'IDM. Les points clés retenus par TopModL sont les suivants :

- le fait que tout est modèle,
- les notions de langages, modèles et rôles,
- le fait que TopModL lui-même est un modèle,
- le fait que tout est explicite, y compris l'architecture de metamodélisation,
- l'indépendance par rapport aux référentiels.

TopModL a permis de mieux cerner le périmètre d'un environnement de metamodélisation, et notamment de mieux cibler les technologies sur lesquelles bâtir une plateforme. D'un point de vue pratique, nous avons mis au point une chaîne de production, basée sur des transformations de modèles (exécutées avec le moteur de transformation MDA Transf de Cédric Dumoulin du LIFL), qui assure effectivement l'indépendance vs. EMOF et Ecore, ainsi qu'envers les dépositaires (tels MDR ou EMF), et aussi développé un éditeur graphique pour modéliser les metamodèles au moyen d'une notation dérivée d'UML.

Les principaux enseignements obtenus lors du développement de la première version de la

plateforme sont les suivants :

- Les avantages de se rendre indépendant du niveau M3 (tel que définit par l'OMG) ne sont pas contrebalancés par les inconvénients de créer un nouveau langage M3 pivot.
- Il est plus judicieux de rechercher la compatibilité maximale avec un langage de metadonnées existant qui dispose déjà d'un grand nombre d'outils.
- Il est difficile de trouver une source de financement pour un projet sur la metamodélisation en générale, il faut rechercher un domaine d'application (ce que je ferai dans le cadre du montage du projet RNTL OpenEmbeDD).

Les motivations et principes de base de TopModL *The TopModL Initiative* ont été présentés à WISME'04 et publiés dans les actes des événements satellites de la conférence internationale UML'04 [103].

L'initiative TopModL peut aujourd'hui être considérée comme un creuset d'idées ayant aboutie d'une part à un prototype, et d'autre part à la spécification d'un environnement pour la metamodélisation, dont une implémentation concrète est Kermeta dont je parlerai largement dans la section suivante.

3.3.2 Le système de metamodélisation Kermeta

« Another lesson we should have learned from the recent past is that the development of "richer" or "more powerful" programming languages was a mistake in the sense that these monstrosities, these conglomerations of idiosyncrasies, are really unmanageable, both mechanically and mentally. I see a great future for very systematic and very modest programming languages. » - E.W. Dijkstra (1972)

Le projet Kermeta est notre principale contribution à l'ingénierie dirigée par les modèles. Kermeta adresse la problématique de l'explosion des metamodèles et de l'hétérogénéité des langages dans la sphère IDM, en offrant les moyens de :

- formaliser les metamodèles,
- formaliser les interactions entre metamodèles,

- utiliser les metamodèles pour décrire des langages,

le tout au sein d'un système de metamodélisation homogène et fortement outillé.

Kermeta est le résultat d'un travail collectif, le fruit d'une communauté open-source (www.kermeta.org) qui s'est structurée initialement autour de l'équipe Triskell de l'Irisa à Rennes, et qui regroupe aujourd'hui des contributeurs de divers horizons.

Nous avons démarré cet effort avec l'aide de plusieurs membres de Triskell, et notamment de Franck Fleurey dont j'ai co-encadré les travaux de thèse *Langage et méthode pour une ingénierie des modèles fiables*. Nous avons unifié nos travaux précédents sur l'opérationnalisation des modèles (TopModL, AGL Netsilon pour le Web et Model Prototyper, langage d'action Xion et de transformation MTL) dans le cadre de la définition d'un environnement de metamodélisation (le projet Kermeta) construit autour d'un langage de metamodélisation exécutable (le langage Kermeta).

Nous appelons *langage de metamodélisation exécutable*, un metalangage qui se distingue d'un langage de metadonnées (comme MOF qui n'est pas exécutable) par sa capacité à décrire tous les aspects de la modélisation d'un langage au moyen de metamodèles, y compris la sémantique (exprimée dans notre cas sous une forme opérationnelle, exécutable par une machine virtuelle).

Le langage Kermeta est ainsi un langage de metamodélisation exécutable, qui intègre metadonnées et besoins d'exécutabilité partagés par les langages de transformation, d'action et de contraintes. Conformément aux enseignements tirés de l'initiative TopModL, nous nous sommes attachés à rendre ce langage le plus largement compatible avec les approches actuelles de metamodélisation (comme EMOF, Ecore...) et leurs outils associés.

D'un point de vue technique, le langage Kermeta est construit par composition d'aspect en tissant l'exécutabilité dans le metamodèle de EMOF. Cette approche générative nous permet principalement de bénéficier de tout l'outillage qui s'applique à EMOF (notamment dans l'environnement Eclipse).

Une présentation du langage Kermeta et de sa construction par tissage d'aspect *(Weaving executability into object-oriented metalanguages [45])* a été réalisée lors de la conférence

internationale MoDELS en octobre 05. Cet article figure en annexe de ce mémoire.

Ce travail s'inscrivait originellement dans la perspective d'une ANRD (Action Nationale de Recherche et Développement) sur l'IDM, dans un effort de convergence entre les travaux de l'initiative TopModL, et les projets TopCaseD (emmené par Airbus), et MDDi (auxquels participent Thalès, l'INRIA et le CEA dans la cadre de la convention CAROLL). Cet effort de convergence s'est finalement concrétisé dans une réponse à l'appel à projets du RNTL 2005, sous la forme d'une participation à la plateforme open-source OpenEmbeDD dont je suis le coordinateur.

3.3.2.1 Principales limitations des solutions actuelles de metamodélisation

Dans la communauté de l'ingénierie dirigée par les modèles, les metalangages à objets tel que MOF, EMOF ou Ecore sont de plus en plus utilisés pour spécifier des metamodèles. Ces langages offrent les concepts de classes, opérations, attributs et relations, mais sans support pour la description des comportements qui sont souvent exprimés en langage naturel. Fondamentalement ce sont des langages de metadonnées, basés sur les objets. A titre d'exemple, voici la définition de l'opération *isInstance* de la classe *Type* de EMOF (section 12.2.3 page 34) :

Operation ***isInstance(element : Element) : Boolean***

"*Returns true if the element is an instance of this type or a subclass of this type. Returns false if the element is null*".

Ce genre de description en langage naturel est difficilement automatisable. De ce fait, lorsque la précision d'expression est nécessaire, il faut avoir recours à l'utilisation d'autres langages qui peuvent être très différents les uns des autres : plutôt impératifs (Java par exemple), plutôt déclaratifs (OCL par exemple [15], voire hybrides dans certains cas (QVT par exemple [104]). Chacun de ces langages présente des caractéristiques intéressantes mais aucun d'entre eux ne peut être réutilisé tel quel pour faire un langage de metamodélisation exécutable.

Java présente l'avantage d'être largement connu des développeurs et dispose d'un grand ensemble d'outils et bibliothèques. Son principal inconvénient est d'être un langage de programmation généraliste, auquel manque des concepts de premier ordre de la modélisation,

telles que les associations ou les propriétés dérivées.

OCL a été initialement conçu pour exprimer des contraintes sur des modèles UML et il existe aujourd'hui une version d'OCL liée à MOF qui permet d'exprimer des contraintes sur des metamodèles. Le langage OCL inclut des expressions et constructions particulièrement adaptées à la navigation dans les modèles. Cependant, OCL étant un langage de contraintes, aucune construction OCL ne permet de créer ou de modifier des modèles. OCL ne peut donc pas être utilisé directement comme langage d'action mais une grande partie des expressions OCL peut être réutilisée au sein d'un langage d'action. Dans cette optique, un langage d'action peut être vu comme une extension à OCL permettant les effets de bord nécessaires à la modification de modèles.

L'Action Semantics (AS) est défini comme le langage d'action d'UML. De la même manière que OCL a migré de UML vers MOF pour être utilisé sur des metamodèles, on pourrait isoler la partie de l'AS applicable à MOF et l'utiliser comme langage d'action. Cette solution est cependant difficile à mettre en œuvre car l'AS n'est pas un langage destiné à un utilisateur manipulateur de metamodèle, mais la spécification d'une machine virtuelle permettant l'exécution d'actions au sein de modèles UML. De ce fait, les concepts présents dans l'AS sont de trop bas niveau par rapport à ce que l'utilisateur est prêt à manipuler pour la définition de metamodèles.

QVT est le langage de transformation de modèles normalisé par l'OMG. QVT est composé d'un cœur déclaratif basé sur des règles dédiées à la transformation de modèles et propose une extension impérative permettant d'implanter des règles de transformations. Les concepts présents dans QVT sont trop spécifiques à la transformation de modèles pour pouvoir être réutilisés dans un langage d'action.

Le fait d'utiliser des langages différents pour définir les différents artefacts relatifs à un metamodèle (c'est-à-dire la structure, les contraintes, la sémantique et les transformations) pourrait présenter l'avantage de pouvoir choisir pour chaque aspect un langage parfaitement adapté. Cependant, le fait que ces langages soient aujourd'hui complètement indépendants présente deux inconvénients majeurs. Premièrement, cela empêche les interactions entre les différents artefacts ; en effet, si les contraintes associées au metamodèle sont en OCL et que la

sémantique est exprimée en langage naturel, il est difficile de faire appel aux contraintes ou aux éléments sémantiques dans une transformation de modèles écrite en Java ou QVT. Deuxièmement, cela rend très difficile toute vérification relative à la cohérence entre ces différents artefacts.

3.3.2.2 Vers la conception d'un nouveau metalangage

Afin de remédier aux problèmes décrits plus haut, nous avons défini un noyau de metamodélisation pour spécifier à la fois la structure et la sémantique des metamodèles, et permettre ainsi de palier les limitations de cohérence et d'interopérabilité liées à l'hétérogénéité des langages décrits plus haut.

L'idée générale, illustrée Figure 7, est de factoriser les concepts communs aux différents langages du domaine de l'IDM, dans un cœur contenant des éléments permettant de spécifier des metamodèles, leur sémantique opérationnelle mais aussi d'instancier ces metamodèles ou de réaliser des transformations de modèles. L'objectif de ce cœur n'est pas de remplacer l'ensemble des langages dédiés disponibles dans l'univers de l'IDM, mais de fournir une base sémantique commune à ces langages afin de leur permettre d'interopérer naturellement. En pratique, ce cœur se présente sous forme d'un langage de metamodélisation appelé Kermeta et d'une machine virtuelle répondant aux besoins spécifiques de la manipulation de metamodèles et de modèles.

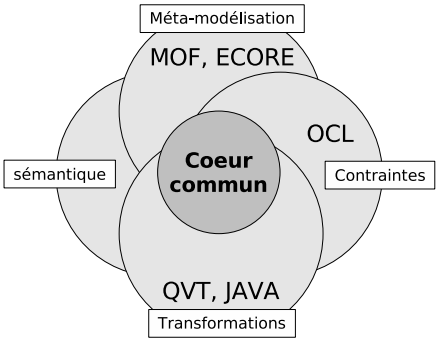

Figure 7 : factorisation des notions communes aux langages de l'IDM

L'intérêt principal de cette approche est de définir les différents artefacts relatifs à un metamodèle de façon homogène. Un metamodèle peut par exemple encapsuler directement les contraintes qui lui sont attachées et la spécification de sa sémantique opérationnelle. Le

noyau de metamodélisation proposé se présente sous la forme d'un metalangage objet construit comme une extension de EMOF. L'implantation de ce langage a été baptisée *Kermeta* (par contraction de *Kernel Meta*).

Les langages de metamodélisation existants, tel que MOF, EMOF, CMOF ou Ecore se basent tous sur les concepts de packages, de classes et d'associations, et diffèrent par des variantes pour la définition de chacun de ces concepts. Malgré ces différences, l'ensemble de ces langages présente un pouvoir d'expression sensiblement équivalent: il est toujours possible de traduire les concepts présents dans un langage donné en utilisant les concepts d'un autre langage. A titre d'exemple, les langages CMOF et MOF 1.4 permettent la définition de classe-associations, ce qui n'est pas le cas de EMOF ou Ecore. Cependant, il est aisé de traduire une classe-association dans un langage qui ne les supporte pas nativement, en utilisant une classe et deux associations.

Dans l'optique de construire un noyau de metamodélisation, de taille volontairement réduite, il est logique de partir du langage de metadonnées qui définit l'ensemble de concepts (structurels) le plus compact puis de l'enrichir par un langage d'action pour apporter l'exécutabilité. Cette approche par extension présente en outre deux avantages majeurs : le premier est qu'elle permet de ne pas risquer de régresser par rapport à l'existant tout en laissant une grande liberté en ce qui concerne les extensions ; le second est que le nouveau langage peut être rendu compatible par construction avec le langage étendu et donc avec les outils existants.

En ce qui concerne les aspects structurels, nous avons ainsi choisi EMOF qui a été défini par l'OMG comme le langage ne contenant que les concepts "essentiels" à la metamodélisation. Le fait de réutiliser EMOF comme base présente également l'intérêt qu'un grand nombre d'outils le supportent du fait de sa normalisation. C'est le cas notamment des outils Eclipse qui, bien que basés sur le langage Ecore, supportent l'import-export des modèles EMOF.

Pour ce qui est de la dynamique, des possibilités très différentes peuvent être envisagées pour compléter un langage de metadonnées par des actions. Par exemple, si l'on souhaite utiliser des machines à états comme langage d'action, on pourrait décider que le comportement de chaque classe EMOF devrait être spécifié par une machine à état. En fait nous avons trouvé

que la principale difficulté de conception est d'ajuster le compromis entre minimalité et utilisabilité. D'un côté, il est nécessaire d'introduire des concepts et des constructions dédiés à l'IDM qui rendent facile la manipulation de modèles, et de l'autre, il faut prêter attention à introduire un minimum de concepts pour conserver l'esprit de noyau de metamodélisation. Il est également difficile de fixer la limite de ce qui appartient à la modélisation et de qui relève plus de la programmation, notamment les mécanismes d'entrées-sorties.

Par ailleurs, nous avons choisi un style de langage orienté-objet, impératif et statiquement typé. Le choix de réaliser un langage à objets est motivé d'une part par le fait que les langages de metadonnées que nous utilisons sont déjà fondés sur des principes objets, et d'autre part par la capacité des langages à objets à permettre la réalisation et la maintenance de systèmes complexes [105] [106] [107]. Le choix d'imposer un langage statiquement typé facilite un maximum de vérifications statiques et est, quant à lui, motivé notamment pour fiabiliser les transformations de modèles.

3.3.2.3 Fabrication de Kermeta

Kermeta réutilise les constructions d'OCL pour la navigation et les expressions et ajoute des constructions permettant la modification de modèles. La Figure 8 présente la hiérarchie d'héritage entre les différents types d'expressions Kermeta.

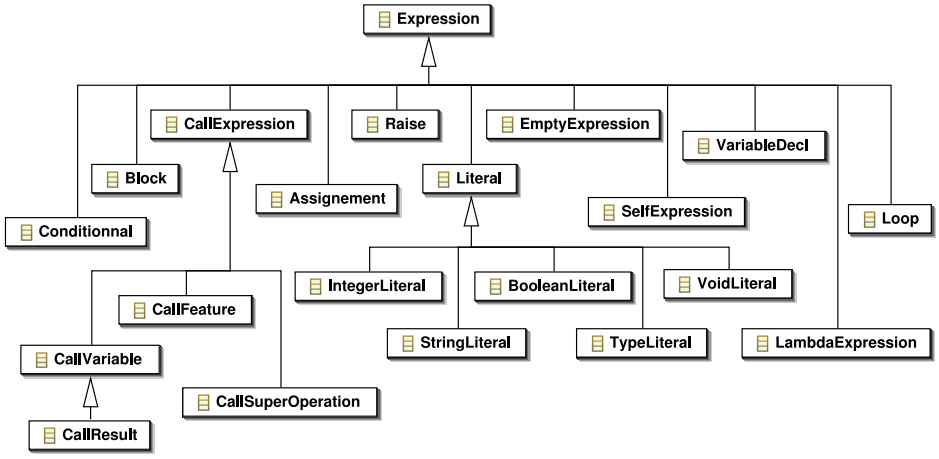

Figure 8 : différents types d'expressions Kermeta

Les principales constructions de ce langage d'action sont:

- Des structures de contrôle classiques comme les boucles (*Loop*), les conditionnelles (*Conditional*) et les blocs (*Block*). Nous avons exclu du langage toutes constructions telles que *break* ou *goto* car elles brisent le flot de contrôle ce qui tend à rendre les programmes difficiles à comprendre et à maintenir.

- La déclaration de variables locales (*VariableDecl*). Étant donné que nous souhaitons réaliser un langage statiquement typé, les variables locales doivent être déclarées avec leur type.

- Des expressions permettant de lire les variables locales et les propriétés ainsi que d'appeler des opérations (*CallExpression* et ses sous-classes).

- Des affectations permettant d'affecter les variables locales et les propriétés des objets (*Assignement*).

- Des expressions littérales pour les types primitifs et les types (*Literal* et ses sous-classes).

- Un mécanisme d'exceptions permettant le traitement des erreurs. L'expression *Raise* permet de lever une exception.

- Une forme limitée de lambda-expressions permettant l'implantation d'itérateurs analogues à ceux d'OCL tel que *select*, *reject* ou *collect*.

La Figure 9 schématise le processus de construction du langage Kermeta et sa promotion au rang de metalangage (ie. au niveau M3). On retrouve sur cette figure les deux niveaux de metamodélisation M2 et M3 de la traditionnelle pile méta. Pour construire l'environnement de modélisation Kermeta, c'est-à-dire l'environnement dans lequel le meta-metamodèle est Kermeta (à droite sur la Figure 9), nous utilisons un environnement de modélisation EMOF (à gauche sur la figure). Dans cet environnement, les metamodèles sont exprimés en EMOF. Étant donné que nous souhaitons construire Kermeta comme une extension de EMOF, les modèle initiaux du processus de construction de Kermeta sont d'une part EMOF (exprimé en EMOF) et d'autre part le metamodèle d'actions présenté plus haut (lui aussi exprimé en EMOF).

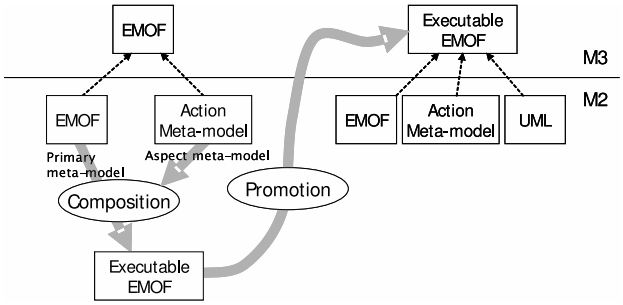

Figure 9 : Construction d'un EMOF exécutable. Les flèches pointillées symbolisent la relation de conformité d'un modèle à son metamodèle et les flèches grises et les ellipses représentent des transformations de modèles.

A partir de ces deux metamodèles, une première transformation (appelée *Composition*) a pour but de composer EMOF et le langage d'action en un metamodèle cohérent. Le résultat de cette transformation est le metamodèle de Kermeta en EMOF. La seconde transformation représentée sur la Figure 9 est la promotion. Cette transformation vise à "remonter" le langage Kermeta au niveau M3 afin de le poser comme base d'un nouvel environnement de modélisation. Ce processus est proche de la démarche qui est utilisée pour faire évoluer un langage et son compilateur d'une version n à une version n+1. On utilise la version n d'un langage pour implanter le compilateur de la version n+1 de ce même langage. Le compilateur de la version n du langage permet ainsi de compiler le compilateur de la version n+1. Une fois compilé, le compilateur de la version n+1 peut se recompiler seul et la version n du compilateur peut être abandonnée. Dans le cas présent EMOF joue le rôle de la version n et Kermeta celui de la version n+1.

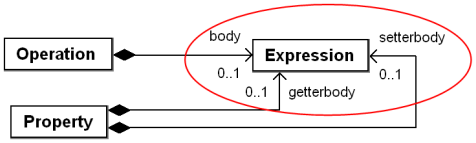

Figure 10 : spécification du corps des opérations par des expressions

La composition de EMOF avec le langage d'action peut paraître simple de primer abord mais elle nécessite en réalité un soin particulier pour assurer les qualités du langage obtenu par composition. Comme le montre la Figure 10, la jonction entre EMOF et le langage d'actions est faite par trois associations entre les classes *Properties* et *Operation* de EMOF et la classe *Expression* du langage d'action. Ainsi, on offre la possibilité de spécifier par des expressions

le comportement des opérations et le calcul des propriétés dérivées. Bien que structurellement ces trois associations suffisent à établir le lien entre EMOF et le langage d'action, un certain nombre de problèmes sémantiques restent à régler quant au système de types et au mécanisme de redéfinition du langage résultant.

La Figure 11 présente le metamodèle du langage Kermeta construit par extension de EMOF. Ce metamodèle est divisé en deux parties: une partie structurelle et une partie comportementale. La partie structurelle (23 classes), en haut sur la figure, est constituée des concepts issus de EMOF. La partie comportementale (26 classes) quant à elle correspond au langage d'action, c'est-à-dire aux sous-classes de la classe *Expression*.

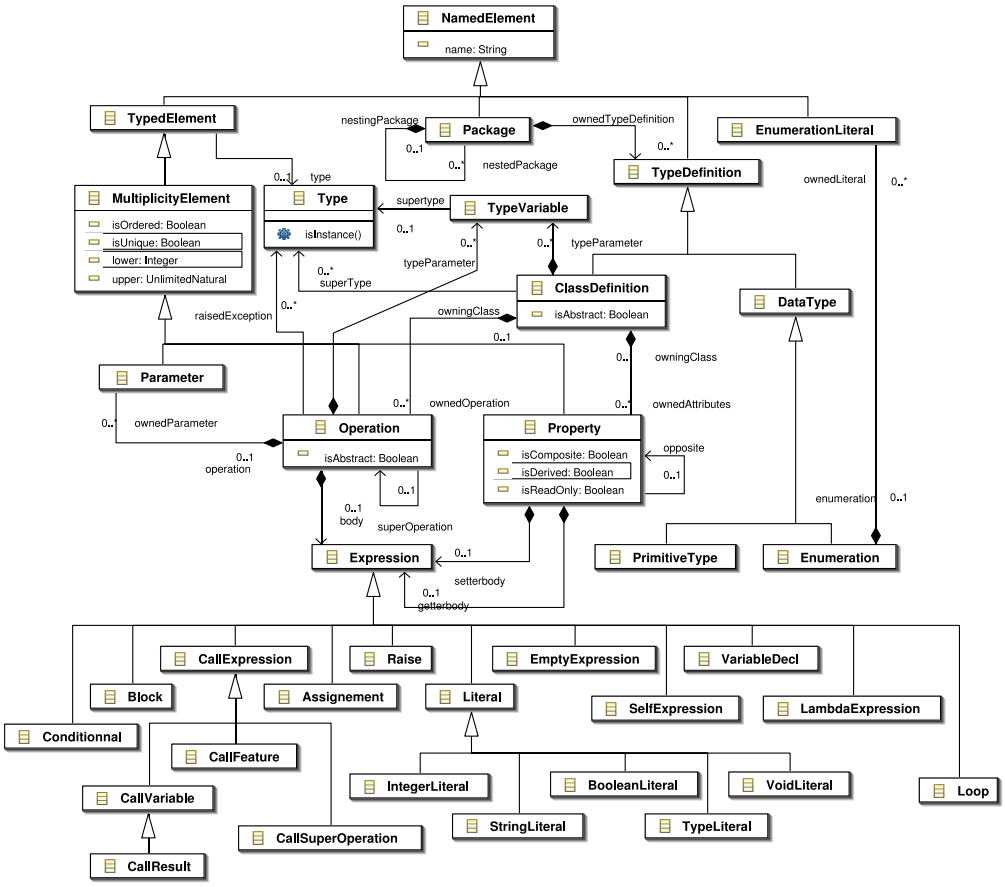

Figure 11 : principales classes du metamodèle de Kermeta

Une description complète du langage Kermeta, incluant la sémantique et des détails sur la machine virtuelle, est donnée par Franck Fleurey dans son mémoire de thèse intitulée

« Langage et méthode pour une ingénierie des modèles fiable ». Un résumé de la sémantique du langage figure en annexe de ce mémoire.

Nous avons débuté le développement de l'environnement Kermeta en janvier 2005 et terminé la réalisation du premier prototype 6 mois plus tard. Cette première version comportait un parseur, un vérificateur de types (*type-checker*), un interpréteur et un éditeur intégrés à Eclipse avec coloration et complétion. Le projet open-source Kermeta compte, en juin 2006, plus de douze développeurs actifs et un nombre d'utilisateurs croissants. En plus des éléments implantés dans la première version de l'environnement Kermeta, les principales nouveautés sont l'interopérabilité avec Java et EMF, un metteur au point (*debugger*), un éditeur graphique et des bibliothèques pour des metamodèles standards tel que UML2 par exemple.

A la lumière du retour de nos utilisateurs, le principal avantage de Kermeta par rapport aux solutions existantes est de permettre l'intégration forte de sémantique et de contraintes dans les metamodèles. Cela permet de bénéficier de l'ensemble des techniques de conception, de programmation, de test et de maintenance classiques du génie logiciel lors de la création de langages de modélisation.

3.3.3 Metamodèle de la syntaxe concrète

Nous avons défini plus haut un système de metamodélisation exécutable par sa capacité à décrire tous les aspects d'un langage. Kermeta, comme les langages de metadonnées dont il dérive, peut décrire la syntaxe abstraite sous la forme des classes d'un metamodèle. En plus, Kermeta permet de spécifier le domaine sémantique sous une forme opérationnelle au moyen des expressions contenues dans le corps des opérations. Il reste donc encore à fournir un moyen pour spécifier la syntaxe concrète, telle qu'elle sera manipulée par les utilisateurs. C'est l'objet de nos contributions sur la syntaxe concrète décrites dans cette section. La Figure 12, représente ces trois dimensions complémentaires de la définition d'un langage.

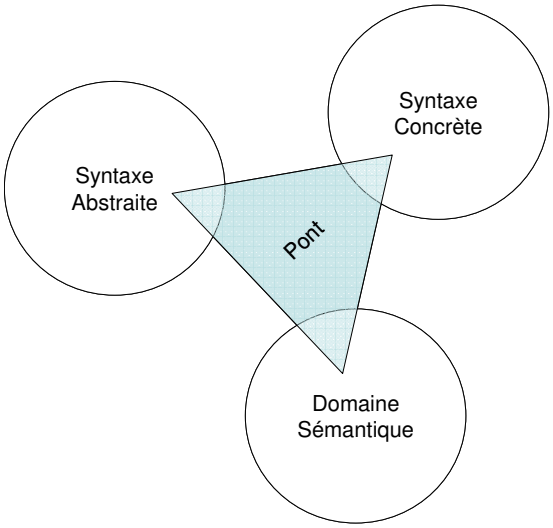

Figure 12 : Aspects de la modélisation d'un langage

En collaboration avec Frédéric Fondement (dont je co-encadre les travaux de thèse à l'EPFL), nous nous sommes intéressés à la synthèse et à l'analyse de syntaxe concrète (textuelle) dans le cadre d'un environnement à base de modèles. Ceci permet notamment de créer facilement des modèles conformes à un metamodèle donné, par exemple pour tester des transformations.

De premiers résultats sur la synthèse de syntaxe concrète à partir de modèles *(Model-driven generative approach for concrete syntax composition* [108]), ont été présentés en octobre 2004 au workshop OOPSLA & GPCE Best Practices for Model Driven Software Development. Nous avons montré notamment comment le metamodèle de navigation dans les pages Web issu de nos travaux sur la modélisation Web, pouvait être généralisé pour automatiser la génération de syntaxe concrète pour des langages décrits par des metamodèles.

En nous basant sur ces premiers travaux, nous avons expérimenté deux approches pour spécifier une syntaxe concrète lorsque la syntaxe abstraite a été définie au moyen d'un metamodèle.

La première approche consiste à instancier une syntaxe concrète générique, qui peut être rendue soit sous forme arborescente (par exemple sous la forme d'un explorateur tel l'éditeur réflexif d'EMF), soit alignée sur le metamodèle de la syntaxe abstraite (par exemple l'éditeur

de l'explorateur d'objet de l'AGL Netsilon), soit textuelle (par exemple avec HUTN [109]). L'avantage de cette approche est que la syntaxe concrète peut être entièrement dérivée de la syntaxe abstraite. Le désavantage est que les syntaxes concrètes ne sont pas personnalisables à l'infini, en fait le sucre syntaxique est prédéfini. Avec Michel Hassenforder du MIPS, nous discutons de la liaison entre les modèles et les grammaires, dans le contexte d'HUTN, dans le papier *HUTN as a Bridge between ModelWare and GrammarWare – An Experience Report*, présenté lors du workshop WISME 2005 [110].

La deuxième approche consiste à ajouter explicitement une information de présentation en syntaxe concrète pour chaque élément de la syntaxe abstraite (ou ensemble d'éléments s'il y a du sucre syntaxique). Nous avons choisi d'implanter cette deuxième approche sous la forme d'un metamodèle dédié à la représentation de la syntaxe concrète. Le principe est d'associer à chaque classe de la syntaxe abstraite (définie dans le metamodèle du langage) un modèle de la syntaxe concrète correspondante, et ainsi de pouvoir construire un modèle (conforme au metamodèle du langage) à partir d'un texte, et inversement de paraphraser un modèle vers sa représentation textuelle.

Dans le papier *Model Driven Analysis and Synthesis of Concrete Syntax [46]*, nous décrivons une spécification bidirectionnelle de la syntaxe concrète. Ce travail jette les bases d'un metamodèle de la syntaxe concrète d'un langage dont la syntaxe abstraite a préalablement été décrite par un metamodèle. La spécification est bidirectionnelle car elle permet à la fois d'aller des textes vers les modèles et inversement des modèles vers les textes. L'idée est d'expliquer au moyen d'un modèle comment une notion de la syntaxe abstraite est rendue dans une syntaxe concrète particulière. Une implémentation qui fonctionne par descente récursive pilotée par modèles est intégrée dans l'environnement de développement de Kermeta. Cet article figure en annexe de ce mémoire.

Notre proposition de metamodèle de la syntaxe concrète est présentée dans la Figure 13. Le metamodèle est constitué d'un ensemble de règles (sous-classes de la classe abstraite *Rule*) qui sont utilisées pour spécifier la syntaxe concrète. Le pont entre le metamodèle du langage et sa syntaxe concrète est basé sur les deux metaclasses *Class* et *Feature*, qui référencent respectivement les classes du metamodèle et leurs propriétés. La classe *Template* réalise la connexion entre une classe du metamodèle et la règle qui lui correspond. La classe *Value* (et

ses sous-classes) et la classe *Iteration* font la connexion entre les propriétés des classes et leurs valeurs. La classe *Value* est utilisée pour les propriétés dont la multiplicité vaut 1, alors que la classe *Iteration* gère les collections associées aux propriétés de multiplicité supérieure à 1. Les autres classes réalisent les constructions habituelles pour la spécification de syntaxe concrète, comme les terminaux, séquences et alternatives.

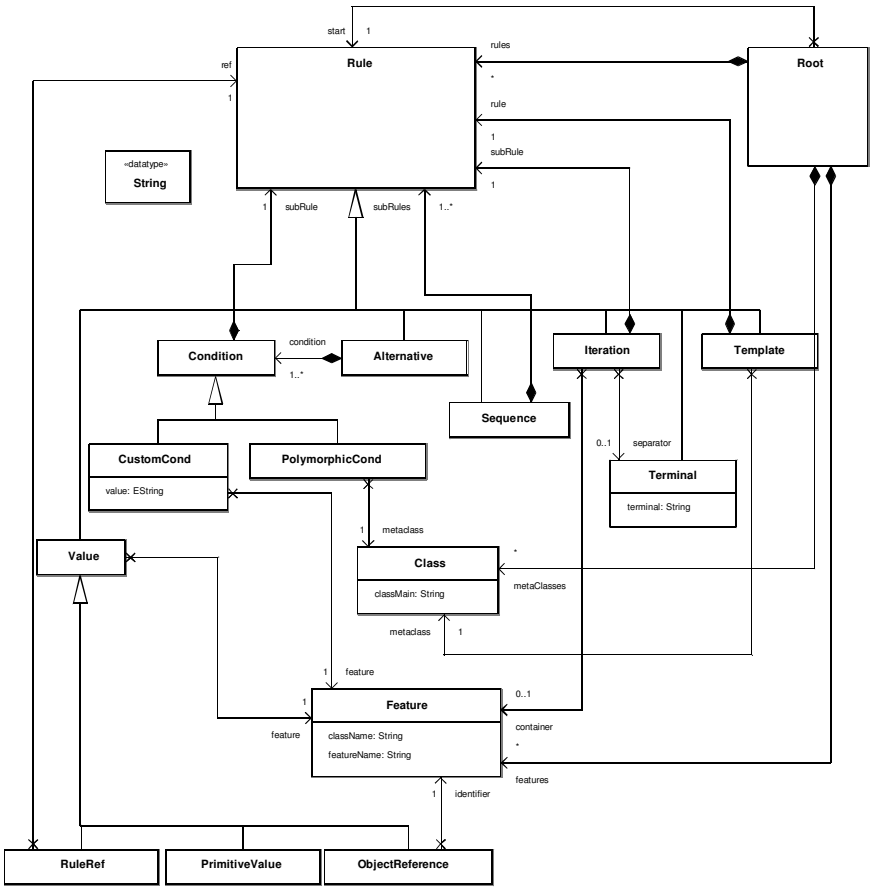

Figure 13 : survol du metamodèle de syntaxe concrète.

Nous avons défini deux transformations symétriques, la première pour analyser un texte et construire un modèle, et la seconde pour sérialiser un modèle vers un texte. Le but est de fabriquer une machine générique capable d'analyser et de synthétiser du texte, de sorte à pouvoir passer sans effort d'une représentation abstraite d'un modèle à une représentation concrète et inversement. La Figure 14 illustre notre démarche.

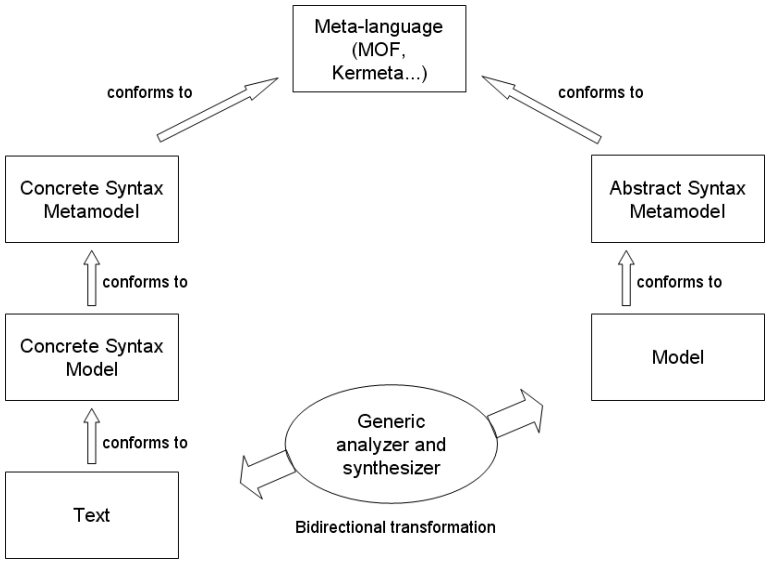

Figure 14 : transformation bidirectionnelle entre textes et modèles.

Lors de l'analyse de texte, le flux d'entrée est découpé en lexèmes et analysé par un analyseur générique qui fonctionne par descente récursive dirigée par un modèle. Il s'agit d'une démarche d'analyse descendante récursive qui se base sur l'information contenue dans le metamodèle du langage et dans le modèle de la syntaxe concrète. Lorsque l'analyseur rencontre une séquence de lexèmes valide du point de vue de la syntaxe concrète, il instancie la syntaxe abstraite et construit la représentation abstraite (en fait le modèle) qui correspond au texte en entrée.

Lors de la synthèse de texte, le texte est généré par un synthétiseur générique qui fonctionne comme un moteur de *template* dirigé par les modèles. Le synthétiseur visite le modèle (conforme au metamodèle de la syntaxe abstraite) puis utilise l'information de présentation décrite dans modèle de syntaxe concrète (conforme au metamodèle de syntaxe concrète) pour injecter du texte dans le flux de sortie.

Ces deux processus sont totalement symétriques et réversibles. Une forme de validation consiste à exécuter plusieurs séquences synthèse-analyse, puis à observer qu'il n'y a pas de différence entre les textes générés successivement.

Nous ne sommes qu'au début de ces travaux sur la syntaxe concrète. Nous travaillons actuellement sur une meilleure intégration entre les espaces technologiques des modèles et

des grammaires.

Un premier point dur est de trouver comment donner accès dans le monde des modèles (utilisés pour décrire des langages) aux capacités de validation des grammaires. La piste que nous suivons actuellement, consiste à identifier plus finement les notions présentes dans les deux univers, et à les fusionner dans un metamodèle unique. Nous envisageons de mettre en place des mécanismes de trace dans les transformations pour remontrer les diagnostics des outils du monde des grammaires dans des termes pertinents dans l'univers de modèles.

Un autre point dur technologique est la performance des analyseurs pilotés par des modèles. Notre piste actuelle passe par l'amélioration de la phase d'analyse lexicale dans le cas des interpréteurs pilotés par modèles, mais aussi par des projections vers les outils de *grammarware* ; un point qui découlera directement des travaux décrits dans le paragraphe précédent.

3.3.4 Exemples d'application de Kermeta et leçons apprises

Comme le montrent les applications décrites dans cette section, Kermeta a été appliqué avec succès à divers travaux dans le domaine de l'ingénierie dirigée par les modèles. En dehors des travaux de l'équipe Triskell, qui développe l'environnement Kermeta, un certain nombre de collaborateurs académiques et industriels utilise Kermeta pour leurs développements (citons AIRBUS, l'ENSTB, l'ENSIETA, FERIA, le LIRMM, France Télécom R&D, Colorado State University, l'Université de Carthagène et Trinity College à Dublin).

Les sections qui suivent présentent trois exemples d'utilisation du langage.

3.3.4.1 Transformation de modèles

La première étude que nous présentons ici est l'utilisation de Kermeta pour la transformation de modèles. Le langage Kermeta n'est pas dédié à la transformation de modèle car il n'intègre aucune construction spécifique à la transformation. Cependant, un des objectifs de Kermeta est de permettre de manipuler des modèles, c'est-à-dire de parcourir, créer et modifier des modèles. A ce titre, il doit donc permettre, et même faciliter, l'écriture de transformations de modèles. Afin de vérifier les aptitudes de Kermeta pour la transformation de modèles, nous nous sommes appuyés sur un ensemble de transformations de modèles proposées dans le

cadre du workshop MTIP (Model Transformation In Practice) [111]. L'objectif de ce workshop était d'évaluer et de comparer les différentes approches de transformation de modèles. Pour cela l'ensemble des participants devait implanter une étude de cas obligatoire : la génération de schémas de base de données à partir de diagrammes de classes. En plus de cette étude, quatre transformations optionnelles étaient proposées :

- la conversion de chiffres arabes en chiffres romains (et vice-versa),
- la déterminisation et minimisation d'automates,
- des refactorings sur des diagrammes de classes,
- la génération de code "non-conventionel".

En utilisant Kermeta, nous avons implanté l'étude de cas obligatoire ainsi que les trois premières études facultatives.

Manifestement, les approches déclaratives présentées lors de workshop permettaient une expression plus simple des transformations de type « mappings ». Par contre, Kermeta, du fait de son style impératif, s'est montré bien plus à l'aise pour les transformations qui comprenaient une composante algorithmique importante. Il est intéressant de noter que parmi les approches proposées au workshop, Kermeta est la seule qui ait traité les cas optionnels.

Une évolution facile a mettre en place pour améliorer le support des transformations de type mapping pourrait se faire en reprenant les travaux de D. Akehurst sur des patterns de transformations en Java [112], et en les appliquant à Kermeta.

3.3.4.2 Aspects pour la modélisation

La seconde étude de cas concerne l'implantation de techniques de *tissage* de modèles. La modélisation par aspects est un domaine de recherche actif de l'ingénierie des modèles. L'idée est de permettre d'exprimer dans des modèles séparés, appelés *modèles d'aspects*, les différents aspects d'un système complexe. Cette séparation permet une bonne gestion de la complexité, de découpler des préoccupations transverses et de réutiliser certains modèles d'aspects. La principale difficulté des techniques de modélisation par aspects est le *tissage* des modèles d'aspect qui permet de construire le modèle complet.

Cette section présente deux techniques de tissage de modèles. La première porte sur la composition des diagrammes de classes, elle a été réalisée en collaboration avec Robert France, Sudipto Ghosh et Raghu Reddy de Colorado State University. La seconde concerne le tissage des diagrammes de séquences et a été réalisée par Jacques Klein, dans le cadre de ses travaux de thèses sur les HMSC.

Pour ces deux expériences, la compatibilité de Kermeta avec d'autres outils de modélisation (en particulier Eclipse/EMF) est un élément important qui permet d'assurer la bonne intégration des opérateurs de tissage dans un environnement de modélisation complet. Les deux cas ont largement utilisé les possibilités offertes par Kermeta pour ajouter des opérations dans les classes des metamodèles, ce qui permis d'encapsuler les opérateurs de tissage directement dans les metamodèles considérés.

Dans le cas du tissage de diagrammes de classes, la composition est structurelle, c'est-à-dire que la composition des modèles revient à la composition des objets qu'ils contiennent. Cette propriété permet d'implanter l'opérateur de composition de façon générique en utilisant les mécanismes d'introspection (ou de réflexion) offerts par Kermeta.

Dans le cas du tissage de diagrammes de séquence, la détection et la composition repose sur la sémantique des diagrammes de séquences. Les opérateurs correspondants sont donc moins génériques mais s'intègrent naturellement sous forme d'opérations à ce metamodèle.

Le fait que Kermeta soit un langage à objets a permis de réutiliser le patron de conception *stratégie* afin d'implanter élégamment les différentes stratégies de détection et composition pour les diagrammes de séquences. Les résultats de cette étude montrent des qualités intéressantes de Kermeta pour l'implantation de techniques de modélisation par aspects. Chacune des deux techniques a pu être conçue et implantée de façon satisfaisante, en particulier de façon réutilisable (opérateur générique dans le cas des modèles de classes) et extensible (signature personnalisable dans le cas des modèles de classes et stratégies de compositions variables dans le cas des diagrammes de séquences).

3.3.4.3 Implantation d'un langage spécifique pour l'ingénierie des exigences

Cette dernière étude de cas présente une chaîne de traitement des exigences logicielles basée

sur l'utilisation de modèles. L'implantation a été réalisée fin 2005 par Waqas Ahmed Saeed au cours d'un stage de Master 2 Recherche dans l'équipe Triskell.

Cette chaîne de traitement des exigences a été conçue dans le cadre du projet CAROLL/Mutation [113] (faisant intervenir l'INRIA, le CEA et Thalès) et utilisée plus récemment par France Télécom. Ces travaux ont fait l'objet d'un certain nombre de publications parmi lesquelles [114] [115] [116]. Nous avons choisi cette application pour valider Kermeta vis-à-vis d'un processus d'ingénierie des modèles complet mettant en jeu la définition de plusieurs langages de modélisation, de leurs sémantiques et de transformations de modèles. Nous avons expérimenté deux approches pour la définition de la sémantique d'un langage dédié. La première solution est l'implantation d'une transformation de modèles permettant de cibler un langage dont la sémantique est connue (compilation). La seconde solution est l'implantation de la sémantique en Kermeta directement dans le metamodèle du langage dédié (interprétation).

Cette étude de cas a permis d'implanter en Kermeta un processus d'ingénierie dirigée par les modèles complet issue de collaborations avec l'industrie. L'intérêt de cette étude est qu'elle concentre une grande variété d'éléments spécifiques à l'ingénierie dirigée par les modèles : langages dédiés, transformations, simulations, etc. Pour implanter le processus complet deux langages dédiés ont été définis : le LDE et le langage de cas d'utilisation. La structure de ces langages a été décrite dans des metamodèles. La sémantique du LDE a été implantée par une transformation de modèles ciblant le langage de cas d'utilisation, ce qui a permis d'obtenir un simulateur de ce langage. Dans la suite de la chaîne, ce simulateur est utilisé pour produire des cas de tests.

Suite à cette expérimentation il a été décidé que les nouveaux développements, réalisés en collaboration avec France Télécom, autour de cette chaîne de traitement des exigences seront implantés en Kermeta. Les prototypes initiaux implantés en JAVA ont été abandonnés au profit de Kermeta qui fournit un meilleur support pour la manipulation de modèles.

3.3.4.4 Leçons apprises

Les applications décrites dans les sections précédentes montrent que le système de metamodélisation basé sur le langage Kermeta est opérationnel. Nous commençons à disposer

d'une communauté d'utilisateurs ; la deuxième journée Kermeta qui s'est tenue le 10 octobre 2006 à Rennes a rassemblé 70 participants qui nous ont fait par de leurs expériences.

La principale leçon que nous avons apprise avec le projet Kermeta, est que nous ne sommes qu'au tout début d'un environnement réellement orienté-modèle.

Certes, un langage comme Kermeta et son environnement support facilite la définition et la manipulation de metamodèles et de modèles, mais l'avantage par rapport à des solutions plus traditionnelles n'est pas décisif. Une alternative à Kermeta serait par exemple d'exprimer la partie structurelle des metamodèles avec un langage de metadonnées, puis de générer une projection de ces metadonnées dans un langage de programmation tel Java, pour ensuite écrire tout le comportement (par exemple les transformations) en Java, en se basant au besoin sur des bibliothèques (plutôt que sur des metamodèles ou des langages spécifiques).

Par rapport à cette approche mixte, Kermeta apporte la manipulation directe des associations, l'homogénéité des types (ce qui n'est pas le cas de Java qui propose des types différents de ceux des metamodèles), et surtout un contrôle de cohérence global sur les différents artefacts (le compilateur Java ne contrôle pas le contenu des chaînes de caractères qui assurent l'interface avec les référentiels de modèles, ou les moteurs de transformations par exemple).

A notre avis, il manque encore à Kermeta un opérateur de fusion de metamodèles, le typage de modèles et une meilleure encapsulation des systèmes de fichiers pour revendiquer pleinement le statut de langage orienté-modèle.

Dans un futur proche, il nous faut commencer par durcir les outils de l'environnement Kermeta, pour réussir à prendre en compte des metamodèles (et surtout des modèles) de plus grande ampleur. Les plus grands metamodèles que nous ayons traités actuellement sont ceux d'UML2 et de Java5. L'expérience montre que nous atteignons avec ces metamodèles les limites de l'implémentation de notre machine virtuelle, tant en vitesse d'exécution qu'en empreinte mémoire. La solution passe par une technique de compilation pour remplacer l'implantation actuelle par interprétation.

L'intégration très forte avec Eclipse est indiscutablement un facteur clé pour l'acceptation de Kermeta par la communauté. Le revers de la médaille, est que cette intégration exige un effort

d'ingénierie considérable. Eclipse, et particulièrement EMF, sont complexes, en évolution rapide (pas de stabilité, peu documentés et aucune visibilité sur les évolutions futures), et nous obligent à faire des choix de conception parfois différents de ceux que nous aurions pu faire dans un environnement « idéal ». Les mécanismes pour traiter les fichiers ne sont pas complètement transparents du fait des technologies sous-jacentes, nous rencontrons beaucoup de problèmes techniques avec le référentiel de modèles d'EMF. Nous devons progresser également sur la documentation et la gestion des versions.

4 Perspectives de recherche

Mon projet scientifique pour les prochaines années s'inscrit dans la lignée de mes travaux précédents sur l'opérationnalisation des modèles et sur l'application des modèles à l'ingénierie des langages.

Le vaste champ d'investigation de l'IDM dépasse de loin ce qu'une personne, ou une équipe peut entreprendre. Nous sommes actuellement dans une phase de défrichage, encore largement empirique. Nous manipulons des notions sans les avoir défini formellement, ni sans avoir défini des opérateurs qui s'appliquent sur ces notions. Pour avancer significativement, l'IDM doit se doter d'une théorie des modèles afin de pouvoir d'une part raisonner sur les modèles et d'autre part afin de pouvoir construire des outils efficaces.

L'objectif de mon projet de recherche est de contribuer à cet effort d'élaboration d'un corpus de connaissances. Il me semble évident que pour relever ce défi il convient de regrouper nos efforts, et de trouver les moyens de collaborer et d'échanger. La structuration de la communauté de recherche impliquée dans l'ingénierie dirigée par les modèles repose sur une identification des thématiques, des vecteurs de publications et aussi sur un cadre technologique fondamental capable d'agir comme un catalyseur pour de nombreux projets. Je compte poursuivre dans cette voie du développement de technologie fondamentale car je trouve que c'est un champ d'expérimentation particulièrement fertile du fait de ses applications larges.

4.1 Verrous scientifiques

Pour pouvoir gérer la complexité d'un système réel, l'ingénierie dirigée par les modèles propose de raisonner sur des modèles pour éviter la complexité de la réalité. Un modèle est ici perçu comme une abstraction de la réalité, où la réalité est restreinte en fonction d'un point de vue particulier. Dans un système logiciel par exemple, les aspects liés à la performance ou à la qualité pourront faire l'objet de modèles spécifiques.

Dans ce cadre, la définition est un point critique de l'ingénierie des modèles. La définition des modèles nécessite de spécifier clairement les objectifs de la modélisation. Quelles sont les entités de la réalité à représenter ? Avec quel niveau de détail faut-il décrire la réalité ? Cependant, pour être exploitable, un modèle doit assurer un certain niveau de qualité :

cohérence des données, complétude des données, sémantique, etc. La validation des modèles est donc un autre point critique de l'ingénierie des modèles.

Pour un même point de vue ou un même objectif, les modèles peuvent prendre plusieurs formes : graphiques, descriptions formelles, descriptions textuelles, etc. Manipuler les modèles requiert alors de définir les concepts principaux associés à la problématique sous-jacente. Les metamodèles vont définir ces concepts, qui permettent de raisonner sur les modèles et de construire un certain nombre d'outils adéquats.

La définition des metamodèles est ainsi au coeur de l'ingénierie des modèles et engendre un grand nombre de langages permettant de décrire les différents aspects de la réalité : modèle logique, modèle conceptuel, modèle physique… Cette multitude de langages est imposée par la diversité des points de vue, et la gestion de cette multiplication des langages constitue un des grands défis de l'ingénierie dirigée par les modèles. L'ingénierie des modèles doit notamment fournir l'outillage nécessaire pour :

- comparer les modèles et les metamodèles entre eux et prouver leurs équivalences ;
- transformer les modèles pour passer d'un langage à un autre ;
- fusionner les modèles et les metamodèles pour intégrer les différents aspects d'une même réalité.

La promesse de l'ingénierie dirigée par les modèles est de capitaliser sur les modèles et les metamodèles. L'emploi effectif de l'ingénierie dirigée par les modèles ne sera toutefois envisageable que lorsque certaines questions auront été traitées :

- quel est le coût de développement des metamodèles (et des modèles) ;
- quel est l'impact de l'IDM sur le cycle de développement en terme de productivité, de réactivité, de maintenance, etc. ;
- comment intégrer facilement l'existant dans l'IDM ?

4.2 Travaux à court terme - La plateforme RNTL OpenEmbeDD

Je coordonne le projet de plateforme RNTL OpenEmbeDD (http://openembedd.inria.fr) qui a pour objectif de mettre l'ingénierie dirigée par les modèles au service des applications temps réels embarquées. L'effort total est de 556 h/m, pour un budget total de 7 millions d'euros et une aide demandée de 2,6 millions.

Le projet se fonde sur le double constat que l'industrie du logiciel embarqué doit d'une part faire face à de multiples défis économiques, techniques et stratégique, alors que d'autre part les méthodes spécifiques au temps-réel sont souvent trop faiblement outillées, difficilement adaptables et n'ont pas été conçues pour être intégrées à une chaîne méthodologique complète.

Dans le domaine du temps réel embarqué (TR/E) des points de rupture existent dans les flots actuels de conception et sont autant des verrous technologiques à l'emploi généralisé de techniques formelles éprouvées, par ailleurs développées à l'écart des standards de représentation ce qui constitue un verrou méthodologique faisant obstacle à leur usage. Un exemple typique de point de rupture est le passage d'une modélisation non temporisée à une modélisation temporisée.

L'ingénierie dirigée par les modèles (IDM), associée à des technologies formelles spécifiques au TR/E, apparaît aujourd'hui comme une piste de solution prometteuse pour dépasser ces obstacles technologiques et méthodologiques. Le projet OpenEmbeDD est le lieu où pourront être levés ces verrous méthodologiques et technologiques. En particulier :

- Favoriser l'émergence d'un standard UML pour le temps réel.
- Assurer l'interopérabilité avec les technologies les plus en pointe comme celles issues du paradigme synchrone ou, en général, les techniques de validation formelle.
- Prendre en compte en amont les caractéristiques temporelles et spatiales des applications, et minimiser les ruptures du flot de conception par la systématisation du recours à des techniques de transformation basées sur les modèles (raffinement, abstraction, synthèse…).

- Maîtriser et capitaliser la chaîne méthodologique.

- Simuler les modèles du temps réel pour formaliser et outiller la partie exécutable des modèles à simuler.

- Aider au développement d'outillage adapté au temps réel et à l'embarqué. selon deux approches. La première, plus traditionnelle, consistant à développer chacun de ces outils de telle manière qu'ils interopèrent via la plateforme. La seconde consistant à utiliser les techniques de génération d'outils proposée par l'IDM. Le choix entre ces deux approches étant guidé suivant la complexité de l'outil à obtenir et la maturité des techniques de l'IDM.

D'un point de vue plus personnel, le projet OpenEmbeDD me donne le cadre idéal pour valider le passage à l'échelle de mes contributions sur l'application de l'IDM à l'ingénierie des langages, en particulier pour le pont entre l'IDM et les grammaires. Nous avons vu plus haut que nous atteignons les limites de l'implantation actuelle de la machine virtuelle Kermeta (avec les metamodèles d'UML2 et de Java5), OpenEmbeDD nous donne aussi les moyens de durcir cette implantation, tout en fournissant des cas d'utilisation industriels, en vraie grandeur.

4.3 Travaux à moyen terme

Cette section survole quelques pistes que j'aimerai poursuivre pour défricher le domaine de l'IDM.

4.3.1 Projection vers des modèles formels et remontée de diagnostics

Les outils d'analyse de propriétés formelles de modèles n'acceptent en général que des formalismes et des notations spécifiques. Beaucoup de projets ont été mis en oeuvre pour appliquer ces techniques sur des applications industrielles et des cas réels. Mais nous constatons aujourd'hui encore leur faible pénétration dans le processus d'ingénierie système et logicielle comparativement aux énormes besoins en terme de recherche de fiabilité et sûreté de fonctionnement des systèmes critiques.

Ce paradoxe qui constitue un verrou méthodologique trouve en partie ses causes dans la

difficulté réelle de manipuler des concepts théoriques et des méthodes formelles dans un cadre industriel et aussi dans les nombreux problèmes non résolus quant aux traitements de systèmes complexes et soumis à des contraintes fortes (temps réel, criticité, déploiement).

Pour lever une partie de ce verrou méthodologique, il me semble pertinent de définir des transformations de modèles et des systèmes de trace, de sorte que les diagnostics retournés par les outils d'analyse formelle puissent être présentés au concepteur d'un système dans des vues spécifiques, compréhensibles par lui, indépendamment des techniques et langages formels mis en œuvre. Cette démarche est proche de celle que nous expérimentons actuellement dans nos travaux sur l'intégration avec les outils du *grammarware*, nous devrions pouvoir réutiliser une partie de ces résultats ici.

4.3.2 Test de transformations de modèles

Les transformations de modèles tiennent une place importante dans l'ingénierie dirigée par les modèles. Elles permettent non seulement l'automatisation de certaines activités de développement mais aussi la capitalisation et la réutilisation de savoir faire. A ce titre, afin d'assurer la qualité du processus de développement, il est nécessaire d'assurer la qualité des transformations de modèles. Dans la communauté de l'IDM, un grand nombre de travaux s'intéressent aux langages et techniques pour le développement de transformations de modèles mais, parmi ces travaux, très peu s'intéressent aux problèmes de la validation.

Le point dur est de trouver comment tester les transformations de modèles. Les transformations de modèles peuvent (et doivent) être validées comme n'importe quel autre programme. Cependant, les transformations de modèles ont la particularité de manipuler des modèles qui sont eux-mêmes décrits par des metamodèles.

Cette particularité peut être exploitée pour développer des critères de test, des générateurs de données de test, des oracles et des techniques de diagnostic spécifiques aux programmes de transformation de modèles. Ces techniques spécifiques permettant alors de compléter les outils de validation disponibles pour les programmes.

4.3.3 Modélisation par aspects

La conception par aspects est un paradigme de conception transverse qui permet de réduire fortement les couplages entre les différentes préoccupations de conception. Initialement issue du monde de la programmation, la technique des aspects peut aussi s'appliquer aux modèles, le seul pre-requis étant l'existence d'un tisseur d'aspects pour le domaine cible qui doit permettre de :

- faire la différence entre le modèle de base et un aspect ;
- spécifier les correspondances entre les entités à composer ;
- disposer de directives pour intégrer des préoccupations transverses ; ces directives permettront à la fois de résoudre les conflits et de gérer l'ordre de composition si plusieurs aspects doivent être intégrés dans le modèle.

La composition des aspects reste cependant problématique lorsqu'elle est centrée sur une approche "programmative" qui rend difficile le développement par réutilisation de modèles déjà tissés ou développés sans connaissance des autres aspects.

Le point dur est d'exprimer des propriétés de composition intuitive, démontrables et probablement non dépendantes d'une relation d'ordre des déclarations. La détection des conflits de composition est alors essentielle.

Une piste pour aborder cette problématique est de considérer la composition de modèles comme une forme spécifique de transformation de modèles et ainsi de bénéficier dans l'univers des aspects des avancées déjà obtenues pour la composition des transformations.

4.3.4 IDM, automatique et génie informatique

L'automatique est la discipline qui étudie les systèmes dynamiques, les signaux et l'information, à des fins de conduite ou de prise de décision. Le génie informatique étudie les systèmes informatiques et informatisés avec lesquels l'homme coopère, destinés à la perception, l'observation, l'aide à la décision et à la conduite de procédés et de phénomènes dynamiques. Le génie informatique, dans ces diverses approches partage avec l'informatique et l'électronique des méthodes de modélisation et de résolution.

Il subsiste néanmoins une différence fondamentale dans les approches de modélisation qui constitue un point dur méthodologique. Les systèmes matériels sont généralement conçus comme une interconnexion de composants modélisés par des modèles analytiques exprimés sous la forme d'équations qui spécifient leurs fonctions de transfert. Les systèmes informatiques sont quant à eux modélisés par des modèles opératoires exprimés par rapport à des machines virtuelles. Les approches de constructions de ces systèmes sont fondamentalement différentes. Pour les premiers, l'opération de base est la composition de fonctions de transfert ; pour les seconds, l'opération de base est le produit d'automates.

Une piste pour rapprocher ces deux points de vue me semble passer par la metamodélisation et le tissage d'aspects. Pour simplifier, j'ai l'intention d'expérimenter une intégration des deux points de vue dans un environnement technologique volontairement restreint (une plateforme temps-réel synchrone).

4.3.5 Modélisation de l'intention

La modélisation ne s'effectue pas dans le vide. Un modèle est réalisé dans un but donné, sous des contraintes données, et est donc le reflet de nombreux compromis. Ces compromis, choix, décisions, objectifs, ne sont pas exprimés explicitement, et ne sont donc pas accessibles aux utilisateurs des modèles.

La place laissée à ces informations implicites est probablement plus large dans le cas d'un langage de modélisation généraliste que dans celui d'un langage spécifique. Un metamodèle agit comme un filtre et permet dans le cas d'un langage spécifique de préciser plus clairement les notions concernées.

Le point dur est de mesurer ou évaluer la place prise par ces considérations implicites, puis de distinguer ce qui dans un metamodèle relève de la description d'un domaine ou d'un regard particulier porté sur ce domaine. L'une des premières difficultés pour atteindre cet objectif est la construction des modèles : quelle est la finalité des modèles visés ? Comment appréhender le système observé ? Comment abstraire cette réalité et faire ressortir les artefacts essentiels ? Quelle granularité de modélisation en fonction de quel contexte d'utilisation?

La possibilité de définir précisément la sémantique opérationnelle des langages utilisés pour

construire les modèles constitue un premier pas pour réduire la part d'implicite dans les modèles. Le but de cette piste de recherche est d'investiguer comment les modèles (ou leur environnement d'élaboration) pourrait être rendu plus explicites, puis d'évaluer le résultat en terme de réutilisabilité des modèles. Par contre, l'intégration de techniques provenant de l'ingénierie des besoins, de l'acquisition des connaissances et des sciences cognitives, entre autre, n'est pas une tâche aisée ; elle nécessite une approche pluridisciplinaire qui n'est pas sans risque.

4.3.6 Cartographie de l'ingénierie des logiciels

L'objectif est de modéliser le processus d'informatisation de manière générale. On peut aussi parler de modéliser le génie logiciel, tant au travers de ses pratiques que des artefacts qu'il manipule. Cette thématique comprend les travaux sur les relations entre modèles, les megamodèles et les zoos de modèles.

Le point dur est de trouver le formalisme qui nous permettra de raisonner sur les artefacts du génie logiciel.

Il me semblerait particulièrement intéressant d'essayer de bénéficier d'un outil de modélisation des théories, telle la théorie des catégories, et de voir comment les différents types de relations que nous découvrons entre les artefacts de modélisation peuvent se formaliser (puis s'analyser) au moyen de morphismes, de foncteurs ou d'autres notions de catégorisation.

5 Conclusion

Dans ce document de synthèse, j'ai mis en lumière le travail scientifique que nous avons mené au cours des années qui se sont déroulées après ma thèse. J'ai notamment montré comment avec l'aide de doctorants, d'étudiants de DEA et d'ingénieurs, j'ai construit un parcours original qui m'a emmené de la modélisation objet des logiciels à la metamodélisation des langages informatiques.

Les principales contributions décrites dans ce mémoire portent sur :

- Des avancées méthodologiques, dans le domaine de la modélisation objet avec UML et aussi dans l'intégration entre l'ingénierie dirigée par les modèles et les méthodes agiles. Ces contributions ont été validées dans des contextes industriels, en collaboration avec Rational Software et ObjeXion Software.

- Des contributions pour la capitalisation du savoir-faire de modélisation des systèmes de commande et contrôle. Ces travaux à la frontière entre l'informatique et l'automatique, ont été réalisés en partenariat avec l'Ifremer et Nipson Printing System.

- Des contributions pour la modélisation opérationnelle des systèmes d'information Web. Ces contributions s'inscrivent dans le cadre d'un transfert de technologie, sous la forme d'une création d'entreprise innovante : ObjeXion Software.

- Des contributions pour l'application de la modélisation à l'ingénierie des langages. Ces travaux regroupés dans le projet open-source Kermeta porté par l'INRIA, font l'objet d'un déploiement industriel dans le cœur de la plateforme RNTL OpenEmbeDD.

Dans ce mémoire j'ai surtout insisté sur nos travaux récents, en particulier sur la définition du système de metamodélisation construit autour du langage Kermeta. Kermeta est un langage de metaprogrammation exécutable construit comme une extension d'EMOF par tissage d'aspect. De même que Niklaus Wirth définissait les programmes comme des structures de données et des algorithmes, Kermeta considère les metamodèles comme des metadonnées et des actions. Cette approche permet de modéliser les langages informatiques selon trois aspects : la syntaxe

abstraite au moyen de classes et de relations, la sémantique opérationnelle au moyen d'expressions écrites en langage d'action, et enfin la syntaxe concrète au moyen d'un metamodèle spécifique.

Le mémoire se termine par les perspectives de recherche dans lesquelles j'aimerai inscrire mes travaux dans les prochaines années.

Bibliographie

[1] P. J. Denning et A. McGettrick. "Recentering Computer Science", *Communications of the ACM*, vol. 48 (11), pp. 15-19, 2005.

[2] P.-A. Muller, *Modélisation objet avec UML*, 420 p., Eyrolles, Paris, 1997.

[3] Y. Amghar, A. Flory, et J.-M. Pinon. "La technologie à objets: domaines et utilisations", *Ingénierie des Systèmes d'Information*, vol. 3 (6), pp. 739-776, 1995.

[4] J.-M. Favre, J. Estublier, et M. Blay-Fornarino, *L'ingénierie dirigée par les modèles. Au-delà du MDA*, 226 p., Hermes, Paris, 2006.

[5] A. van Deursen, P. Klint, et J. Visser. "Domain-Specific Languages: An Annotated Bibliography", *ACM SIG-PLAN Notices*, vol. 35 (6), pp. 26-36, 2000.

[6] C. Consel et R. Marlet. "Architecturing software using a methodology for language development". In *Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming*, pp. 170-194, Pisa, Italy, Lecture Notes in Computer Science, 1490, 8 septembre 1998.

[7] J. Sztipanovits, G. Karsai, C. Biegl, T. Bapty, K. Ldeczi, et A. Misra. "MULTIGRAPH: an architecture for model-integrated computing". In *Proceedings of the First IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95)*, pp. 361-368, Southern Florida, USA, November 6-10, 1995.

[8] J. Sztipanovits et G. Karsai. "Model-Integrated Computing", *Computer*, vol. 30 (4), pp. 110-111, 1997.

[9] J. Greenfield, K. Short, S. Cook, et S. Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, 666 p., Wiley, Indianapolis, 2004.

[10] R. Soley, MDA Model-Driven Architecture - White paper, *Electronic Source: Object Management Group*, November, 27 2000, ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf

[11] K. Gabor, S. Janos, L. kos, et B. Ted. "Model-integrated development of embedded software", *Proceedings of the IEEE*, vol. 91 (1), pp. 145-164, 2003.

[12] L. Akos, B. Arpad, M. Miklos, V. Peter, N. Greg, S. Jonathan, et K. Gabor. "Composing Domain-Specific Design Environments", *Computer*, vol. 34 (11), pp. 44-51, 2001.

[13] D. James. "GME: the generic modeling environment". In *Proceedings of the 18th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages,*

and Applications (OOPSLA '03), pp. 82-83, Anaheim, CA, USA, October 2003.

[14] OMG, Model Driven Architecture, *Electronic Source: Object Management Group*, http://www.omg.org/mda/

[15] OMG, UML 2.0 Object Constraint Language (OCL) Final Adopted Specification, *Electronic Source:* http://www.omg.org/cgi-bin/doc?ptc/2003-10-14

[16] W3C, XSL Transformations (XSLT) Version 2.0, *Electronic Source: World Wide Web Consortium*, 8 june 2006, http://www.w3.org/TR/xslt20/

[17] W3C, XQuery 1.0: An XML Query Language Version 2.0, *Electronic Source: World Wide Web Consortium*, 8 june 2006, http://www.w3.org/TR/xquery/

[18] K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszky, U. Prange, G. Taentzer, D. Varro, et S. Varr 'o-Gyapay. "Model Transformation by Graph Transformation: A Comparative Study". Presented at *MTiP 2005, International Workshop on Model Transformations in Practice (Satellite Event of MoDELS 2005)*, Montego Bay, Jamaica, October 2005.

[19] InteractiveSoftware, The leading platform for model-driven architecture - White paper, *Electronic Source: Interactive Software GMBH*, http://www.interactive-objects.com/products/arcstyler/overview/

[20] ObjecteeringSoftware, Objecteering 6: the UML MDA tool for model driven development, *Electronic Source:* http://www.objecteering.com

[21] Compuware, OptimalJ - Model-driven development for Java, *Electronic Source: Compuware*, http://www.compuware.com/products/optimalj/

[22] S. Burmester, H. Giese, J. Niere, M. Tichy, J. Wadsack, P., R. Wagner, L. Wendehals, et A. Zundorf. "Tool Integration at the Meta-Model Level within the FUJABA Tool Suite", *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 6 (3), pp. 203-218, 2004.

[23] MiaSoftware, Mia -Transformation, *Electronic Source:* http://www.mia-software.com/

[24] PathfinderSolutions, PathMATE: Transformation Engine, *Electronic Source:* http://www.pathfindermda.com/

[25] F. Jouault et I. Kurtev. "Transforming Models with ATL". Presented at *MTiP 2005, International Workshop on Model Transformations in Practice (Satellite Event of MoDELS 2005)*, Montego Bay, Jamaica, October 2005.

[26] Didier Vojtisek et J.-M. Jézéquel. "MTL and Umlaut NG - Engine and Framework for

Model Transformation", *ERCIM News 58*, vol. 58 8 juillet 2004, 2004.

[27] andromda.org, AndroMDA, *Electronic Source:* http://www.andromda.org/

[28] P. Braun. "Metamodel-based Integration of Tools". Presented at *Tool Integration in System Development (Workshop of the 4th joint meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering)*, Helsinki, September 2003.

[29] M. Alanen et I. Porres. "Coral: A Metamodel Kernel for Transformation Engines". Presented at *Second European Workshop on Model Driven Architecture (MDA)*, Canterbury, United Kingdom, September 7th-8th 2004.

[30] C. Dumoulin, ModTransf, *Electronic Source:* http://www.lifl.fr/west/modtransf/

[31] L. Tratt, QVTEclipse, *Electronic Source:* http://qvtp.org/downloads/qvtp-eclipse/

[32] UMT-QVT, UMT, *Electronic Source:* http://umt-qvt.sourceforge.net/

[33] K. Smolander, K. Lyytinen, V.-P. Tahvanainen, et P. Marttiin. "MetaEdit: a flexible graphical environment for methodology modelling". In *Proceedings of the CAiSE '91: third international Conference on Advanced Information Systems Engineering*, pp. 168-193, New York, NY, USA, 1991.

[34] J.-P. Tolvanen et M. Rossi. "MetaEdit+: defining and using domain-specific modeling languages and code generators". In *Proceedings of the OOPSLA '03: 18th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 92-93, Anaheim, California, October 26-30 2003.

[35] Xactium, XMF-Mosaic, *Electronic Source:* http://www.xactium.com

[36] P.-A. Muller. "Comment identifier les différents types de relations dans les modèles orientés-objets, application à la méthode de Booch". In *Proceedings of the Journées de synthèse technologies objets, AFCET*, pp. 89-100, Paris, Janvier 1996.

[37] M.-C. Roch, P.-A. Muller, et B. Thirion. "Improved flexibility of a document production line through object-oriented remodeling". In *Proceedings of the 2nd IMACS-IEEE International Multiconference, CESA'98: computational engineering in systems applications*, pp. 152-159, Nabeul Hammamet, 1-4 April 1998.

[38] M.-C. Roch, P.-A. Muller, G. Metzger, et B. Thirion. "Modélisation d'une application de composition automatique de documents avec UML", *Génie Logiciel*, vol. (46), pp. 134-138, 1997.

[39] N. Gaertner, P.-A. Muller, et B. Thirion. "Idiomes, Patterns, Frameworks, un tour

d'horizon". Presented at *Journée COOSI AFCET*, Paris, Mai 1997.

[40] P.-A. Muller et N. Gaertner, *Modélisation objet avec UML*, 520 p., Eyrolles, Paris, 2000.

[41] *Software Product Lines; Experiences and Research Directions*, The International Series in Engineering and Computer Science, vol. 576, 552 p., Kluwer, Dordrecht, Netherlands, 2000.

[42] K. Czarnecki et U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, 864 p., Addison Wesley, New York, NY, USA, 2000.

[43] P.-A. Muller, P. Studer, F. Fondement, et J. Bézivin. "Platform Independent Web Application Modeling and Development with Netsilon", *Software and Systems Modeling*, vol. Vol 4 (4), pp. 424-442, 2005.

[44] W. El Kaim, P. Studer, et P.-A. Muller. "Model Driven Architecture for Agile Web Information System Engineering." In *Proceedings of the OOIS 2003, 9th International Conference on Object-Oriented Information Systems*, pp. 299-303, Geneva, 2817.

[45] P.-A. Muller, F. Fleurey, et J.-M. Jézéquel. "Weaving executability into object-oriented meta-languages". In *Proceedings of the MoDELS/UML 2005 - ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, pp. 264-278, Montego Bay, Jamaica, Lecture Notes in Computer Science, 3713, October 2005.

[46] P.-A. Muller, F. Fleurey, F. Fondement, M. Hassenforder, R. Schneckenburger, S. Gérard, et J.-M. Jézéquel. "Model-Driven Analysis and Synthesis of Concrete Syntax". In *Proceedings of the MoDELS/UML 2006 - ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*, pp., Genova, Italy, Lecture Notes in Computer Science, 4199, October 2003.

[47] P.-A. Muller et B. Thirion. "Integrating OSF/MOTIF User Interfaces with Ada Applications". In *Proceedings of the 5th International Conference on Software Engineering and its Applications*, pp. 659-668, Toulouse, december 1992.

[48] P.-A. Muller. "RMI, a « motified » version of RXI", *Rational Technical Representatives Newsletter*, vol. 24, 1991.

[49] P.-A. Muller. "Integration, CASE tools and software engineering", *Rational Technical Representatives Newsletter*, vol. 17, 1991.

[50] M. J. Stefik, D. G. Bobrow, et K. M. Kahn. "Integrating access-oriented programming

into a multiparadigm environment", *IEEE Software*, vol. 3 (1), pp. 11-18, 1986.

[51] T. A. Henzinger et J. Sifakis. "The Embedded Systems Design Challenge". Presented at *FM'06: 14th Symposium on Formal Methods*, Hamilton, Canada, August 21 - 27, 2006.

[52] P.-A. Muller, M. Laï, et A. Peuch. "Une expérience de développement orienté-objet d'un logiciel de contrôle-commande de robot sous-marin: le projet VORTEX". In *Proceedings of the 6th International Conference on Software Engineering and its Applications*, pp., Paris, novembre 1993.

[53] D. Roberts et R. Johnson. "Evolve frameworks into domain-specific languages". In *Proceedings of the PLOP'96: Third International Conference on Pattern Languages of Program Design*, pp., Allerton Park, Illinois, Sept. 4-6, 1996.

[54] P.-A. Muller. "Rapport d'expérience sur l'enseignement de l'approche orientée-objets". In *Proceedings of the congrès biennal de l'AFCET*, pp. 139-148, Toulouse, Novembre 1995.

[55] P.-A. Muller. "Un exemple d'enseignement global de l'approche orientée-objets". In *Proceedings of the Object-Expo*, pp. 179-184, Paris, Décembre 1995.

[56] P. Kruchten. "Architectural Blueprints - The "4+1" View Model of Software Architecture", *IEEE Software*, vol. 12 (6), pp. 42-50, 1995.

[57] P.-A. Muller. "Architecture objet". Presented at *Séminaire EEA*, Mulhouse, Mars 1995.

[58] P.-A. Muller. "Expression des vues d'architecture avec UML". Presented at *Journées COOSI AFCET*, Paris, mars 1997.

[59] P.-A. Muller. "Architecture avec UML". Presented at *10th International Conference on Software Engineering and its Applications*, Paris, 5 décembre 1997.

[60] P.-A. Muller. "Représentation des vues d'architecture avec UML", *Génie Logiciel*, vol. (46), pp. 128-133, 1997.

[61] P.-A. Muller. "Les derniers développements d'UML 1.0". Presented at *SSD'97: Solutions for Software Development*, Paris, Décembre 1997.

[62] P.-A. Muller. "Démarche de modélisation d'un système d'information avec UML". In *Proceedings of the 16th Congrès INFORSID: Informatique des Organisations et Systèmes d'Information et de Décision*, pp. 11, Montpellier, 12-15 mai 1998.

[63] P.-A. Muller. "Mise en œuvre d'UML, une méthode de développement universelle".

Presented at *SSD'98: Solutions for Software Development*, Paris, Décembre 1998.

[64] P.-A. Muller. "Solution cherche problème". Presented at *Journées UML 2000*, Rennes, 10 novembre 2000.

[65] P.-A. Muller. "Introduction à UML", *Point DBF*, vol. 96, pp. 15-17, 1999.

[66] P.-A. Muller. "Mise en œuvre d'UML, vers une démarche de développement universelle", *Point DBF*, vol. 96, pp. 20-22, 1999.

[67] M. Bouzeghoub, G. Gardarin, et P. Valduriez, *Les objets*, 390 p., Eyrolles, 1997.

[68] P. Desfray. "UML profiles versus metamodeling extensions, an ongoing debate." Presented at *UML In The.Com Enterprise: Modeling CORBA, Components, XML/XMI And Metadata Workshop*, Palm Springs, 2000.

[69] G. Martin, *L'imprimerie*, Que sais-je? 126 p., PUF, Paris, 1993.

[70] J.-C. Léac, *La chaîne graphique*, 134 p., Eyrolles, Paris, 2006.

[71] Dupoirier, *Technologie de la GED: L'édition électronique*, 304 p., Hermes, 1994.

[72] APROGED, GED - Glossaire des termes, *Electronic Source: Association des professionnels de la GED*, http://www.aproged.org/

[73] I. Jacobson, M. Christerson, P. Jonsson, et G. Övergaard, *Object-Oriented Software Engineering - A Use Case Driven Approach*, 528 p., Addison Wesley, Reading, Ma, 1992.

[74] C. Gransart, J.-M. Geib, et P. Merle, *CORBA: Des concepts à la pratique*, 384 p., Dunod, 1999.

[75] M.-C. Roch, P.-A. Muller, et B. Thirion. "Mise en oeuvre d'UML pour la modélisation objet de publipostages complexes". Presented at *Object Expo 97*, Paris, Novembre 1997.

[76] M.-C. Roch, P.-A. Muller, G. Metzger, et B. Thirion. "Modélisation d'une application de composition automatique de documents avec UML". In *Proceedings of the 10th International Conference on Software Engineering and its Applications*, pp., Paris, 5 décembre 1997.

[77] J. L. Shaerer, A. T. Murphy, et H. H. Richardson, *Introduction to system dynamics*, 380 p., Addison-Wesley, Reading, 1971.

[78] P. de Larminat, *Automatique - Commande des systèmes linéaires*, 352 p., Hermes, Paris, 1993.

[79] J. M. Flaus. "Modeling and analysis of hybrid dynamical systems: an introduction",

Journal européen des systèmes automatisés, vol. 32 (7-8), 1998.

[80] M. Boasson. "Control System Software", *IEEE Trans. on Automatic Control*, vol. 38 (7), 1993.

[81] C. Foulard, S. Gentil, et J.-P. Sandroz, *Commande et régulation par calculateur numérique - de la théorie aux applications*, 575 p., Eyrolles, 1987.

[82] J. P. Perez, *Systèmes temps réel - Méthodes de spécification et de conception*, 243 p., Dunod, 1990.

[83] M. Rivoire et J.-L. Ferrier, *Cours d'automatique, Commande par calculateur - Identification*, vol. Tome 3, 240 p., Eyrolles, 1997.

[84] K. J. Aström et K.-E. Arzen, *An introduction to intelligent and autonomous control*, Expert Control, 427 p., Kluwer academic publishers, Norwell, 1993.

[85] D. Driankov, H. Hellendoorn, et M. Reinfrank, *An introduction to fuzzy control*, 316 p., Springer-Verlag, Berlin, 1993.

[86] N. Gaertner et B. Thirion. "A framework for fuzzy knowledge based control", *Software - Practice and Experience*, vol. 30, pp. 1-15, 1999.

[87] N. Kettani et P.-A. Muller. "Validating UML Models Through Prototyping", *Rose Architect*, vol. 2 (1), pp. 20-25, 1999.

[88] P.-A. Muller. "An example of UML model level Prototyping". In *Proceedings of the 11th IEEE/IFIP International Workshop on Rapid System Prototyping*, pp., Paris, 23 Juin 2000.

[89] O. Burgard et P.-A. Muller. "Représentation d'un modèle UML codé en XMI". Presented at *Journées GRACQ*, Nantes, 7 juin 1999.

[90] O. Burgard. "Représentation d'un modèle UML codé en XMI dans Internet Explorer 5". Presented at *Journées COOSI*, Paris, 16 septembre 1999.

[91] O. Burgard, P.-A. Muller, et B. Thirion. "Vers une programmation unifiée avec XSLT: applications aux serveurs Web". Presented at *Journées du CRESPIM (Centre de Recherche et d'Enseignement en Sciences pour l'Ingénieur - Mulhouse)*, Mulhouse, 24 janvier 2004.

[92] P.-A. Muller. "Netsilon: Le développement web sans tracas", *Internet Professionnel*, vol. juin-juillet 2002, pp. 92-93, 2002.

[93] P.-A. Muller. "MDA: Une réalité, un rêve? Retour d'expérience". Presented at *7ème réunion du groupe de travail « Ingénierie de la modélisation », OFTA*, Paris, 7

octobre 2002.

[94] P.-A. Muller. "Platform Independent Web Application Modeling". Presented at *Seminar on Language Engineering for Model-Driven Development*, Dagstuhl, 5 mars 2004.

[95] P.-A. Muller, P. Studer, et J. Bézivin. "Platform Independent Web Application Modeling." In *Proceedings of the UML 2003 - ACM/IEEE 6th International Conference on The Unified Modeling Language*, pp. 220-233, San Francisco, CA, Lecture Notes in Computer Science, 2863, October 2003.

[96] T. Cros, *Maîtriser les projets avec l'extrême programming*, 186 p., Cépaduès, Toulouse, 2004.

[97] W. El Kaim, O. Burgard, et P.-A. Muller. "MDA Compliant Product Line Methodology, Technology and Tool for Automatic Generation and Deployment of Web Information Systems". In *Proceedings of the 14th International Conference on Software Engineering and its Applications*, pp., Paris, Décembre 2001.

[98] W. El Kaim, O. Burgard, et P.-A. Muller. "Outils, Méthodologies et Technologies de produits MDA pour la génération et le déploiement automatique de systèmes d'information Web", *Génie Logiciel*, vol., pp. 23-33, 2002.

[99] P.-A. Muller, D. Bresch, et P. Studer. "Model-Driven Architecture for Automatic-Control: An Experience Report." In *Proceedings of the UML 2004 - ACM/IEEE 7th International Conference on The Unified Modeling Language*, pp. 260-274, Lisbon, Lecture Notes in Computer Science, 3273, October 2004.

[100] P.-A. Muller et D. Bresch. "Model-Driven Architecture for Distributed and Embedded Process-Control". Presented at *CETSIS'05: cinquième colloque sur l'enseignement des technologies et des sciences de l'information et des systèmes*, Nancy, 25-27 Octobre 2005.

[101] P.-A. Muller. "A propos des niveaux de modélisation dans l'approche MDA, un exemple de paramétrage d'application". Presented at *Journées Meta*, Vannes, 6 février 2003.

[102] J. Bézivin, S. Gérard, P.-A. Muller, et L. Rioux. "MDA Components: Challenges and Oppotunities". In *Proceedings of the International Workshop on Metamodelling for MDA*, pp. 23 - 41, York, England, November 2003.

[103] P.-A. Muller, C. Dumoulin, F. Fondement, et M. Hassenforder. "The TopModL

Initiative". In *Proceedings of the WISME 2004: 3rd Workshop in Software Model Engineering (Satellite Event of UML 2004)*, pp. 242-245, Lisbon, Portugal, Lecture Notes in Computer Science, 3297, March 2005.

[104] OMG., MOF QVT Final Adopted Specification, *Electronic Source: Object Management Group*, November 2005, http://www.omg.org/docs/ptc/05-11-01.pdf

[105] J.-M. Jézéquel, *Object-oriented software engineering with Eiffel*, 320 p., Addison Wesley Longman Publishing, Redwood City, 1996.

[106] E. Gamma, R. Helm, R. Johnson, et J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 395 p., Addison-Wesley, 1995.

[107] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, et W. Lorensen, *Object-Oriented Modeling and Design*, 500 p., Prentice Hall, Englewood Cliffs, 1991.

[108] P.-A. Muller et J.-M. Jézéquel. "Model-driven generative approach for concrete syntax composition". Presented at *Best Practices for Model Driven Software Development'04 (OOPSLA & GPCE Workshop)*, Vancouver, 25 octobre 2004.

[109] OMG, Human-Usable Textual Notation, *Electronic Source: Object Management Group*, August 2004, http://www.omg.org/technology/documents/formal/hutn.htm

[110] P.-A. Muller et M. Hassenforder. "HUTN as a Bridge between ModelWare and GrammarWare - An Experience Report". Presented at *WISME 2005: 4th Workshop in Software Model Engineering (Satellite Event of MoDELS 2005)*, Montego Bay, October 3rd 2005.

[111] P.-A. Muller, F. Fleurey, D. Vojtisek, Z. Drey, D. Pollet, F. Fondement, P. Studer, et J.-M. Jézéquel. "On Executable Meta-Languages applied to Model Transformations". Presented at *MTiP 2005, International Workshop on Model Transformations in Practice (Satellite Event of MoDELS 2005)*, Montego Bay, Jamaica, October 2005.

[112] D. H. Akehurst, B. Boardbar, A. Evans, W. G. J. Howells, et K. D. McDonald-Maier. "SiTra: Simple Transformations in Java". In *Proceedings of the MoDELS'06: 9th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pp. 351-364, Genova, Lecture Notes in Computer Science, 4199, October 2006.

[113] Caroll, "The Carroll Research Program."

[114] C. Nebut, F. Fleurey, Y. Le Traon, et J.-M. Jézéquel. "Requirements by contracts allow automated system testing". In *Proceedings of the ISSRE'03:14th IEEE*

International Symposium on Software Reliability Engineering, pp., Denver, CO, USA, November.

[115] C. Nebut et F. Fleurey. "Une méthode de formalisation progressive des exigences basée sur un modèle simulable". In *Proceedings of the LMO'05: Conférence sur les Langages et Modèles à Objets*, pp. 145-158, Bern, Switzerland, L'Objet logiciel, bases de données, réseaux, RSTI série l'Objet, 11 N° 1-2/2005.

[116] C. Nebut, F. Fleurey, Y. Le Traon, et J.-M. Jézéquel. "Automatic Test Generation: A Use Case Driven Approach", *IEEE Transactions on Software Engineering*, vol. 32 (3), pp. 140-155, 2006.