Frédéric Fondement Curriculum Vitæ détaillé

1. Plan

2. Information essentielles

2.1. Résumé

Depuis septembre 2008, je suis Maître de Conférences stagiaire après un emploi d'une année en tant qu'Attaché Temporaire d'Enseignement et de Recherche (1/2 ATER) au sein de l'École Nationale d'Ingénieurs Sud-Alsace (ENSISA) de l'Université de Haute-Alsace.

Auparavant, j'ai assuré pendant quatre années des fonctions d'assistant d'enseignement et de recherche à l'École Polytechnique Fédérale de Lausanne (EPFL). Cette période correspond également à la préparation de mon doctorat, au sein du Laboratoire de Génie Logiciel (LGL). J'ai également eu l'occasion d'occuper des fonctions d'ingénieur expert en développement d'application au sein de l'équipe Triskell de l'INRIA, et d'ingénieur logiciel dans la PME ObjeXion Software SA, jeune pousse fondée par un enseignant chercheur dans le cadre de la loi sur l'innovation.

A l'occasion de chacune de ces expériences, j'ai pu me consacrer à des recherches toutes articulées autour de la modélisation de systèmes à dominante logicielle. Je m'intéresse tout particulièrement à la modélisation des langages de modélisation (métamodélisation), à leur utilisation et réutilisation pratique et agile dans le cadre de l'Ingénierie Dirigée par les Modèles (IDM), ainsi qu'à la définition de leur syntaxe concrète, qu'elle soit graphique ou textuelle.

Ces diverses expériences m'ont également permis de concevoir et d'assurer des cours complet de génie logiciel ainsi que d'IDM, de suivre des travaux pratiques sous forme de projets de génie logiciel avec UML, de concevoir et suivre deux projets de semestre, ainsi que d'assurer des formations professionnelles sur la modélisation d'applications internet.

Par ailleurs, j'ai également eu l'occasion de participer à divers projets de développement logiciel, et ai été responsable de suivi d'équipe dans le cadre d'un projet de recherche européen. Je suis également responsable scientifique pour la part UHA du projet VETESS et encadre un ingénieur de recherche à plein temps.

Ce parcours correspond à une découverte progressive des métiers de la recherche. J'ai en effet commencé par un DRT effectué dans une entreprise innovante, puis intégré un laboratoire de recherche INRIA en tant qu'ingénieur expert. Fort de ces expériences, et plus au fait des réalités du métier, j'ai souhaité m'orienter plus spécifiquement vers la recherche et l'enseignement en préparant un doctorat à l'EPFL, puis en intégrant un poste d'ATER puis de maître de conférences à l'UHA.

Si mon thème directeur a été jusqu'à présent l'IDM, j'ai toujours porté attention au côté pratique de cette technique particulièrement adaptée aux langages dédiés à un domaine précis. Ainsi, je me suis par exemple intéressé aux techniques de compilation, aux langages à objets, aux applications internet, aux systèmes d'information, au raffinement, aux aspects, aux composants logiciels, aux familles de produits, ou aux langages graphiques.

2.2. Informations essentielles

NUMEN	15S0802339UDC

Coordonnées personnelles	19, rue du Ballon 68700 Uffholtz 03 89 39 28 43 06 73 71 03 66 f.fondement@laposte.net
Coordonnées professionnelles	Université de Haute Alsace École Nationale Supérieure d'Ingénieurs Sud Alsace (ENSISA) 12, rue des frères Lumière 68 093 Mulhouse Cedex 03 89 33 69 78 frederic.fondement@uha.fr http://fondement.free.fr/lgl/
Naissance	Le 27 Décembre 1977, à Montbéliard (Doubs - 25)
Nationalité	Française
Situation actuelle	**Maître de Conférences stagiaire** Université de Haute Alsace (UHA) École Nationale Supérieure d'Ingénieurs Sud Alsace (ENSISA) Laboratoire Modélisation Intelligence Processus Systèmes (MIPS - EA 2332) groupe Logiciels et Systèmes Intelligents (LSI) 12, rue des frères Lumière 68 093 Mulhouse Cedex
Diplômes	DUT GEII, ingénieur ESSAIM en informatique et automatique, DRT GEII, Doctorat ès Sciences EPFL *(voir section 2.3, page 4)*.
Recherche	Génie Logiciel par l'Ingénierie Dirigée par les Modèles: transformations de modèles, réutilisabilité des métamodèles et des transformations, spécifications de syntaxes concrètes textuelles et graphiques. Participation aux projets *FacSimile*, *Netsilon*, *MTL*, *Fondue Builder*, *Parallax*, *SVG-Based Modeling Tools*, *Blanddern*. Participation au projet de recherche VETESS. Mots-Clefs: Ingénierie dirigée par les modèles, métamodélisation, transformation de modèles, syntaxes graphiques, syntaxes textuelles, MOF, JMI, OCL, UML. Publications d'articles scientifiques (2 articles de revue internationale - *SoSyM*, 2 chapitres de livre, 4 conférences internationales avec publication des actes - *Models*, *ECMDA-FA* - 2, *EDOC*, et 4 ateliers internationaux avec comité de lecture), *voir page 19*. Participation aux comités de programme des conférences nationales *IDM 2005*, *IDM 2006*, et *IDM 2008*. Lecteur référent, notamment pour la revue internationale *SoSyM*, et le cycle de conférences internationales *UML/Models*. *(voir section 3, page 8)*

Enseignement	Enseignements à l'École Polytechnique Fédérale de Lausanne (EPFL): - niveau L3, génie logiciel avec la méthode Fusion et la notation UML/OCL (sur 2 ans), 70 étudiants, 4h30 CM, 2h TD, 24h TP - niveau L3, 2 projets de semestre, niveau L3, représentation en SVG de modèles MOF, 56h TP Enseignements à l'École Nationale Supérieure d'Ingénieurs Sud Alsace (ENSISA): - niveau M1, ingénierie dirigée par les modèles, 21 étudiants (2007-8) puis 34 (2008-9), cumul de 10h cours, 24h TD, 168h TP - niveau M1, génie logiciel, 33 étudiants, 20h cours, 26h TD, 36 TP - niveaux M1 et M2, 4 projets de semestre sur 2 années, 96h TP **pour un total de 357 heures éq. TD** *(voir section 4, page 30)*
Administration	Suivi d'équipe et établissement de délivrables pour le projet de recherche européen ITEA CAFE sur les lignes de produits logiciels. Encadrement d'un ingénieur de recherche et responsable scientifique pour la part UHA du projet VETESS.

2.3. Titres universitaires

Novembre 2007:	**Doctorat** ès Sciences au Laboratoire de Génie Logiciel (LGL) de l'École Polytechnique Fédérale de Lausanne (EPFL), Suisse pour la thèse 3927 intitulée «Concrete syntax definition for modeling languages.» suite à la soutenance privée du 24 Septembre 2007 et présentée à Lausanne en soutenance publique le 2 Novembre 2007. Mention: accepté sans réserve Composition du jury: *Directeur*: Dr. Thomas Baar (EPFL, désormais Tech@Spree) *Président du jury*: Prof. Emre Telatar (EPFL) *Rapporteur*: Prof. Pierre-Alain Muller (UHA / INRIA) *Rapporteur*: Prof. Bernhard Rumpe (Technische Universität Braunschweig) *Rapporteur*: Prof. Alain Wegmann (EPFL) Mots-clefs: Ingénierie dirigée par les modèles, Métamodélisation, Ingénierie des langages, Syntaxes concrètes, Syntaxes textuelles, Syntaxes graphiques, Scalable Vector Graphics (SVG)

Décembre 2001:	**Diplôme de Recherche Technologique** (DRT) en Génie Electrique et Informatique Industrielle (GEII) de l'Université de Haute Alsace (UHA) pour le mémoire intitulé «Création d'un langage d'action pour un logiciel MDA.» décrivant le stage accompli chez ObjeXion Software S.A à Vieux-Thann (68). Mention: très bien. Composition du jury: *Président et Responsable de stage*: Prof. Michel Hassenforder (UHA) *Maître de Stage*: Dr. Philippe Studer (ObjeXion - désormais UHA) *Examinateur*: Prof. Jean-Marc Jézéquel (INRIA) *Examinateur*: Prof. Bernard Thirion (UHA) *Examinateur*: Olivier André (ObjeXion) Mots-clefs: Netsilon, Xion, Modélisation d'applications internet, Langage d'action, Object Constraint Language (OCL), Java, UML, SQL, Compilateurs, Model Driven Architecture (MDA).
Juillet 2000:	**Diplôme d'Ingénieur** (habilité CTI) de l'École Supérieure des Sciences Appliquées pour l'Ingénieur de Mulhouse (ESSAIM - désormais ENSISA) de l'Université de Haute Alsace (UHA). Stage de fin d'études intitulé «Développement d'un interpréteur OCL pour une machine virtuelle UML» accompli chez ObjeXion Software S.A à Vieux-Thann (68). *Responsable de stage*: Prof. Michel Hassenforder (UHA) *Maître de stage*: Dr. Philippe Studer (ObjeXion - désormais UHA) Rang: 2/61
Juillet 1997:	**Diplôme Universitaire de Technologie** (DUT) en Génie Electrique et Informatique Industrielle (GEII) option Electrotechnique de l'Institut Universitaire de Technologie de Belfort-Montbéliard Stage de fin d'études intitulé «Développement de gestionnaires de PARTS sous Navigator» chez Peugeot S.A. à Sochaux (25 - service DETA/MSC/DCL/SUP). *Responsable de stage*: Jean-Pierre Barvidat (IUT Belfort) *Maître de stage*: Emmanuel Chamouton (PSA) Rang: 3/69
Juillet 1995:	**Baccalauréat** Scientifique (série S) obtenu à Belfort (90) Mention: Passable

2.4. Parcours professionnel

Depuis Septembre 2008:	**Maître de Conférences stagiaire** au Laboratoire Modélisation, Intelligence, Processus, Systèmes (MIPS) de l'École Nationale Supérieure d'Ingénieur Sud Alsace (ENSISA) de l'Université de Haute Alsace (UHA) à Mulhouse (68) *Responsables*: Prof. Michel Hassenforder, Prof. Bernard Thirion Recherches et Enseignements autour de l'Ingénierie Dirigée par les Modèles et du Génie Logiciel; encadrement de projets. Démarrage du projet *Blanddern*.
Septembre 2007 à Septembre 2008:	**Attaché Temporaire d'Enseignement et de Recherche** (1/2 ATER) au Laboratoire Modélisation, Intelligence, Processus, Systèmes (MIPS) de l'École Nationale Supérieure d'Ingénieur Sud Alsace (ENSISA) de l'Université de Haute Alsace (UHA) à Mulhouse (68) *Responsables*: Prof. Michel Hassenforder, Prof. Bernard Thirion Recherches et Enseignements autour de l'Ingénierie Dirigée par les Modèles. Poursuite des projets *SVG-Based Modeling Tools*, *Sintaks et TCSSL Tools*.
Octobre 2003 à Décembre 2006:	**Assistant d'Enseignement et de Recherche** au Laboratoire de Génie Logiciel (LGL) de l'École Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Suisse *Responsable*: Dr. Thomas Baar (Enseignant Chercheur) Recherches et Enseignements autour de l'Ingénierie Dirigée par les Modèles et préparation du doctorat. Participation aux projets *Fondue Builder*, *Parallax*. Mise en place et développement du projet *SVG-Based Modeling Tools*.
Décembre 2002 à Septembre 2003:	**Ingénieur expert** de recherche et développement dans l'équipe Triskell de l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) de l'Institut National de Recherche en Informatique et en Automatique (INRIA) Rennes (35) *Responsable*: Prof. Jean-Marc Jézéquel Participation à la conception et au développement d'un langage de transformation de modèles (projet *MTL*). Suivi d'équipe pour le projet de recherche européen ITEA CAFE.

Septembre 2000 à Juillet 2002:	**Ingénieur logiciel** dans l'équipe de recherche et développement d'ObjeXion Software S.A., Vieux-Thann (68) *Responsable*: Dr. Philippe Studer Etudes et développements de divers aspects de modeleurs d'applications (projets *Netsilon* et *FacSimile*), incluant la conception et la réalisation d'un langage d'action pour UML, sa transformation partielle en SQL, d'une application internet pour l'administration d'objets. Réalisation d'applications internet grâce aux outils. Réalisation de formations aux outils, support et documentation.

2.5. Compétences et connaissances

Informatique:

- Génie logiciel (Ingénierie dirigée par les modèles, métamodélisation, méthode Fondue, JUnit, …)
- Architecture objet (Java, C++, UML/OCL, Rose, Objecteering, …), aspects (tisseur pour MTL)
- Techniques de compilation (ANTLR, réalisation d'un interprète avec contrôle des types pour OCL, compilateurs pour les langages Xion - vers SQL - et MTL - vers Java)
- Environnements de développement (Eclipse, JBuilder, CVS, ANT, NSIS, viewcvs, statcvs, …)
- Systèmes d'information (SGBDR comme MySQL, SQL, mapping objet-relationnel, JMI MDR, EMF,...)
- Architectures n-tiers (RMI, Sockets, XML/DOM, SVG, HTML, PHP, Servlet, Javascript, Apache, Tomcat, Resin, LAMP)
- Informatique et automatique industriels (C, multitâche, grafcet)
- Outils de diffusion (Moodle, Adobe FrameMaker, Microsoft Word et PowerPoint, Adobe InDesign, HTML, LaTeX...)
- Étude des besoins, architecture logicielle, tests, réalisation de documentations, formation, support

Anglais (langue de travail, rédaction d'articles scientifiques, présentations)

3. Activités de recherche

Ce chapitre se base sur des références bibliographiques données en section 3.12, page 22 sous forme numérique (par exemple [14]). Les papiers pour lesquels je suis co-auteur et dont la liste se trouve en section 3.9, page 19 sont référencés sous forme alphanumérique (par exemple [FB05]). Les projets mentionnés sont eux résumés en section 3.13, page 25.

Mes axes de recherche ont pour cadre l'Ingénierie Dirigée par les Modèles (IDM), notamment la définition et le support des langages de modélisation. Je me suis particulièrement concentré sur les aspects méthodologiques quant à l'utilisation de cette technique, mais aussi sur les aspects pratiques, avec un certain nombre d'études de cas, qui ont montré certains manques comme la spécification agile de syntaxes concrètes réutilisables pour les langages de modélisation.

Ma carrière a été marquée par la conception et le support de langages de modélisation à tout niveau d'abstraction. Par exemple, j'ai participé à la création d'un langage de transformation de modèles impératif et orienté objet (projet *MTL*), technologie vitale à l'utilisabilité de l'IDM, que j'ai plus tard amélioré pour y intégrer la programmation par aspects [SFS04b] (projet *Parallax*). Je me suis également concentré sur UML en créant un langage d'action [MSFB05] (projet *Netsilon*) et en développant un interpréteur OCL (project *FacSimile*), tous deux destinés à la gestion de modèles d'objets instances de classes UML. A côté de ces langages textuels compilés (Xion) ou interprétés (OCL), j'ai participé à la réalisation de modeleurs graphiques tel un modeleur et générateur d'applications internet [MSFB05] (projet *Netsilon*), ainsi qu'un IDE pour le support de la méthode Fondue [BMFS06] (projet *Fondue Builder*) qui permet la conception de systèmes réactifs en alliant la méthode Fusion à la notation UML.

Ces activités, à l'échelle de l'IDM, s'étendent sur une longue période. J'ai donc découvert au fur et à mesure, de façon très concrète, l'intérêt de techniques telles que la métamodélisation, les dépositaires de modèle, ou les langages de transformation de modèle [MFV+05], tout en mettant à profit mes connaissances des techniques objet, de compilation, de conception d'interfaces, etc. Cependant la création de langage de modélisation semblait encore possible à améliorer. C'est dans cet esprit que j'ai intégré le groupe TopModL [MDFH04].

Dans ce cadre, et fort de mon expérience en développement de langages, je me suis attelé à l'étude de la définition de syntaxes concrètes appliquées à l'IDM. Je me suis donc attaché à l'étude des syntaxes textuelles [MFF+08, MFF+06, FSGM06] (projets *Sintaks et TCSSL Tools*), ainsi qu'à celle des syntaxes graphiques [FB05, Fon08] (projet *SVG-Based Modeling Tools*).

Au-delà de réflexions fondamentales sur la modélisation [MFB09], je me suis également penché sur l'utilisation méthodologique des langages [FS04], et plus spécifiquement sur l'intégration de plates-formes. Je me suis par exemple intéressé à l'intégration des aspects de distribution à une architecture logicielle grâce à une hiérarchie de décorateurs type profil UML [SFS04a] (projet *Parallax*), et à la modélisation de familles de produits logiciels [ZJF03], toutes deux supportées par des transformations de modèles.

La suite de cette section décrit un peu mieux l'IDM, les recherches auxquelles j'ai pu participer, ainsi que les perspectives que j'envisage. Je fournis également la liste des publications référencées ci-dessus, que j'ai pu co-produire au cours de ces recherches, et une liste de papiers de référence sur lesquels je m'appuie dans cette section. Finalement, je fournis une description synthétique des projets de développement logiciel auxquels j'ai participé, et qui ont contribué, ou sont le résultat, des recherches décrites ici.

3.1. Contexte

L'évolution du matériel informatique a rendu possible l'exécution de logiciels de plus en plus complexes. Pour faire face à cette complexité, le génie logiciel a nécessairement dû évoluer en trouvant toujours de nouvelles approches pour simplifier la création de logiciels. Deux grandes catégories d'approches peuvent être citées: les *techniques d'abstraction*, et les *méthodes*.

Les techniques d'*abstraction* ont permis aux développeurs de décrire leurs systèmes logiciels en des termes qui, peu à peu, se rapprochaient plus de la pensée humaine que du langage machine. C'est ainsi qu'on a vu apparaître les langages de seconde (assembleurs) ou de troisième (C, Java) génération. Tout en restant supportés par des outils réalisant leurs promesses, certains langages se sont également spécialisés pour résoudre des problèmes d'un domaine plus précis (les langages de quatrième génération, par exemple SQL ou SVG), voire même se sont limités à la description du problème (les langages de cinquième génération comme regexp).

Les *méthodes* permettent de piloter les développements logiciels en identifiant les acteurs et leurs activités. Il peut s'agir de règles bien précises (comme B [1]), ou plus simplement de règles de bonne conduite (comme pour eXtreme Programming [2]). Certains considèrent d'ailleurs que les processus de développement logiciel peuvent être vus comme des processus logiciels [3].

C'est sur ces deux plans qu'intervient l'Ingénierie Dirigée par les Modèles (IDM) [4], en organisant l'abstraction ainsi qu'en permettant, au moins partiellement, de décrire des méthodes de développements. L'IDM promeut l'utilisation de modèles comme éléments principaux de la description d'un logiciel, en suivant une approche par étapes, dans lesquelles sont développés des modèles abstraits, peu à peu améliorés pour y intégrer les détails dont ont besoin la ou les plates-formes de déploiement finales. L'originalité de l'IDM est d'automatiser les processus de développements. D'une part, les modèles sont exprimés dans un langage de modélisation: l'IDM propose le mécanisme de la métamodélisation pour décrire la syntaxe abstraite de ces langages [5]. D'autre part, les modèles peuvent être mis en relation par des transformations de modèles [6]. C'est dans les métamodèles qu'on pourra retrouver une modélisation de l'*abstraction* offerte par les langages de modélisation. De plus, une fois choisis les langages de modélisation et les transformations de modèles les reliant, on se retrouve face à un cadre *méthodologique* pilotant les développements [FS04].

En outre, et suivant les principes qui ont conduit au développement des langages de quatrième génération [7], de nombreux auteurs insistent sur l'importance du choix du langage de modélisation pour un problème donné. Cependant, une démarche de développement pilotée par les modèles peut faire intervenir, pour chaque projet de développement, de nombreux domaines à de nombreux niveaux d'abstraction. Une conséquence immédiate de ces principes est une prolifération des langages de modélisation, qui peuvent parfois avoir besoin d'être adaptés à un problème précis. Nous avons le sentiment qu'il y a beaucoup à attendre d'une ingénierie des langages de modélisation simplifiant la création et l'outillage de ces langages, et c'est dans cette direction que se sont axées mes recherches.

Mes travaux ont tous tourné autour de l'IDM, que ce soit pour en appliquer les principes par la fourniture d'outils supportant un langage de modélisation donné (section 3.2, section 3.3), ou, plus récemment, par l'amélioration de ses principes et techniques (section 3.4, section 3.5, section 3.6). Les perspectives de recherches que je propose (section 3.8) tiennent en la poursuite de cette dernière direction.

3.2. Interpréteur OCL pour une machine virtuelle UML (2000)

Ce travail a eu pour cadre mon stage d'ingénieur au sein de la PME ObjeXion Software, à Vieux-Thann (68) en 2000.

Problématique. UML [8] est un langage généraliste de modélisation de systèmes à prépondérance logicielle. La jeune pousse ObjeXion Software avait développé un prototypeur de modèles UML, nommé FacSimile. Ce prototypeur est capable de lire un modèle de classes UML et de générer un environnement logiciel dans lequel il est possible d'instancier des objets, de définir la valeur de leurs paramètres, et d'indiquer leurs relations. Toutes ces informations restent persistantes grâce à une base de données relationnelles, suivant les prescriptions données par [9].

OCL [10] est un langage dont le premier standard remonte à 1997. Il s'agit d'un langage de contraintes (sans effet de bord) adapté à UML qui permet notamment de préciser les diagrammes de classes. Il est par exemple possible de contraindre la valeur d'un attribut à dépendre du modèle d'instance correspondant. OCL peut également être utilisé comme langage de requête, par exemple pour trouver les instances de classe qui vérifient certaines propriétés. Mon but était d'intégrer un interpréteur OCL à FacSimile.

Résultats et réalisations. J'ai été en charge de l'intégration d'un interpréteur OCL à l'outil de prototypage de modèle UML (projet *FacSimile*). L'interpréteur réalise un contrôle des types statique, se base sur l'exploration du prototype. L'interpréteur est intégré à l'outil et permet de réaliser:

- l'évaluation d'une requête simple,
- la recherche d'instances vérifiant (ou violant) une contrainte,
- l'implémentation d'une opération sans effet de bord,
- la vérification d'un modèle en fonction des contraintes décrites dans un fichier annexé au prototype.

Diffusion. Les développements on été intégrés à l'outil *FacSimile (p. 29)* de la firme ObjeXion et furent commercialisés. Une version de démonstration du logiciel (limité à 20 classes) est en téléchargement libre[1], et a été distribuée avec le livre «Modélisation Objet avec UML» (ISBN 2-212-09122-2), de Pierre-Alain Muller, diffusé à environ 40 000 exemplaires.

3.3. Modélisation d'applications internet (2001-2002)

Ce travail a eu pour cadre mon stage de Diplôme de Recherche Technologique, suivi de mon embauche en tant qu'ingénieur logiciel, au sein de la PME ObjeXion Software, à Vieux-Thann (68) en 2001 et 2002.

Problématique. Une application internet contient fréquemment des aspects de présentation, de cinématique, et de données persistantes. Afin de rendre les données persistantes, les développeurs d'applications internet utilisent le plus souvent une base de données relationnelle. Ces données sont utilisées dans une application exécutée par le serveur de pages internet afin de produire lesdites pages suivant des prescriptions graphiques et cinématiques (le plus souvent HTML).

1. http://fondement.free.fr/objx/modelprototyper/

Nous avons montré dans la section précédente que des données objet, modélisées par un diagramme de classes UML, peuvent être stockées en base de données relationnelle [9]. C'est ainsi que nous proposons de modéliser une application internet en s'appuyant sur les diagrammes de classes plutôt que de la développer directement pour des outils dédiés. Le modèle ainsi produit peut donc être "compilé" vers des plates-formes concrètes différentes. Dans le cas d'applications internet, les plates-formes nécessaires sont de deux types: les plate-forme dédiées au *stockage de données* (le plus souvent des bases de données relationnelles), et les plates-formes dédiées à l'*exécution d'application*, exécutant le programme de l'application qui peut être par exemple donné en PHP ou en Java.

Résultats et réalisations. La modélisation que nous proposons suit trois étapes. La première est de modéliser les données sous forme de diagrammes de classe (le modèle métier). La seconde est de décrire les aspects graphiques en produisant des exemples pertinents de code HTML (ou tout autre format textuel). La troisième est de modéliser la cinématique du site en fonction des données (le modèle de navigation).

Pour ce faire, nous avons proposé un langage de modélisation dédié à la description de la cinématique d'une application internet. Ce langage permet de composer les aspects graphiques en fonction des données. Ce modèle, lui aussi indépendant de la plate-forme d'exécution, a pu être traduit indifféremment dans les langages PHP et Java. Les grands principes de ce langage sont les suivants. Chaque page, dont la charte graphique est donné par un patron sous forme de texte, peut présenter des données (directement ou calculées), intégrer d'autres pages suivant des conditions de données, et lier d'autres pages également sous condition.

Pour consulter ou manipuler les données au sein du modèle de cinématique, par exemple pour exprimer les conditions sus-citées, nous avons proposé un langage dédié. OCL peut être vu comme un langage de requêtes sur des données objets modélisées sous forme de diagramme de classes. Si un tel langage peut être utilisé pour explorer les données, il ne peut que difficilement les manipuler. C'est ainsi que nous avons proposé Xion, un langage d'action à typage statique qui ajoute à OCL un style impératif ainsi que les effets de bords (création/destruction d'objet, modification des associations et des propriétés).

Les langages de modélisation proposés ont été supportés par l'outil *Netsilon*, qui offre un environnement de modélisation pour les diagrammes de classes modélisant les données, les patrons textes, le modèle de navigation graphique, et le langage textuel d'action Xion. De plus, l'outil est capable de générer et déployer à partir de ces modèles une application internet complète pour les plates-formes de *stockage de données* MySQL, Oracle 8i et PostgreSQL, et les plates-formes d'*exécution* PHP, JSP et Java Servlet.

Au même titre que d'autres types d'applications, les applications internet peuvent présenter des comportements communs, comme par exemple un système d'authentification ou un forum (lequel peut requérir lui-même un système d'authentification). C'est pourquoi nous avons également rendu possible la définition de composants internet constitués des mêmes types de modèles. Les pages internet peuvent être des pages requises ou fournies, et des interfaces sont soit requises soit fournies en fonction de leur réalisation ou non par une classe du modèle métier. Une mise en correspondance des éléments requis avec des éléments fournis permet la réutilisation de ces composants au sein d'un modèle d'applications internet.

Pour résumer, nous avons proposé une série de langages indépendants des plates-formes de déploiements pour la modélisation d'applications internet complètes et réutilisables. Ces langages peuvent par après être compilés pour cibler un certain nombre de plates-formes. C'est ainsi que nous avons appliqué l'IDM au domaine de la création d'applications internet [11].

Diffusion. *Netsilon (p. 28)* a été commercialisé par la société ObjeXion Software et a été utilisé pour la conception et le déploiement de sites internet commerciaux comme Domaine.fr (voir http://www.domaine.fr/) ou PRH (http://www.prh-france.fr). Une version d'évaluation limitée à un site de déploiement est en téléchargement libre[1]. L'approche a été développée dans l'article [MSFB05]. J'ai eu également la responsabilité d'assurer des formations professionnelles aux clients d'ObjeXion quant à l'utilisation de Netsilon.

3.4. Transformation de modèles (2003)

Ce travail a pour cadre mon emploi d'ingénieur expert en développement d'applications au sein de l'équipe Triskell de l'INRIA en 2003.

Problématique. Les travaux sur Xion (voir la section précédente) ont montré l'efficacité d'un langage d'action sur la manipulation de modèle de classes UML. Or, le standard MOF relatif à la déclaration de syntaxes abstraites de langages de modélisation (les *métamodèles*) [12] est un sous-ensemble des diagrammes de classes UML. Xion peut donc servir à manipuler des (méta-)modèles MOF.

A côté de la métamodélisation [5], les transformations de modèles sont de première importance pour l'application d'un processus de développement de type IDM [6]. Nous avons donc fait évoluer Xion pour la transformation de modèles. Nous avons nommé cette évolution MTL (pour Model Transformation Language). MTL est donc un langage impératif pour la transformation de modèles. Cependant, par rapport à Xion, MTL intègre notamment la spécification de structures objet (classes, associations, héritage multiple) et la décomposition en librairies hiérarchiques (introduisant par le fait une certaine notion d'aspect [13]). MTL peut manipuler indifféremment des modèles stockés en dépositaires (EMF, MDR, etc.) ou des modèles constitués d'instances de classes MTL.

Résultats et réalisations. Afin de supporter MTL, nous avons développé un compilateur vers Java. Les développements ont suivi une approche dite de "bootstrap" dans laquelle le langage s'auto-décrit (langage autogène). En effet, nous avons développé un compilateur pour un sous-ensemble du langage (BasicMTL). La syntaxe abstraite de MTL a été décrite en BasicMTL, ce qui a permis de transformer, par un programme en BasicMTL, une spécification MTL en BasicMTL. Un exemple est l'intégration à MTL des associations n-aires que ne supporte pas BasicMTL. Par contre, BasicMTL supporte les références et la surcharge des accesseurs. Une transformation MTL a donc permis de transformer les associations en un ensemble de références complétées par des fonctions d'accesseurs de propriétés garantissant l'intégrité des données. C'est ainsi que MTL a pu être peu à peu enrichi.

Dans un souci de réutilisabilité et d'adaptabilité, MTL a été défini comme un langage modulaire. Une modularisation simple permet de définir une classe de transformations dans plusieurs unités de compilations (fichiers), en y ajoutant des propriétés (par exemple des attributs ou des relations). De plus, une modularisation de raffinement est possible. Les bibliothèques MTL dans lesquelles sont définies les classes proposent un mécanisme d'héritage de classes à l'instar des classes qui proposent un mécanisme d'héritage de propriétés. Les classes des librairies peuvent ainsi être raffinées en y ajoutant ou surchargeant des propriétés.

1. http://fondement.free.fr/objx/netsilon/

En termes de traduction, les classes MTL sont traduites en des interfaces Java répétant la structure d'héritage. De plus, pour chaque classe MTL, est générée une classe Java implémentant l'interface correspondante. Chacune des méthodes définies localement à la classe est traduite, alors que chacune des méthodes héritée est déléguée à une instance particulière dont la classe définit directement ladite méthode. La résolution des récepteurs de messages est ainsi achevée par le contrôle des types MTL et non Java. Les librairies sont des classes appliquant les patrons usines singletons (factories), où les méthodes d'instanciation de classes peuvent être surchargées. Les accès aux modèles à transformer se font à travers une API spécifique adaptée aux dépositaires voulus par des pilotes.

Diffusion. *MTL (p. 26)* est un projet "open source" supporté par l'INRIA [14]. Le projet a depuis évolué pour devenir le projet Kermeta [MFV+05, 15], dédié à l'exécutabilité des métamodèles. Le projet EJOSA, qui traite du développement d'application J2EE, propose l'utilisation de MTL comme plate-forme de développement des extensions modèle vers modèle. Cette approche, appliquée à la visualisation d'implications financières, a été développée par Lofi Dewanto dans le livre «Anwendungsentwicklung mit Model Driven Architecture - dargestellt anhand vollständiger Finanzpläne» (ISBN 978-3-8325-1480-8).

3.5. Utilisation et réutilisation des composants IDM (2004-2007)

Ce travail a pour cadre mon emploi d'ingénieur expert en développement d'applications au sein de l'équipe Triskell de l'INRIA en 2003, et fut poursuivi lors de ma thèse de doctorat, au sein du Laboratoire de Génie Logiciel de L'EPFL, à Lausanne (Suisse) de 2004 à 2006.

Problématique. L'IDM s'appuie sur les modèles. Nous avons vu que l'application d'un processus IDM se décrit par les langages de modélisation et les transformations de modèles utilisés. Nous appelons composant IDM tout élément permettant de supporter, décrire ou documenter une méthode supportée par l'IDM [16]. Les métamodèles (qui représentent la syntaxe abstraite des langages de modélisation), les transformations de modèles, et les spécifications conduisant la génération de code à partir de modèles sont des exemples de composants IDM.

Un exemple de processus IDM est celui décrit par la méthode Fondue [17], qui allie la notation UML à la méthode Fusion. Fondue possède quatre niveaux d'abstraction représentés par les phases de conception, d'analyse, de développement, et d'implémentation. Chacun de ces niveaux d'abstraction possède un certain nombre de langages pour exprimer un système sous forme de modèle. Par exemple, la phase d'analyse utilise une forme dérivée des notations des diagrammes de classes, des diagrammes d'états, et d'OCL.

Une première forme de réutilisation des composants IDM est de prévoir à l'avance quels sont les choix possibles dans une chaîne de développement. Les familles de produits logiciels permettent de décrire toute une chaîne de développement, en incluant les variants (c'est à dire les choix possibles), qui viendront ou non répondre aux points de variation [18]. Un modèle de variation permet de dériver la famille de produits en un produit donné directement utilisable.

Cependant, il est souvent nécessaire d'adapter un processus IDM à des préoccupations qui ne sont pas prévues en amont. Ceci revient à ajouter a posteriori des niveaux d'abstraction à un processus IDM. Par exemple, un modèle UML peut avoir à être complété pour tenir compte de préoccupations transverses telles que les aspects de distribution ou de sécurité. Le problème est que les transformations de modèles et les générateurs de code, au départ conçus pour UML, ne tiendront pas compte de ces modifications.

Résultats et réalisations. Nous avons modélisé la phase d'analyse de la méthode Fondue au sein d'un métamodèle et créé une suite d'outils (au sein du projet *Fondue Builder*) afin d'éditer graphiquement des modèles, de vérifier la cohérence de ces modèles, et de simuler le fonctionnement d'un système. Nous avons pu constater que le métamodèle est un moyen bien adapté pour définir la communication entre outils en jouant le rôle d'interface.

Ensuite, inspiré par la technique des familles de produits, nous avons étudié comment décrire les diagrammes de classe d'une ligne de produits, en indiquant les variants tout en spécifiant grâce à OCL les incompatibilités ou nécessités entre variants. La dérivation de produit est possible avec une transformation de modèle que nous avons définie et réalisée grâce à *MTL*, en s'appuyant sur un modèle de configuration.

Enfin, nous avons poursuivi ces recherches en étudiant l'intégration de préoccupations transverses en aval de la conception d'un processus IDM. Nous avons étudié une méthode pour ajouter des niveaux d'abstraction à des processus IDM existants, basée sur l'utilisation des mécanismes de décoration. Ces décorations doivent adapter les langages de modélisation par l'intermédiaire de leurs métamodèles (par exemple en utilisant des profils UML), tout en étant prises en compte par les transformations de modèles existantes.

Comme exemple de décoration de métamodèles, nous avons montré comment introduire des aspects de distribution dans un modèle de classes UML par une hiérarchie de profils. Un premier profil décrit les décorations qu'il est possible d'ajouter au modèle de classes. Par exemple, une décoration peut exprimer le fait qu'une interface soit visible à l'extérieur du système. Des profils plus concrets, complétant le premier profil, permettent d'indiquer comment cette distribution pourra être réalisée sur une plate-forme concrète, comme CORBA. Des transformations de modèles interactives (que nous avons réalisée en *MTL*) permettent d'automatiser le processus de décoration.

Pour faire en sorte qu'une transformation de modèle existante prenne en compte ces décorations, nous nous sommes inspirés de la programmation par aspect [13]. La programmation par aspect permet d'intégrer à un programme un certain nombre de modifications (dites advices) à des endroits (dits join points) donnés par des règles de recherche (dits pointcuts). Nous avons rendu le langage *MTL* (décrit en section 3.4) capable de gérer la programmation par aspect en utilisant la procédure décrite précédemment, soit en utilisant un mécanisme de décoration offert par MTL. Une transformation (en MTL) a été écrite afin de tisser un aspect MTL sur une transformation MTL.

Diffusion. La méthodologie utilisant les composants IDM est publiée dans l'article [FS04]. La modélisation des diagrammes de classes pour les familles de produits est donnée en [ZJF03], dans le cadre du projet européen ITEA CAFE. Les communications [SFS04a, SFS04b] donnent respectivement les démarches pour adapter UML et MTL à des besoins supplémentaires. Les projets "open source" *Fondue Builder (p. 27)* et *Parallax (p. 27)* permettent respectivement de supporter la phase d'analyse de la méthode Fondue, et de générer de manière extensible du code Java à partir de diagrammes UML étendus.

3.6. Syntaxes textuelles et graphiques (2004-2008)

Ce travail a pour cadre ma thèse de doctorat, au sein du Laboratoire de Génie Logiciel de L'EPFL, à Lausanne (Suisse) de 2004 à 2006, puis de mon poste actuel d'ATER, au sein du laboratoire MIPS de l'Université de Haute Alsace, Mulhouse (68).

Problématique. L'IDM fait une utilisation massive des modèles, et donc des langages de modélisation [19]. Pour définir un langage, de modélisation ou autre, il est nécessaire de produire trois éléments distincts: une syntaxe abstraite, une syntaxe concrète, et une sémantique [20]. Nous avons déjà mentionné la métamodélisation comme technique de définition de syntaxe abstraite. De nombreuses études traitent de la spécification de la sémantique d'exécution [21, 22]. C'est pourquoi nous nous sommes penchés sur la définition de syntaxes concrètes pour les métamodèles, qu'elles soient textuelles ou graphiques. Nos expériences sur OCL (section 3.2), Xion (section 3.3), MTL (section 3.4) et Fondue Builder (section 3.5) ont montré que dans la réalisation d'un langage, beaucoup d'énergie est dépensée dans la réalisation d'une interface permettant de visualiser des modèles. Les structures de textes peuvent depuis longtemps être définies par des grammaires de type EBNF [23], et les langages graphiques par des grammaires de graphes [24]. Notre travail a porté sur l'adaptation de ces techniques au monde de l'IDM.

Résultats et réalisations. Nous avons déterminé un langage spécifique afin de modéliser la syntaxe textuelle d'un langage dont la syntaxe abstraite est définie sous forme de métamodèle. Ce langage s'inspire d'EBNF, ainsi que de Netsilon (décrit en section 3.3) en ce qui concerne la manipulation des modèles. Nous avons bien sûr formalisé la syntaxe abstraite de ce langage par un métamodèle. Un premier prototype, nommé *Sintaks*, interprète la spécification de la syntaxe textuelle afin soit de représenter un modèle sous forme textuelle, soit de retrouver un modèle à partir de sa représentation textuelle. Pour cette dernière tâche, l'outil *TCSSL Tools* compile cette même spécification en une spécification pour compilateur de compilateur [25], et génère un programme de rendu orienté patron [26] pour représenter un modèle sous forme de texte.

La spécification de syntaxes graphiques est souvent plus interactive. Les concepts peuvent être représentés par des icônes qui peuvent être modifiées en suivant des règles prédéterminées. Dans une approche interactive, chaque modification doit être immédiatement répercutée sur le modèle ainsi visualisé. Nous avons introduit une définition des syntaxes graphiques en deux étapes: la *spécification* et la *réalisation*.

La *spécification* détermine tous les éléments d'une syntaxe graphique. Par exemple, pour représenter un diagramme d'états, nous aurons besoin de rectangles, de textes, de séparateurs, de liens, de bouts de liens, etc. Nous proposons de décrire et de mettre en relation ces éléments par un métamodèle. Chaque élément doit pouvoir connaître sa relation spatiale avec les autres (à côté de, connecté à, intégré à, en intersection avec, etc.). Un second métamodèle connecte cette syntaxe graphique à la syntaxe abstraite. Des règles de contraintes, en OCL, permettent de préciser les relations entre la syntaxe abstraite et les éléments graphiques. Il peut par exemple être dit que la représentation d'une relation doit être connecté à la représentation d'un noeud, si et seulement si les éléments représentés sont bien connectés dans le modèle instance de la syntaxe abstraite.

La *réalisation* tient les promesses de la spécification. Elle doit pouvoir représenter graphiquement les icônes sur une scène et permettre à la personne en charge de la manipulation du modèle un certain nombre d'interactions. Pour se faire, nous avons choisi d'expérimenter le standard SVG pour la représentation de dessins vectoriels [27]. Des patrons en SVG donnent la représentation graphique de chaque icône. La variabilité au sein des icônes est prise en compte par des contraintes dans un langage adapté à SVG [28]. Enfin, nous avons développé avec l'environnement DoPIdom [29] un certain nombre d'interactions standard afin de rendre le modèle manipulable. Des exemples d'interactions sont le déplacement, l'édition de texte, et le redimensionnement. Ces interactions intègrent un mécanisme d'événements permettant d'impacter le modèle manipulé.

Diffusion. Les idées sur les syntaxes textuelles ont été développées dans les articles [FSGM06, MFF+06, MFF+08] et les projets “open source” *Sintaks et TCSSL Tools (p. 25)*, en collaboration avec l’INRIA, l’Université de Haute-Alsace, et le Commissariat à l’Énergie Atomique. Les syntaxes graphiques, quant à elles, sont développées dans [FB05, Fon08]. Le projet “open source” *SVG-Based Modeling Tools* implémente la phase de réalisation.

3.7. Intégration des technologies basées modèles sur un outil existant (2009)

Ce travail a pour cadre mon poste de Maître de Conférence stagiaire, au sein du laboratoire MIPS de l’Université de Haute Alsace, Mulhouse (68).

Problématique. Test Designer est un logiciel édité par la société Smartesting S.A. Son but est de générer des tests fonctionnels à partir de modèles comportementaux exprimées dans un dialecte d’UML. Un problème est que ce dialecte n’est pas défini de façon formelle, et seule des documentations utilisateurs permettent de savoir quelles parties d’UML sont véritablement utilisées par l’outil. Ce problème est récurrent lorsqu’on réutilise un langage pour en construire un dialecte ou un nouveau langage, comme c’est le cas lorsqu’on établi un profil UML: le langage réutilisé ne l’est souvent pas dans son intégralité.

Résultats et réalisations. La première idée a été de transformer Test Designer en un véritable composant de traitement de modèle. En effet, s’il lit bien un modèle UML, l’outil utilise un format intermédiaire propriétaire rendant impossible l’intégration à un nouveau langage à une autre équipe que l’équipe de développement de Smartesting. Pour ce faire, nous avons défini un métamodèle inspiré de ce format intermédiaire, donc propre à Test Designer, exprimant toutes les données véritablement utilisées par l’outil. Nous avons également établi des transformations entre ce format intermédiaire et tout modèle basé sur ce métamodèle. L’intérêt immédiat pour résoudre le problème et la possibilité de travailler au niveau modèle. Notre travail suivant consiste en la définition exacte des parties d’UML qui sont véritablement utilisées par l’outil. Nous avons dégagés deux parties essentielles: les aspects structurels purs, et les aspects de cohérence de modèles. Les aspects structurels purs traitent du sous-ensemble d’UML au niveau de son métamodèle: quels sont les concepts du langage UML utilisés par Test Designer. Nous avons ainsi développé un métamodèle de sélection de concepts, puis des transformations extrayant un métamodèle dégagé des concepts inutiles à partir d’un métamodèle complet. De même, nous sommes capables de générer des transformation éliminant les instances de ces concepts inutiles dans des modèles: les modèles sont ainsi rendus compatibles avec le métamodèle diminué. Les aspects de cohérence sont plus destinés à permettre d’exprimes des règles où des concepts ne peuvent être instanciés que d’une certaine manière; les concepts en question sont toujours là, mais la sémantique statique est modifiée. Pour résoudre ce problème, nous pensons utiliser la reconnaissance de patrons: certains patrons sont positifs et permettent de conserver les instances les appliquant, ou au contraire négatifs, et expriment une incompatibilité dans le sous-ensemble du langage considéré.

Diffusion. Ce travail a pour cadre le projet de recherche VETESS, projet labellisé “Pôle véhicule du futur” en collaboration avec Smartesting S.A., Clemessy S.A., l’Université de Haute Alsace et l’Université de Franche-Comté. Les résultats de ce projet seront publiés comme délivrables. Les développements seront soit intégrés à Test Designer (métamodèle spécifique) ou publiés en Open Source.

3.8. Perspectives

Un projet de développement souhaitant utiliser la “méthodologie” IDM doit faire face à l’utilisation et la gestion de nombreux langages de modélisation. Chaque langage vient avec une spécification, mais aussi des outils permettant de les utiliser, que ce soit pour générer des tests, faire des simulations, générer du code ou encore d’autres modèles. Si les langages de modélisation bénéficient désormais d’un bon support pour leur définition, ils restent cependant difficiles à gérer: s’il est parfois nécessaire de les adapter à un problème donné, il est aussi nécessaire de pouvoir réutiliser aussi simplement que possible les outils qui leur sont associés.

Un exemple est Xion, décrit en section 3.3. Xion est en fait une réutilisation d’OCL auquel ont été ajoutées des constructions et une syntaxe Java. Malgré l’existence d’un compilateur pour OCL ciblant SQL, les travaux consistant à implémenter un compilateur pour les différentes versions du langage SQL que nous ciblions ont dû être menés de bout en bout. De même, il a été nécessaire d’écrire un éditeur de code Xion contenant des mécanismes d’aide à la production alors que de tels éditeurs existent déjà pour Java.

Je souhaite bien sûr améliorer les techniques de spécification de syntaxes concrètes, par exemple en rendant les syntaxes textuelles plus interactives, ou en permettant une approche analytique des syntaxes graphiques. Nous aurions alors une solution d’ingénierie des langages plus accessible. Cependant, il reste nécessaire de développer les techniques de gestion efficace des langages. Parmi les techniques nécessaires, on peut citer la composition des langages évitant le développement complet de langages par réutilisation, à l’instar de la programmation par composants [30].

Le but souhaité est de mettre en place un atelier de composition de langages et d’intégration d’outils, qu’ils soient ouverts (comme les outils “open source”) ou fermés (comme les outils propriétaires). Ces outils peuvent être de natures très variées: interpréteur, compilateurs, vérificateurs, générateurs de test, éditeurs graphiques ou textuels, transformations, etc. Un paramètre important est la complexité desdits outils, dont la reconstruction pour un langage donné n’est pas économiquement viable. Cependant, ces outils proposent tous des interfaces qui sont les métamodèles des langages manipulés, à l’instar de Test Designer auquel a été adjoint un métamodèle comme format de données d’entrée (voir section 3.7, page 16).

Les langages peuvent être composés de différentes manières: un langage peut soit s’appuyer sur, soit intégrer un ou plusieurs autres langages qui doivent alors être manipulés de manière cohérente. Le plus souvent, un langage `A`, plus abstrait et mieux adapté à un problème donné (comme un DSL [7]), s’appuie sur un langage `B`, plus concret et efficacement outillé (on peut parler de plate-forme), à travers un compilateur qui va transformer une spécification `A` en spécification `B`. Dans cette approche, un problème est l’abstraction des informations requises et fournies par les outils pour `B` dans un formalisme cohérent du point de vue de `A`. Par exemple, un langage de diagramme de classes UML peut souhaiter s’appuyer sur le langage C++ (il existe d’ailleurs de nombreux générateurs de code qui vont dans ce sens). Au cas où le compilateur C++ détecte un problème ou produit un avertissement dans le code généré, il sera nécessaire de remonter cette information au concepteur du diagramme UML de manière à ce qu’elle soit compréhensible d’un point de vue UML. On supposera que les informations en question seront structurées, par exemple par des métamodèles, qu’il s’agisse des informations produites par les outils, ou présentées en cohérence avec le modèle abstrait. Si les informations sont les mêmes, elle auront tout de même besoin de subir une restructuration par un adaptateur. On remarquera qu’un autre exemple pour `B` peut être une spécification en assembleur.

Dans l'approche par intégration, `A` et `B` font partie du même niveau d'abstraction. `A` peut laisser des "zones d'ombre" qui seront de la responsabilité d'un langage externe `B`. Par analogie aux composants, les outils manipulant des modèles exprimés en `A` fournissent le métamodèle A, et requièrent les zones d'ombre sous la forme de métamodèles requis arbitraires. Ce dernier métamodèle requis devra être mis en correspondance avec le métamodèle effectif du langage `B`, à nouveau à travers un adaptateur. Un exemple pour `A` est `OCL`, qui a besoin d'un modèle de classes et d'instances. Les différentes versions d'UML et de MOF, mais aussi les bases de données relationnelles ou les documents XML peuvent jouer ce rôle et sont donc des possibilités concrètes pour `B`. Une fois de plus, il sera nécessaire de faire travailler ensemble les outils développés pour `A` (comme un interpréteur OCL) et pour `B` (comme un RDBMS) de manière cohérente. Des adaptateurs seront alors nécessaires, adaptateurs qui pourront être sélectionnés par des mécanismes de modularité tels que ceux introduits dans MTL: les "zones d'ombre" étant raffinées par les adaptateurs réagissant en tant que proxies sur les modèles réels.

Les compilateurs, et plus récemment les transformations de modèles, sont les réponses courantes au souci d'adapter des structures d'informations. Cependant, il arrive fréquemment que plusieurs plates-formes (et donc langages) soient nécessaires pour réaliser toutes les promesses des spécifications abstraites. Il est donc nécessaire que les outils prévus pour les langages concrets puissent travailler de concert sur des spécifications communes, mais vues à travers les métamodèles des langages pour lesquels ils sont prévus. En d'autres termes, une solution au problème d'intégration serait le retypage de modèles aux besoins des différents outils les utilisant de manière concurrente. Ceci nécessite des transformations qui doivent être rendues bidirectionnelles, incrémentales, concurrentes, et particulièrement performantes. Or, il existe très peu de tels transformateurs de modèles, de par la complexité de leur réalisation. De plus, ces transformations multiplieraient le nombre de modèles nécessaires (une copie par métamodèle). Une alternative serait la création de différentes vues sur un seul et même modèle à travers le prisme de différents métamodèles, vues pouvant être manipulées par un outil exactement comme un modèle réel.

Un exemple que je propose d'étudier est l'intégration de divers outils de vérification ou de simulation des systèmes de commande/contrôle. En effet, plusieurs paradigmes, supportés par plusieurs langages, peuvent entrer en jeu lors de la conception de tels systèmes: systèmes à événements discrets synchrones ou asynchrones, systèmes continus décrits par des formules mathématiques, etc. La description de tels systèmes hétérogènes peut être rendue ardue par les relations qui peuvent exister entre ces différentes descriptions, relations trop souvent informelles et limitant l'utilisabilité et la pertinence des outils associés à propos système complet.

Les adaptateurs de modèles sont donc des éléments qui peuvent jouer un rôle important dans la réutilisation de langage, au même titre que dans la réutilisation de composants, par "simple" retypage d'informations. Ils peuvent jouer également un rôle important dans l'abstraction de l'information. Une description abstraite en même temps qu'une réalisation efficace de ces adaptateurs est donc souhaitable. C'est ainsi que je me propose de déterminer un langage pour ces adaptateurs, de les réaliser de manière efficace, et de les mettre en jeu dans une réalisation modulaire de spécifications inter-langages, qu'ils s'agisse de spécifications compilées (avec abstraction d'informations) ou interprétées (avec collaboration). C'est cette proposition qui est à l'origine du projet *Blanddern*.

Cependant, cet axe que je propose n'est une première étape dans la composabilité agile des langages de modélisation. En effet, il n'est question ici que de réutiliser les outils développés autour de ces langages à partir d'une adaptation de la syntaxes abstraite de ces langages. Un axe de recherche à plus long terme viserait à atteindre des objectifs de véritable composabilité de la description des langages eux-mêmes, non seulement au niveau de leur syntaxe abstraite, mais aussi concernant leur syntaxe concrète et leur sémantique.

3.9. Publications

Je présente ici la liste des publications dont je suis co-auteur, tout en donnant, si possible, la sélectivité des comités de lecture ou programme. J'indique également une évaluation de ma participation à l'effort de rédaction, et le nombre de citations selon le site Google Scholar[1] au 15 Mai 2009: cette liste de publications inclut 7 papiers cités au moins 7 fois (soit un index H de 7)[2].

3.9.1. Chapitres de livres

[BMFS06] Thomas Baar, Slaviša Marković, **Frédéric Fondement**, and Alfred Strohmeier, Definition and correct refinement of operation specifications., Research Results of the DICS Program (Jürg Kohlas, Bertrand Meyer, and André Schiper, eds.), Lecture Notes in Computer Science, vol. 4028, Springer, 2006, pp. 127–144. *(cité 2 fois; participation de 10%)*

[SFS04b] Raul Silaghi, **Frédéric Fondement**, and Alfred Strohmeier, "Weaving" MTL model transformations., Model Driven Architecture (Uwe Aßmann, Mehmet Aksit, and Arend Rensink, eds.), Lecture Notes in Computer Science, vol. 3599, Springer, 2004, pp. 123–138.
Ce livre regroupe, suivant l'avis d'un comité de lecture dédié, les améliorations des meilleurs papiers des ateliers internationaux Model Driven Architecture: Foundations and Applications - MDAFA des années 2003 et 2004. Cet article est en effet l'amélioration d'un article soumis et présenté les 10 et 11 juin 2004, Linköping, Suède. *(cité 6 fois; participation de 50%)*

3.9.2. Journaux internationaux

[MFF+08] Pierre-Alain Muller, **Frédéric Fondement**, Franck Fleurey, Michel Hassenforder, Rémi Schnekenburger, Sébastien Gérard, and Jean-Marc Jézéquel, Model-driven analysis and synthesis of textual concrete syntax., Software and System Modeling (SoSyM), 2008, (DOI: 10.1007/s10270-008-0088-x - to appear). *(cité 3 fois; participation de 30%)*

[MSFB05] Pierre-Alain Muller, Philippe Studer, **Frédéric Fondement**, and Jean Bézivin, Platform independent web application modeling and development with Netsilon., Software and System Modeling (SoSyM) 4 (2005), no. 4, pp. 424–442. *(cité 35 fois; participation de 30%)*

3.9.3. Conférences internationales avec comité de lecture et publication des actes

[MFB09] Pierre-Alain Muller, Frédéric Fondement, Benoît Baudry, Modeling modeling, 12th International Conference on Model-Driven Engineering Languages and Systems - MoDELS, Denver, USA, October 4-8, 2009. *(en cours de soumission)*

1. http://scholar.google.fr/
2. Jorge E. Hirsch, An index to quantify an individual's scientific research output., Proceedings of the National Academy of Sciences, vol. 102, no. 46, 2005, pp. 16569-16572.

[Fon08] **Frédéric Fondement**, Graphical concrete syntax rendering with SVG., Fourth European Conference on Model Driven Architecture Foundations and Applications - ECMDA-FA (Philippe Desfray, Alan Hartman, Richard Paige, Arend Rensink, Andy Schürr, Regis Vogel, Jos Warmer, eds.), Berlin, Germany, June 9-12, 2008, Lecture Notes in Computer Science, Springer, 2008. *(sélectivité de 26%; participation de 100%)*

[FB05] **Frédéric Fondement** and Thomas Baar, Making metamodels aware of concrete syntax., First European Conference on Model Driven Architecture Foundations and Applications - ECMDA-FA (Alan Hartman and David Kreische, eds.), Nuernberg, Germany, November 7-10, 2005, Lecture Notes in Computer Science, vol. 3748, Springer, 2005, pp. 190–204. *(sélectivité de 29%; cité 33 fois; participation de 60%)*

[MFF+06] Pierre-Alain Muller, Franck Fleurey, **Frédéric Fondement**, Michel Hassenforder, Rémi Schnekenburger, Sébastien Gérard, and Jean-Marc Jézéquel, Model-driven analysis and synthesis of concrete syntax., 9th International Conference on Model-Driven Engineering Languages and Systems - MoDELS (Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, eds.), Genoa, Italy, October 1-6, 2006, Lecture Notes in Computer Science, vol. 4199, Springer, 2006, pp. 98–110. *(sélectivité de 29%; cité 9 fois; participation de 10%)*

[SFS04a] Raul Silaghi, **Frédéric Fondement**, and Alfred Strohmeier, Towards an MDA-oriented UML profile for distribution., 8th International IEEE Enterprise Distributed Object Computing Conference - EDOC, Monterey, California, September 20-24 2004, IEEE Computer Society, 2004, pp. 227–239. *(sélectivité de 36%; cité 20 fois; participation de 40%)*

3.9.4. Ateliers internationaux avec comité de lecture

[MFV+05] Pierre-Alain Muller, Franck Fleurey, Didier Vojtisek, Zoé Drey, Damien Pollet, **Frédéric Fondement**, Philippe Studer, and Jean-Marc Jézéquel, On executable meta-languages applied to model transformations., Model Transformations In Practice Workshop, satellite event of the MoDELS 2005 Conference, Montego Bay, Jamaica, October 3rd, 2005. *(cité 43 fois; participation de 10%)*

[FS04] **Frédéric Fondement** and Raul Silaghi, Defining Model Driven Engineering Processes., 3rd International Workshop in Software Model Engineering (WiSME@UML), satellite event of the UML 2004 Conference, Lisbon, Portugal, October 11, 2004. *(cité 17 fois; participation de 70%)*

[MDFH04] Pierre-Alain Muller, Cédric Dumoulin, **Frédéric Fondement**, and Michel Hassenforder, The topmodl initiative., 3rd International Workshop in Software Model Engineering (WiSME@UML), satellite event of the UML 2004 Conference, Lisbon, Portugal, October 11, 2004., further published in UML Satellite Activities (Nuno Jardim Nunes, Bran Selic, Alberto Rodrigues da Silva, and José Ambrosio Toval Álvarez, eds.), Lecture Notes in Computer Science, vol. 3297, Springer, 2004, pp. 242–245. *(cité 3 fois; participation de 20%)*

[ZJF03] Tewfik Ziadi, Jean Marc Jézéquel, and **Frédéric Fondement**, Product Line Derivation with UML., Software Variability Management Workshop, Groningen, The Netherlands, February 13-14, 2003, pp. 94–102. *(cité 19 fois; participation de 30%)*

3.9.5. Rapport technique

[FSGM06] **Frédéric Fondement**, Rémi Schnekenburger, Sébastien Gérard, and Pierre-Alain Muller, Metamodel-Aware Textual Concrete Syntax Specification, Tech. report LGL-REPORT-2006-005, École Polytechnique Fédérale de Lausanne (EPFL), 2006. *(cité 5 fois; participation de 70%)*

3.9.6. Mémoires

[Fon07] **Frédéric Fondement**, Concrete syntax definition for modeling languages., PhD. Thesis 3927, École Polytechnique Fédérale de Lausanne (EPFL), 2007. *(cité 3 fois)*

[Fon01] **Frédéric Fondement**, Création d'un Langage d'Action UML pour un Logiciel MDA, Mémoire de stage de DRT, Université de Haute Alsace en partenariat avec ObjeXion Software S.A., 2001.

[Fon00] **Frédéric Fondement**, Développement de Composants pour une Machine Virtuelle UML., Mémoire de stage de diplôme d'ingénieur, Ecole Supérieure des Sciences Appliquées pour l'Ingénieur de Mulhouse (ESSAIM) en partenariat avec ObjeXion Software S.A., 2000.

[Fon97] **Frédéric Fondement**, Développement de Gestionnaires de PARTS sous Navigator., Mémoire de stage de DUT GEII, Institut Universitaire de Technologie de Belfort-Montbéliard en partenariat avec Peugeot S.A., 1997.

3.9.7. Documentation

Guide de l'utilisateur et documentation du langage Xion pour le logiciel *Netsilon* (accessible dans l'aide du logiciel et en ligne à l'URL http://fondement.free.fr/objx/netsilon/).

3.10. Encadrement

Je souhaite également indiquer ici que j'ai eu le plaisir d'encadrer deux projets de semestre en tant qu'assistant d'enseignement et de recherche à l'EPFL, ainsi que quatre en tant qu'ATER puis maître de conférences à l'ENSISA. Plus de précisions sont apportées en section 4.2, page 31 sur l'aspect pédagogique. Ces projets de semestre se sont inscrits dans le cadre des projets de développement logiciel *SVG-Based Modeling Tools* et *Blanddern*. Je suis l'auteur des sujets ainsi que l'encadrant unique de ces projets.

J'ai également eu l'opportunité d'encadrer un assistant étudiant à plein temps pour une durée de deux mois. Amit Raj, étudiant en cycle Master à l'Indian Institute of Technology (IIT) de Kanpur, a pu assurer cet emploi. Le travail portait sur l'implémentation d'un parseur OCL dont l'arbre de syntaxe abstraite était structuré par une bibliothèque MTL. Cette bibliothèque peut être adaptée a posteriori à un langage concret par raffinement. Ce travail a montré la possibilité de décrire un langage (ici OCL) étendant un autre langage a priori inconnu mais dont certains aspects structurels sont représentés par un modèle abstrait. Ce travail a également montré qu'il est possible d'adapter par raffinement cette description abstraite à un langage concret (par exemple UML ou une base de données relationnelle), cependant au prix de développements coûteux et répétitifs. C'est sur la base de ces conclusions que je propose d'améliorer la description de tels adaptateurs (voir section 3.8, page 17).

Le projet de recherche VETESS, projet labellisé “Pôle véhicule du futur” est financé à hauteur de 209k€. Il s’agit d’une collaboration entre Smartesting S.A. (porteur), Clemessy S.A, Peugeot S.A., l’Université de Haute-Alsace, et l’Université de Franche-Comté. Au sein de ce projet commencé en Septembre 2009 et d’une durée de 24 mois, j’ai la responsabilité scientifique du travail devant être réalisé par l’Université de Haute-Alsace, ainsi que la responsabilité de l’encadrement d’un ingénieur de recherche à plein temps. La section 3.7, page 16 apporte plus de détails sur le sujet traité.

Le projet de recherche IMMSI, financé par la région Alsace, est un projet pluridisciplinaire visant à simplifier la réalisation de chaînes de traitement d’images, telles qu celle qu’on peut trouver dans la microscopie optique ou électronique. Dans ce cadre, je co-encadre la thèse de doctorat de Charles-Georges Guillemot débutée en Octobre 2008. Nous nous intéressons plus particulièrement dans ce projet à la définition interactive des chaînes de traitement. L’idée est de demander au concepteur d’une chaîne de traitement les raisons qui motivent les éléments de la chaîne afin de nourrir une base de connaissances qui peut resservir plus tard à aider à construire de nouvelle chaîne. Il s’agit là encore d’une application pratique des technologies basées modèle dans les domaines des workflows (pour la définition des chaînes de traitement) et des ontologies (pour la base de connaissances).

3.11. Activités d’évaluation

Parallèlement à ces activités de recherche, j’ai eu l’occasion de participer à l’évaluation d’articles scientifiques dans le domaine de l’IDM.

J’ai été membre du comité de programme pour l’atelier national *Ingénierie Dirigée par les Modèles* (IDM) pour les années 2008[1], 2005 et 2006.

Par ailleurs, j’ai fréquemment l’occasion de jouer un rôle de relecteur. Ce fut le cas pour le cycle de conférences internationales *MoDELS* (anciennement nommé UML), pour les sessions 2009, 2008, 2005, 2003, 2001 et 2000. Je fus également relecteur pour 3 numéros de la revue internationale *Software and Systems Modeling* (SoSyM)[2]. Plus occasionnellement, je fus relecteur pour les revues internationales Science of Computer Programming (Elsevier - Vol. 53, n. 3) sur les lignes de produits logicielles, IET Software (IET), Journal of Systems and Software (Elsevier), et World Wide Web Journal (Springer), pour le livre Model-Driven Software Development[3], et pour la conférence internationale Visual Languages and Human-Centric Computing (VL-HCC) de 2005.

3.12. Références

Je produis ici la liste des publications sur lesquelles s'appuie le présent chapitre, mais dont je ne suis pas auteur.

[1] J.-R. Abrial, The B-book: assigning programs to meanings. New York, NY, USA: Cambridge University Press, 1996.

1. http://planetmde.org/idm08/
2. http://www.sosym.org/
3. Sami Beydeda, Matthias Book, Volker Gruhn. Model-Driven Software Development. ISBN 3-540-25613-X, Springer, 2005

[2] K. Beck and C. Andres, Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional, 2004.

[3] L. J. Osterweil, “Software processes are software too,” in ICSE, pp. 2–13, 1987.

[4] S. Kent, “Model Driven Engineering,” in IFM (M. J. Butler, L. Petre, and K. Sere, eds.), vol. 2335 of Lecture Notes in Computer Science, pp. 286–298, Springer, 2002.

[5] C. Atkinson and T. Kühne, “The role of meta-modeling in MDA,” in Workshop in Software Model Engineering (WISME@UML), (Dresden, Germany), October 2002.

[6] S. Sendall and W. Kozaczynski, “Model Transformation: The Heart and Soul of Model-Driven Software Development,” IEEE Software, vol. 20, no. 5, pp. 42–45, 2003.

[7] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. Oliva, T. Sheard, I. Smith, and L. Walton, “A Software Engineering Experiment in Software Component Generation,” in ICSE, pp. 542–552, 1996.

[8] Adaptive Ltd., Alcatel, Borland Software Corporation, Computer Associates International, Inc., Telefonaktiebolaget LM Ericsson, Fujitsu, Hewlett-Packard Company, I-Logix Inc., International Business Machines Corporation, IONA Technologies, Kabira Technologies, Inc., MEGA International, Motorola, Inc., Object Management Group., Oracle Corporation, SOFTEAM, Telelogic AB, Unisys, and X-Change Technologies Group, LLC, “Unified Modeling Language (UML), version 2.1.1.” OMG Document formal/07-02-05, February 2007.

[9] E. Marcos, B. Vela, and J. M. Cavero, “A Methodological Approach for Object-Relational Database Design using UML,” Software and System Modeling, vol. 2, no. 1, pp. 59–75, 2003.

[10] Adaptive Ltd., Boldsoft, France Telecom, International Business Machines Corporation, IONA Technologies, and Object Management Group, “Object Constraint Language specification, v2.0.” OMG Document formal/06-05-01, May 2006.

[11] J. Mukerji and J. Miller, “MDA guide, v1.0.1.” OMG Document omg/03-06-01, June 2003.

[12] Adaptive, Ceira Technologies, Inc., Compuware Corporation, Data Access Technologies, Inc., DSTC, Gentleware, Hewlett-Packard, International Business Machines, IONA, Object Management Group, MetaMatrix, Softeam, SUN, Telelogic AB, and Unisys, “Meta-Object Facility (MOF) core, v2.0.” OMG Document formal/2006-01-01, January 2006.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin, “Aspect-Oriented Programming,” in ECOOP, pp. 220–242, 1997.

[14] D. Vojtisek and J.-M. Jézéquel, “MTL and umlaut NG - engine and framework for model transformation,” ERCIM News, vol. 58, pp. 46–47, July 2004.

[15] F. Fleurey, Langage et méthode pour une ingénierie des modèles fiable. PhD thesis, Université de Rennes 1, October 2006.

[16] E. Jouenne and V. Normand, “Tailoring IEEE 1471 for MDE support,” in UML Satellite Activities (N. J. Nunes, B. Selic, A. R. da Silva, and J. A. T. Álvarez, eds.), vol. 3297 of Lecture Notes in Computer Science, pp. 163–174, Springer, 2004.

[17] S. Sendall and A. Strohmeier, “UML Based Fusion Analysis Applied to a Bank Case Study,” in UML (R. B. France and B. Rumpe, eds.), vol. 1723 of Lecture Notes in Computer Science, pp. 278–291, Springer, 1999.

[18] P. Clements, L. Northrop, and L. M. Northrop, Software Product Lines : Practices and Patterns. Addison-Wesley Professional, August 2001.

[19] R. Pohjonen, “Boosting Embedded Systems Development with Domain-Specific Modeling,” RTC Magazine, pp. 57–61, April 2003.

[20] D. Harel and B. Rumpe, “Meaningful Modeling: What’s the Semantics of "Semantics"?,” Computer, vol. 37, no. 10, pp. 64–72, 2004.

[21] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel, “Weaving Executability into Object-Oriented Meta-languages,” in International Conference on Model Driven Engineering Languages and Systems (MoDELS), vol. 3713 of LNCS, pp. 264–278, Springer, October 2005.

[22] S. Markovic and T. Baar, “An OCL Semantics Specified with QVT,” in MoDELS (O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, eds.), vol. 4199 of Lecture Notes in Computer Science, pp. 661–675, Springer, 2006.

[23] International Organization for Standardization, “Information Technology - Syntactic Metalanguage - Extended BNF.” ISO/IEC 14977, August 2001.

[24] H. Göttler, “Attributed graph grammars for graphics,” in Graph-Grammars and Their Application to Computer Science (H. Ehrig, M. Nagl, and G. Rozenberg, eds.), vol. 153 of Lecture Notes in Computer Science, pp. 130–142, Springer, 1982.

[25] T. J. Parr and R. W. Quong, “ANTLR: A predicated-LL(k) parser generator,” Software - Practice and Experience, vol. 25, no. 7, pp. 789–810, 1995.

[26] Eclipse, “Java Emitter Templates (JET),” 2005.

[27] D. Jackson and C. Northway, “Scalable Vector Graphics (SVG) Full 1.2 specification.” World Wide Web Consortium, Working Draft WD-SVG12-20050413, April 2005.

[28] C. L. McCormack, K. Marriott, and B. Meyer, “Constraint SVG,” in WWW Alt. ’04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, (New York, NY, USA), pp. 310–311, ACM Press, 2004.

[29] O. Beaudoux, “DoPIdom: une approche de l’interaction et de la collaboration centrée sur les documents,” in IHM ’06: Proceedings of the 18th international conference on Association Francophone d’Interaction Homme-Machine, (New York, NY, USA), pp. 19–26, ACM Press, 2006.

[30] C. Szyperski, Component Software: Beyond Object-Oriented Programming. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

3.13. Projets logiciels

Je présente ici succinctement les projets de développement logiciel auxquels j'ai eu l'occasion de participer, et dont il est fait mention dans les sections précédentes. Ces divers projets ont tous été le support de recherches. Lorsque ces recherches ont fait l'objet d'un article dont je suis co-auteur, cet article est référencé. Si le projet a été le support de recherches relatives à une publication, cet article est référencé entre parenthèses. Pour chaque projet, je détaille la thématique, produit une page internet de référence (si disponible), et indique le rôle que j'ai tenu.

Blanddern UHA Architecte et auteur (ATER puis MdC) 2008-2009 4 développeurs	La connection de deux composants nécessite impérativement une conformité de leurs contrats. Lorsque ça n'est pas le cas, il est parfois possible d'écrire un adaptateur traduisant les requêtes d'un composant en requêtes compréhensibles par l'autre composant. Toutefois, il s'agit là d'une tâche souvent complexe sans pour autant être à forte valeur ajoutée. Blanddern permet de définir de tels adaptateurs simplement lorsque les composants communiquent au moyen de modèles. Des exemples de tels composants sont les transformations de modèles, ou les outils CASE. http://blanddern.berlios.de/ Je suis ici l'auteur de l'idée originale. J'ai choisi l'architecture du logiciel. J'ai également coordonné les développements, qui avaient pour contexte des projets de semestre ENSISA (voir section 4.5, page 33).
Sintaks et TCSSL Tools INRIA, UHA et CEA Co-auteur de l'idée originale (doctorant, puis ATER) 2005-2007 [MFF+06]	Ces deux projets traitent de la modélisation de syntaxes textuelles pour les langages de modélisation dont la syntaxe abstraite est décrite sous la forme d'un métamodèle prédéfini. A partir d'une telle description, il doit être possible de fournir des outils capables de créer en dépositaire un modèle à partir d'une représentation textuelle, et de générer la représentation textuelle d'un modèle en dépositaire. *(voir section 3.6, page 14)* http://www.kermeta.org/sintaks/ Je suis ici le co-auteur de l'idée originale. Si, contrairement aux autres projets, je n'ai pas participé aux développements, j'ai pu organiser et/ou participer aux réunions de travail.

SVG-Based Modeling Tools EPFL architecte et auteur principal (doctorant) 2004-2008 4 développeurs [Fon08] [FB05]	Le but du projet est de fournir un outil permettant de réaliser la syntaxe concrète d'un langage de modélisation graphique. Pour chaque méta-élément à représenter, il permet de définir des patrons décrits en langage SVG. Ces patrons sont complétés par des composants DOM prédéfinis indiquant quels sont les possibilités de manipulation graphique, ainsi que leurs événements (par exemple le déplacement, l'accrochage de liens, ou l'intégration d'un contenu). Des instructions Java-JMI au sein du patron permettent de décrire le comportement face à ces événements, notamment les effets sur le modèle représenté qui se trouve stocké en dépositaire. De plus, des instructions de contraintes C-SVG permettent de spécifier les relations entre ces éléments graphiques (par exemple un carré qui grandit pour contenir un texte). L'outil développé permet d'instancier les patrons SVG tout en autorisant les comportements indiqués par les composants DOM, en respectant les contraintes C-SVG et en interprétant les instructions Java-JMI correspondant aux événements. *(voir section 3.6, page 14)* http://fondement.free.fr/lgl/projects/probxs/ Je suis ici l'auteur de l'idée originale. J'ai choisi l'architecture du logiciel et me suis occupé d'intégrer les outils choisis. J'ai également participé et coordonné les développements, qui avaient pour contexte des projets de semestre EPFL.
MTL INRIA co-auteur (ingénieur expert) 2003 3 développeurs [MFV+05] ([SFS04b])	MTL est un langage de transformation de modèles. Il s'agit d'un langage impératif orienté objet, incluant des considérations comme l'héritage multiple, les associations, ou le raffinement via les librairies qui offrent un mécanisme d'héritage d'ordre supérieur. Il est capable de manipuler un ensemble de modèles disponibles à travers un certain nombre de dépositaires de types divers (MDR, Poseidon, EMF, …). Un modèle peut également être décrit par des librairies MTL et ainsi être stocké par des objets MTL. Cette dernière caractéristique a permis d'appliquer une approche dite de "bootstrap", c'est à dire construit sur un noyau de langage et étendu par des transformations en MTL. MTL est supporté par un compilateur vers le langage Java. *(voir section 3.4, page 12)* http://gforge.inria.fr/projects/mtl/ Le définition du langage et de l'architecture de sa réalisation fut une tâche collective. Je me suis également concentré sur la définition de l'interface avec les modèles à transformer de manière à rester indépendant du dépositaire de modèles et du langage de métamodélisation.

Fondue Builder EPFL collaborateur, conseiller, webmestre (doctorant) 2004 4 développeurs ([BMFS06])	La méthode Fondue allie la notation UML à la méthode Fusion. Cependant, certaines différences existent par rapport à la notation UML standard. Fondue Builder est un modeleur dédié à la phase d'analyse de la méthode Fusion. Pour les aspects graphiques, nous avons modifié l'outil Together de Borland par l'intermédiaire de scripts décrivant les composants graphiques utilisés par la méthode. Un visiteur d'arbre permet de transformer ces éléments graphiques en un modèle stocké dans un dépositaire, conforme au métamodèle de Fondue tel que nous l'avons défini. Un cadre (framework) de transformation de modèles (en MTL) permet de vérifier les contraintes de cohérence du modèle. Le modèle ainsi créé et vérifié peut permettre de multiples utilisations comme l'animation des modèles ou la génération de tests. *(voir section 3.5, page 13)* Je me suis occupé de l'export du modèle Fondue vers un dépositaire de modèles formatés par le métamodèle que j'avais conçu. Une seconde étape fut la vérification des modèles ainsi stockés par une transformation de modèles (grâce au projet *MTL*). Dans le contexte de ce projet, j'ai eu l'occasion de former les participants à la technique des dépositaires de modèles, de mettre en place le site internet (grâce au projet *Netsilon*), et d'adapter un prototypeur de modèles.
Parallax conseiller (doctorant) 2004 4 développeurs [SFS04a] [SFS04b]	Les générateurs de code ont cette caractéristique de ne s'appuyer que sur une seule sorte de modèle pour générer du code pour une seule sorte de plate-forme. Si l'on change de type de modèle ou de plate-forme, il est nécessaire de redéfinir entièrement la génération, même s'il ne s'agit de ne prendre en compte qu'une simple extension du modèle, par exemple un profil. Parallax est un environnement de génération de code basé sur un double mécanisme d'extension. Une extension définit une génération de code par un programme lisant un modèle derrière un dépositaire et générant un arbre abstrait du langage cible (par exemple Java). L'originalité de Parallax est de pouvoir ajouter des extensions à ces extensions en s'appuyant sur les aspects: si le modèle d'entrée doit être adapté à des préoccupations orthogonales, ces extensions vont définir des aspects faisant prendre en compte ces adaptations par le générateur de code. *(voir section 3.5, page 13)* J'ai agi pour ce projet en qualité de conseiller relatif aux techniques de métamodélisation. Je me suis d'abord occupé, avec le Dr. Silaghi, de définir une architecture de profils UML, et de son application concrète à la distribution d'un système. Nous nous sommes par la suite occupés de définir un langage d'aspect pour *MTL*, dont j'ai réalisé le tisseur.

Netsilon
ObjeXion S.A.

collaborateur (stage DRT puis ingénieur logiciel)

2001 et 2002

5 développeurs

[MSFB05] [MFV+05]

Netsilon est un projet commercial. C'est un modeleur d'application internet complet capable de générer des applications internet de type JSP, Servlet ou PHP s'appuyant sur une base de données MySQL, Oracle ou PostgreSQL. La modélisation se fait en trois étapes.

1. Le *modèle métier* est un modèle de classe UML et permet de définir la structure de la base de données. Pour la manipulation et l'interrogation des données, nous y avons adjoint un langage d'action spécifique, Xion, qui se situe au niveau modèle plutôt qu'au niveau relationnel. Lors de la génération, ce langage est donc traduit dans le dialecte SQL de la base de données choisie.
2. Le *modèle de navigation* est un modèle spécifique qui permet de décrire la cinématique des différentes pages d'une application internet, en fonction des données auxquelles on a accès grâce à Xion.
3. L'aspect graphique des pages est donné par des patrons dans le langage géré par l'application internet, généralement HTML. Cependant des expériences concluantes ont permis de générer des macros QuarkXPress ou du SVG.

En outre, Netsilon intègre un outil permettant de générer un administrateur complet de données objet à partir du modèle métier, opérant concrètement sur la base de données. *(voir section 3.3, page 10)*

http://fondement.free.fr/objx/Netsilon

Mes contributions à ce projet sont multiples. Mon activité la plus importante a été de définir le langage d'action Xion, capable d'interroger et de manipuler une instance du modèle métier stockée en base de données. J'ai réalisé les premières couches du compilateur (analyses lexicales, syntaxiques et sémantiques, production de l'arbre abstrait), et un éditeur de code intelligent. Une autre tâche importante fut de transformer partiellement les expressions Xion analysées en expressions SQL spécifiques à la base de données utilisée. Je me suis également occupé de la génération de l'administrateur d'objets métier, de l'interface de développement du modèle de navigation, et de la décomposition des projets en composants réutilisables. J'ai également rédigé le guide de l'utilisateur, dispensé des formations aux clients, et développé des applications internet commerciales avec l'outil.

Les principes et la réalisation du projet sont mieux décrits dans l'article [MSFB05].

FacSimile
ObjeXion S.A.

collaborateur
(stage ingénieur)

2000

3 développeurs

FacSimile est un projet commercial permettant de prototyper des modèles de classes UML avec support OCL. Un modèle de classes UML est développé sous Rose ou donné sous forme XMI, ce qui permet de configurer une base de données relationnelle. FacSimile est un outil qui permet d'instancier des objets et des liens (c'est à dire un prototype) tout en s'appuyant sur cette base de données. Un interpréteur OCL permet d'interroger le modèle objet, ou de trouver les défauts du prototype. *(voir section 3.2, page 10)*

http://fondement.free.fr/objx/modelprototyper

J'ai réalisé la partie OCL constituée d'un processeur de texte, d'un contrôleur des types, et d'un interpréteur. Cet interpréteur est capable d'interroger le prototype que ce soit dans le but de trouver une donnée, vérifier des contraintes, vérifier les instances réalisant une propriété, ou définir le corps des méthodes sans effet de bord.

4. Activités d'enseignement

L'École Polytechnique Fédérale de Lausanne (EPFL) forme principalement des ingénieurs dans diverses disciplines (physique, microtechniques, matériaux, architecture, etc.). En tant qu'assistant d'enseignement et de recherche, j'ai pu assurer des enseignements au sein de la faculté informatique et communication à des étudiants en dernière année de préparation de Bachelor (équivalent à la 3ème année de Licence en France). A ce titre, j'ai pu participer aux suivis du projet de génie logiciel (section 4.1), et ai été responsable de deux projets de semestre (section 4.2).

L'École Nationale Supérieur d'Ingénieurs Sud Alsace (ENSISA), basée sur le campus de l'Illberg de Mulhouse, assure la formation d'ingénieurs dans quatre disciplines (mécanique et systèmes, systèmes et signaux, textile et fibre, informatique et communication). En tant qu'ATER ou que maître de conférences, j'ai eu la responsabilité des travaux pratiques d'ingénierie des métamodèles (IMM), d'ingénierie dirigée par les modèles (IDM) et de génie logiciel pour des étudiants de 2ème année de la filière informatique et communication (soit en 1ère année de Master dans le système LMD - section 4.3). J'ai également eu la responsable de quatre projets de semestre. De plus, j'ai invité et organisé une étape de la conférence métier Agile Tour 2008

Année	Lieu	Niveau	Cours	CM	TD	TP	Éq. TD
2003-2004	EPFL	L3	Projet GL	2:30		16:00	14:25
2003-2004	EPFL	L3	Cours GL		2:00		2:00
2004-2005	EPFL	L3	Projet GL	2:00		8:00	8:20
2004-2005	EPFL	L3	Projet de semestre			28:00	18:40
2005-2006	EPFL	L3	Projet de semestre			28:00	18:40
2007-2008	ENSISA	M1	IMM			60:00	40:00
2007-2008	ENSISA	M1	IDM			60:00	40:00
2007-2008	ENSISA	M1	Projet de semestre			24:00	16:00
2008-2009	ENSISA	M1	IDM	10:00	24:00	48:00	71:00
2008-2009	ENSISA	M1	Génie Logiciel	20:00	26:00	36:00	80:00
2008-2009	ENSISA	M1	Projet de semestre			48:00	32:00
2008-9	ENSISA	M2	Projet			24:00	16:00
Total équivalent TD:							**357:05**

4.1. Projet de Génie Logiciel (EPFL - 2004-2005)

Le projet de génie logiciel concerne des étudiants en section d'informatique de niveau L3. Il peut leur apporter un maximum de 10 crédits, sur un total de 60 à acquérir sur l'année. Le projet s'étale sur toute l'année universitaire.

Ce projet prépare le futur ingénieur informaticien à développer du logiciel en équipe en appliquant la méthode de développement logiciel Fusion alliée à la notation UML. L'enseignement est articulé autour de la réalisation d'un projet de groupe qui nécessite un important effort hors des heures attribuées à l'horaire. Les étudiants y développent une application logicielle de complexité moyenne en suivant une démarche rigoureuse qui doit les éveiller aux problèmes, principes, méthodes, et techniques du génie logiciel. Durant les heures consacrées au travail de groupe sur le projet, l'assistant responsable de la phase courante rencontre les groupes selon un horaire établi afin de les conseiller et de répondre à leurs questions. En dehors de ces rencontres périodiques, les groupes se réunissent et travaillent comme ils l'entendent. Des périodes supplémentaires de disponibilité sont également communiquées pour chaque phase du projet par l'assistant concerné. Chaque groupe développe un produit en accomplissant un travail de développement par phases. A l'issue de la plupart des phases, un document est à rendre et un des étudiants du groupe expose le résultat du travail accompli durant l'étape achevée. Les différentes phases sont la proposition d'un projet (2 semaines), la description du manuel de l'utilisateur (4 semaines), l'analyse (5 semaines), le développement (5 semaines), et l'implémentation (6 semaines). Le logiciel développé fait l'objet d'une démonstration et d'un test d'acceptance à la fin du projet.

Je fus l'assistant co-responsable à 50% de la phase d'analyse pour les années universitaires 2004-2005 et 2005-2006, qui ont concerné un total de 70 étudiants. Ceci a impliqué le suivi du travail, la participation à l'évaluation des présentations des différentes phases (soit un total de 24 heures), ainsi que la correction des rapports. De plus, j'ai été responsable de cours ex-cathedra sur l'utilisation d'un outil de développement pour le langage de programmation Java, et sur l'utilisation de la bibliothèque de composants d'interfaces hommes/machines Swing (pour un total de 4:30).

4.2. Encadrement de projets de semestre (EPFL - 2005-2006)

Dans le cadre de l'obtention de leur Bachelor EPFL (appelé Licence en France), les étudiants ingénieurs de l'EPFL doivent accomplir des projets de semestre. La faculté évalue à 312 heures, réparties sur 14 semaines, la charge de travail pour chaque étudiant afin de mener à bien son projet. Il peut leur apporter jusqu'à 12 crédits sur un total de 60 à acquérir sur l'année. Les sujets sont proposés par les enseignants de la faculté.

J'ai eu l'occasion de superviser à 100% deux projets de semestre dont j'avais proposé les sujets. Le premier projet s'est déroulé au semestre d'été 2005 et a concerné un étudiant. Le second projet s'est déroulé au semestre d'hiver 2005-2006 et a concerné deux étudiants. J'avais organisé le suivi par des rencontres hebdomadaires d'au moins deux heures. De plus, les étudiants ont eu à exposer leur projet et travaux par trois présentations aux membres du laboratoire (présentation du sujet, présentation intermédiaire et présentation finale).

4.3. Ingénierie Dirigée par les Modèles (ENSISA - 2008-2009)

Je suis responsable du cours Ingénierie Dirigée par les Modèles depuis l'année universitaire 2008-2009 que j'ai du monter dans son intégralité. Ce cours a concerné cette année 34 étudiants de la section informatique et réseaux de l'ENSISA au niveau M1 pour un volume horaire total de 71h éq. TD. Lors de l'année universitaire 2007-2008, j'ai été le responsable des travaux pratiques. Ces travaux pratiques concernaient 21 étudiants répartis en 3 groupes. Le volume horaire total était de 120h de TP.

Lors des cours magistraux, j'ai pu introduire cette nouvelle approche et indiquer comment l'intégrer dans une démarche de développement logicielle. Cette introduction est soutenue par un exemple issu du monde industriel. Ensuite, sont détaillées les technologies clefs que sont la métamodélisation et la transformation de modèle. Cette dernière partie me permet d'introduire la programmation déclarative, ainsi que la reconnaissance de patrons. Le dernier cours de cette série indique les modes émergents de définition de syntaxes concrètes pour les langages de modélisation.

Les séances de travaux dirigés sont organisés alternativement selon deux modes. Dans le premier mode, les étudiants doivent résoudre le problème de la définition d'un sous-ensemble du langage des diagrammes d'états d'UML. Dans le second mode, ils définissent et appliquent leurs solutions sur les outils qu'ils auront à utiliser lors des séances de travaux pratiques. Ce mode utilise la technique dite du dojo où une personne travaille sur un ordinateur, aidé par les autres membres du groupe et l'animateur (l'enseignant), puis est rapidement remplacé par un autre membre du groupe. En plus d'un bonne introduction aux outils, cette technique a l'avantage de montrer les erreurs communes et leur solution, tant au niveau technique que théorique.

Les travaux pratiques sont organisés en projet de développement de langage. Le projet est découpé en tâches que chaque étudiant doit réaliser. Le travail réalisé pour chacune des tâches est le support nécessaire de la tâche suivante. L'avantage de cette méthode d'enseignement est de préparer le futur ingénieur informaticien à fournir un travail constant et cohérent, motivé par la perspective d'un but précis.

Le but choisi est la modélisation et l'outillage d'un langage appris par ailleurs de manière théorique. Ce langage, nommé IACA, est développé par le Dr. Didier Bresch dans le cadre de ses cours sur la programmation multitâche et temps réel, sans disposer pour le moment d'environnement de développement. Il s'agit d'un langage de composants réutilisables pour les plates-formes embarquées de type microcontrôleur. Lors de ces travaux pratiques, les étudiants ont pour travail de modéliser les concepts du langage par un métamodèle, de formaliser des contraintes de cohérence des modèles, et de programmer un microcontrôleur par transformation de modèle.

4.4. Génie Logiciel (ENSISA - 2009)

Je suis responsable du cours de Génie Logiciel depuis l'année universitaire 2008-2009 que j'ai du monter dans son intégralité. Il concerne des étudiants de la section informatique et réseaux de l'ENSISA au niveau M1 et occupe 80h éq. TD. Pour cette première année universitaire, le cours a concerné 33 étudiants.

Le cours de génie logiciel est présenté comme une introduction aux divers aspects du développement de logiciels. Chaque chapitre présente un métier particulier de la chaîne de développement, sans pour autant se vouloir exhaustif. Le but est de donner aux futurs ingénieurs les clefs qui permettront de comprendre et de s'intégrer rapidement au monde industriel. Si le cours ne se focalise pas sur une méthode ou une notation en particulier, mon souhait est qu'il permette aux futurs ingénieurs de pouvoir s'adapter aux usages présents et futurs.

Les chapitres abordés couvrent le cycle de vie du logiciel après une description de celui-ci: ingénierie des besoins, architecture globale et détaillée, composants et contrats d'assemblage, programmation défensive, animation de contrats, test et développement dirigés par les tests. Sont également présentés des outils d'aide au développement: gestionnaires de configuration, référentiels de test, générateurs de tests, forums, systèmes de gestion de bugs. En parallèle, s'est déroulée une étape de l'Agile Tour à laquelle étaient conviés les étudiants (voir section 4.6, page 34), afin de les sensibiliser aux méthodes agiles, et de façon plus générale aux aspects humains.

Les travaux dirigés portent sur l'étude complète d'un système simple de contrôle d'ascenseur. A l'instar du cours, l'étude porte sur chacun des aspects de la chaîne de développement (besoins, architecture, tests). Afin de réaliser les exercices, une méthode de développement simple est présentée. Cette méthode, Fondue[1], également enseignée à l'École Polytechnique Fédérale de Lausanne et à l'université McGill à Montréal, allie la notation UML à la méthode de développement Fusion. Cependant, nous y intégrons des aspects essentiels des méthodes actuelles comme l'incrémentalité et le raffinement. À l'instar des TD d'IDM (section 4.3, page 32), des séances de dojo permettent également de découvrir les outils manipulés en travaux pratiques: JUnit comme environnement de développement de tests, Smartesting Test Designer comme simulateur de modèle UML et comme générateur de tests basés modèles.

Les travaux pratiques visent, en 3 séances de 4 heures pour chaque étudiant, à l'implémentation du système étudié en travaux dirigés, soit le système de commande d'un ascenseur. Une spécification formelle du système est donnée ainsi qu'un simulateur d'ascenseur (une maquette réelle étant en préparation). La réalisation doit se faire de manière dirigée par les tests. La notation dépend non seulement de la bonne réalisation fonctionnelle du système, mais également de la qualité des tests, évaluée selon leur couverture en termes de fonctionnalité et de code. Les développements se font par binôme et doivent être supportés par un gestionnaire de configuration (CVS) préalablement mis à disposition.

4.5. Encadrement de projets de semestre (ENSISA 2008-2009)

À l'ENSISA, les projets de 2ème année permettent à 2 étudiants (niveau M1) de traiter pendant 192h à temps plein d'un sujet proposé par un enseignant. Les projets de 3ème année (niveau M2) durent eux 150h, mais ont lieu en même temps que les cours. J'ai jusqu'à présent eu la responsabilité de 3 projets de 2ème année, et 1 de 3ème année.

Les projets intitulés "Adaptateurs de modèles" (2ème année) et "Mise en place d'un mécanisme d'événements sur une vue modèle" (3ème année) ont pour cadre le projet open source *Blanddern (p. 25)*. Dans ce cadre, les étudiants ont pu découvrir et mettre en oeuvre des technologies de type dépositaire de modèles, programmation par aspect, ou programmation logique (Prolog).

1. http://lgl.epfl.ch/research/fondue/index.html

Les projets intitulés “Développement de tests pour une application Javascript” (2ème année) et “Développement d'un visualisateur de comportement d'internaute” ont pour cadre une application de suivi des visites d’internautes intégré dans un site commercial. Le premier projet avait pour but l’écriture des tests fonctionnels pour l’application internet et de stress pour l’application, ainsi que la mise en oeuvre d’un dépositaire de tests. Le second projet a pour but d’améliorer l’application de suivi en y intégrant un visualisateur des mouvements de la souris de l’internaute. Dans les deux cas, les étudiants ont pu découvrir les applications dites “web 2.0” (Javascript, AJAX, DOM, etc.).

4.6. Agile Tour 2008

Dans le but de sensibiliser les étudiants au monde du travail, il est toujours bon de leur faire rencontrer des industriels avec une grande pratique. C’est ainsi que j’ai souhaité accueillir à l’ENSISA une étape de l’Agile Tour 2008[1] qui s’est déroulée l’après-midi du 3/10/2008. Cette conférence métier vise à promouvoir les méthodes de développement logicielles de type agiles, dont le succès s’affirme de plus en plus avec le temps. Si cette conférence peut viser un public d’étudiants, elle concerne également les industriels qui sont venus en nombre à l’événement. C’est ainsi que les étudiants ont pu non seulement écouter des présentations données par des industriels et des consultants du secteur du génie logiciel, mais aussi discuter avec les membres participants. Je me suis occupé de toute l’organisation de l'accueil de l’étape, à savoir la publicité (listes de diffusions, article dans le Journal des Entreprises), l'accueil physique (salle, pot,...), ainsi que de la sélection du programme.

4.7. Projet d’enseignement

La complexification des logiciels actuels rend l’enseignement du *génie logiciel* particulièrement important, de la phase de l’établissement du cahier des charges à la phase de test. De par sa capacité à capturer les abstractions et à orienter les processus de développement logiciel, l’*ingénierie dirigée par les modèles* semble être un bon support pour ce genre de formation. Si le génie des méthodes est important, il ne faut pas non plus perdre de vue les méthodes elles-mêmes, qu’elles soient adaptées au développement rapide au sein d’équipes restreintes (comme eXtreme Programming), ou au développement de logiciels fiables (comme B). Mon expérience dans l’enseignement de la méthode *Fondue*, qui allie la méthode Fusion à la notation UML, m’a montré qu’une telle méthode pouvait être un bon support pour l’enseignement.

La formation à l’informatique se doit également de contenir de nombreux aspects pratiques, en relation avec les besoins actuels de l’industrie, mais aussi en tenant compte des résultats de la recherche qui ont désormais passé le cap de la mise en application. Par exemple, l’enseignement de paradigmes de programmations, tels la programmation par *objet*, par *composants*, par *aspects*, ou par *contrat*, me semblent importants, tout en les mettant en pratique par l’utilisation de technologies cohérentes comme Java, J2EE, AspectJ ou JML. Une attention particulière doit également être portée aux outils de développement (collaboration, IDE...).

J’ai également la chance d’avoir une expérience industrielle en tant qu’ingénieur logiciel, et ai pu participer à de nombreux projets de développement collaboratifs (voir section 3.13, page 25). J’aimerais également faire partager cette expérience tant sur des aspects théoriques, comme indiqué ci-dessus, que sur des aspects plus pratiques, comme la maîtrise des langages de programmation, la conception d’interfaces utilisateur, la maîtrise des technologies internet

1. http://www.agiletour.com/

(comme HTML, Javascript, PHP), la conception et l'utilisation de bases de données, et bien sûr des outils de développement eux-mêmes (développement collaboratif, gestion de la ligne de compilation, installateurs). Une de mes priorité est également de rester en phase avec le monde de l'entreprise, auquel nous devons préparer les étudiants. Il s'agit bien sûr de garder à l'esprit qu'une carrière s'étale sur de très longues périodes du point de vue de l'informatique, et de ne pas préparer les futurs ingénieurs qu'aux technologies du moment, mais de leur donner les clefs pour appréhender la nouveauté.

Du fait de cette forte évolutivité de la discipline, un informaticien se doit de perpétuellement veiller à remettre à jour ses connaissances. Ainsi, comme j'ai déjà pu le montrer, je suis toujours prêt à acquérir de nouvelles compétences et bien sûr à les transmettre. C'est dans cette optique que je me prépare à une immersion en équipe de développement au sein un projet industriel de large ampleur (Dassault Systèmes).

5. Adresses utiles

Vous trouverez ici les coordonnées des divers encadrants de mes activités de recherche ou d'enseignement.

Prof. Michel Hassenforder
groupe Logiciel et Systèmes Intelligents (LSI)
du laboratoire Modélisation, Intelligence, Processus Systèmes (MIPS - EA2332),
École Nationale Supérieure d'Ingénieur Sud Alsace (ENSISA)
de l'Université de Haute Alsace (UHA)

Co-responsable du groupe LSI à l'ENSISA,
responsable de stage de mon Diplôme de Recherche Technologique,
ancien professeur lors de mes études à l'ESSAIM.

12, rue des frères Lumière
68 093 Mulhouse Cedex

michel.hassenforder@uha.fr

03 89 33 69 70

Prof. Bernard Thirion
groupe Logiciel et Systèmes Intelligents (LSI)
du laboratoire Modélisation, Intelligence, Processus Systèmes (MIPS - EA2332),
École Nationale Supérieure d'Ingénieur Sud Alsace (ENSISA)
de l'Université de Haute Alsace (UHA)

Co-responsable du groupe LSI à l'ENSISA,
ancien professeur lors de mes études à l'ESSAIM.

12, rue des frères Lumière
68 093 Mulhouse Cedex

michel.hassenforder@uha.fr

03 89 33 69 70

Prof. Pierre-Alain Muller
Premier vice-président, chargé des systèmes d'information
de l'Université de Haute Alsace (UHA)

Responsable UHA pour le projet VETESS,
rapporteur de ma thèse,
ancien CEO d'ObjeXion Software S.A.,
ancien professeur lors de mes études à l'ESSAIM.

12, rue des frères Lumière
68 093 Mulhouse Cedex

pierre-alain.muller@uha.fr

06 76 88 74 64

http://www.irisa.fr/triskell/members/pierre-alain.muller/

Dr. Thomas Baar
Tech@Spree Engineering GmbH

Enseignant chercheur à l'École Polytechnique Fédérale de Lausanne,
au Laboratoire de Génie Logiciel, jusqu'au 30/09/2007,
responsable du cours magistral de génie logiciel,
directeur de ma thèse.

Bülowstr. 66
D-10783 Berlin (Allemagne)

thomas.baar@acm.org

+49 30 235520 48

http://lgl.epfl.ch/members/baar/

Prof. Jean-Marc Jézéquel
Directeur de l'équipe Triskell
de l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA)
de l'Institut National de Recherche en Informatique et en Automatique (INRIA)

Ancien responsable INRIA du Projet ITEA CAFE dont j'assurais le suivi,
initiateur du projet MTL auquel j'ai participé.

Campus de Beaulieu
35 042 Rennes Cedex

jean-marc.jezequel@irisa.fr

02 99 84 71 92

http://www.irisa.fr/prive/jezequel/jmjFR.html

Dr. Philippe Studer
Maître de conférences
groupe Logiciel et Systèmes Intelligents (LSI)
du laboratoire Modélisation, Intelligence, Processus Systèmes (MIPS - EA2332),
École Nationale Supérieure d'Ingénieur Sud Alsace (ENSISA)
de l'Université de Haute Alsace (UHA)

Ancien directeur technique d'ObjeXion Software S.A.,
maître de stage de mon Diplôme de Recherche Technologique.

12, rue des frères Lumière
68 093 Mulhouse Cedex

philippe.studer@uha.fr

03 89 33 69 64

Prof. Bernhard Rumpe
Directeur du laboratoire Software Systems Engineering
de la Technische Universität Braunschweig

Rapporteur de ma thèse.

Informatikzentrum
Mühlenpfordtstr. 23
D-38106 Braunschweig (Allemagne)

b.rumpe@sse-tubs.de

+49 531 391 2276

http://www.rumpe.de/

Prof. Alain Wegmann
Directeur du Systemic Modeling Laboratory (LAMS)
de l'École Polytechnique Fédérale de Lausanne (EPFL)

Rapporteur de ma thèse.

EPFL - I&C - LAMS
Bâtiment BC (bureau BC103)
Station 14
CH-1015 Lausanne (Suisse)

alain.wegmann@epfl.ch

http://lamswww.epfl.ch/people/wegmann