Une plateforme multilangage pour l'ingénierie des systèmes d'information

Marie-Noëlle Terrasse — Marinette Savonnet — Eric Leclercq
George Becker

Laboratoire LE2I - Université de Bourgogne - B.P. 47870, F-21078 Dijon cedex
{marie-noelle.terrasse, marinette.savonnet, eric.leclercq}@u-bourgogne.fr
becker@khali.u-bourgogne.fr

RÉSUMÉ. *Dans cet article, nous illustrons le lien entre abstraction et métamodélisation dans l'ingénierie des systèmes d'information. Nous montrons en particulier comment plusieurs niveaux de langages sont en général imbriqués dans les environnements de métamodélisation afin de rendre compte des choix faits (à différents niveaux d'abstraction, pour différents types d'utilisateurs, etc.). Nous proposons une architecture de métamodélisation qui repose sur un triple niveau de description : modèle, métamodèle et paradigme de modélisation. La description de niveau métamodèle est unilangage et formalisée tandis que la description de niveau paradigme de modélisation est multilangage et informelle.*

ABSTRACT. *In this paper, we draw a link between abstraction and metamodeling in the information system engineering. We underline how multiple levels of languages can be embedded in metamodeling environments in order to report on the choices being made (at different levels of abstraction, for different types of users, etc.). We propose a metamodeling architecture which is based on a three-level description: model, metamodel, and modeling paradigm. The description of the metamodel level is unilingual and formal, while the description of the modeling paradigm level is multilingual and informal.*

MOTS-CLÉS : *métamodélisation, UML, ingénierie des systèmes d'information, descriptions multilangage.*

KEYWORDS: *metamodeling, UML, information system engineering, multilingual descriptions.*

RSTI - ISI – 8/2003. Systèmes d'information et langage naturel, pages 11 à 32

1. Introduction

La réflexion que nous menons est centrée sur un paradoxe apparent liant ingénierie des systèmes d'information, métamodélisation et langages, *i.e.*, une double évolution vers le besoin de descriptions plus formelles d'une part et de descriptions plus proches du langage naturel d'autre part. Nous avons choisi deux exemples de cette évolution « divergente » des besoins en terme de langage de description : pour l'utilisation de méthodes formelles et pour l'utilisation de techniques de métamodélisation en ingénierie du logiciel.

Tout d'abord, nous assistons à une extension du domaine d'application des méthodes formelles en ingénierie du logiciel. Initialement strictement limitées à la vérification de consistance de modèles, les méthodes formelles empiètent maintenant d'une part sur la capture de besoins des utilisateurs et d'autre part sur la génération semi-automatique de code. Avec cette extension du domaine d'application des méthodes formelles, s'amplifie un « vieux » problème. Les interlocuteurs compétents (que ce soit pour la phase de description des besoins ou pour la phase de génération de code) sont rarement des spécialistes de méthodes formelles. De même, les spécialistes des méthodes formelles méconnaissent en général les subtilités des domaines d'application à modéliser. Il apparaît donc nécessaire de trouver un niveau (un langage) de description « intermédiaire » qui puisse servir de support à un travail concerté des spécialistes du domaine et des spécialistes des méthodes formelles.

Par ailleurs, l'ingénierie des systèmes d'information est devenue au fil des années un domaine d'une extrême complexité : complexité des systèmes réels, complexité des environnements de déploiement (avec les besoins de distribution, d'interopérabilité, d'ouverture sur le web), complexité des environnements d'exploitation (avec de nombreux intervenants ayant des besoins et des prérogatives différents), complexité liée aux demandes des utilisateurs (en terme de fiabilité des systèmes, de sécurité, etc.). En réponse à de telles attentes, l'ingénierie des systèmes d'information a mis en place des mécanismes d'abstraction sophistiqués qui sont actuellement pris en charge par les environnements de métamodélisation. De tels environnements, souvent basés sur UML, intègrent aussi des langages formels (logique, théorie des ensembles, Z, VDM) ainsi que les méthodes formelles correspondantes. Néanmoins, conjointement à ces langages formels, l'ingénierie des systèmes d'information a recours à des langages moins précis mais plus lisibles (comme, par exemple, le langage naturel). Le mode de travail effectif des équipes d'ingénierie utilisant la métamodélisation est donc basé sur l'utilisation de notations multiples (incluant le langage naturel) comme support de travail. Une des options possibles semble ainsi être d'inclure le langage naturel comme une des notations devant supporter la communication des divers spécialistes impliqués dans les processus d'ingénierie du logiciel.

Dans la suite de cette introduction, nous illustrons – sur l'exemple d'UML – le lien entre abstraction et métamodélisation dans l'ingénierie des systèmes d'information. Puis nous montrons comment plusieurs niveaux de langage sont en général imbriqués dans les environnements de métamodélisation et nous proposons de considérer un triple niveau de description : modèle, métamodèle et paradigme de modélisation.

1.1. *Les mécanismes d'abstraction*

L'abstraction s'exerce en métamodélisation sous deux formes : abstraction par conceptualisation et abstraction par projection. UML semble être devenu la référence en ce qui concerne ce second mode d'abstraction.

– **L'abstraction par conceptualisation** est calquée sur la dichotomie instance-modèle instituée par les bases de données entre ce qui est lié à un état temporaire de l'application et ce qui caractérise de façon permanente l'application. Le plus souvent, ainsi que l'a proposé l'OMG [UML 00], la description d'un système d'information est structurée en quatre couches qui correspondent à des niveaux d'abstraction différents : instance, modèle, métamodèle et méta-métamodèle. La couche métamodèle définit les constructeurs qui sont nécessaires à l'élaboration des modèles pour un domaine d'application particulier. Le métamodèle est donc une description – à haut niveau d'abstraction – du domaine d'application. Les profils de l'OMG, par exemple, construisent des métamodèles UML pour des domaines spécifiques. Une des premières extensions proposées – pour les applications temps réel – a été largement adoptée. Chaque nouveau métamodèle est construit en utilisant les mécanismes d'extension d'UML[1]. La couche méta-métamodèle décrit comment le monde réel est perçu. Le méta-métamodèle choisit la logique sous-jacente, (*e.g.*, logique booléenne ou logique modale), le modèle temporel ou spatial qui sera utilisé pour représenter la sémantique du monde réel, etc.

– **L'abstraction par projection** repose sur les principes de séparation et combinaison des points d'intérêt [BEZ 98, BRE 98]. La séparation des points d'intérêt est implémentée par des modèles constitués de plusieurs vues qui sont orthogonales : la vue des fonctionnalités externes, la vue structurelle, la vue de la dynamique, etc. Ces vues orthogonales doivent être articulées en une spécification consistante du système : c'est la combinaison des points d'intérêt. Elle doit garantir à la fois que les vues sont consistantes entre elles et qu'aucune information ne manque. Dans la plupart des architectures de métamodélisation, c'est le métamodèle qui est en charge de la séparation et de la combinaison des points d'intérêt.

1.2. *Les environnements de métamodélisation*

Les environnements de métamodélisation qui sont proposés (que ce soit dans l'industrie ou le monde universitaire) offrent un noyau qui permet l'élaboration de métamodèles (décrivant un domaine d'application) et de modèles dérivés. Dans le cadre de tels environnements, l'ingénierie des systèmes d'information est un processus constitué de deux principales étapes : choix d'un métamodèle puis création du modèle proprement dit dans le cadre défini par le métamodèle. En pratique, il s'avère que le choix d'un métamodèle est une tâche ardue. En effet, afin d'effectuer un choix en toute connaissance de cause, il faudrait connaître de façon précise tous les métamodèles

1. Les mécanismes d'extension d'UML sont les tagged values, les contraintes et les stéréotypes.

disponibles afin de déterminer s'il est possible de partir d'un métamodèle existant ou pas. De plus, pour pouvoir modifier (par une extension) un métamodèle existant, il faut maîtriser l'extension qui a produit ce métamodèle, ce qui est en général loin d'être simple.

Lorsque l'on étudie les méthodes d'élaboration des métamodèles, on se rend compte que la description des domaines d'application en terme de métamodèles est menée d'une façon bien différente. Les équipes d'ingénierie travaillent effectivement en réutilisant une bibliothèque de métamodèles mais les descriptions utilisées pour choisir parmi ces métamodèles sont des descriptions semi-formelles et souvent multi-langages. De telles descriptions peuvent être ambigues mais elles sont plus lisibles aux yeux de la plupart des intervenants. Nous appelons *paradigme de modélisation* une telle description abstraite d'un domaine d'application. Un paradigme de modélisation peut être exprimé à l'aide de langage naturel (anglais en général), théorie des ensembles, logique... Ainsi le processus de modélisation est en fait constitué de trois étapes :

– définition d'un paradigme de modélisation (qui est le plus souvent construit à partir d'un paradigme de modélisation choisi dans une bibliothèque),

– instantiation de ce paradigme de modélisation en un métamodèle (en utilisant les mécanismes d'extension d'UML),

– instantiation de ce métamodèle en un modèle (ou en une cascade de modèles de plus en plus concrets).

Notre objectif est donc de proposer une plateforme de métamodélisation dans laquelle les paradigmes de modélisation puissent être intégrés et exploités. En sections 2 et 3, nous discutons du besoin d'utiliser plusieurs niveaux de langages (y compris le langage naturel) dans le cadre de l'ingénierie des systèmes d'information. En section 4, nous présentons notre architecture de métamodélisation basée sur le triplet paradigme de modélisation, métamodèle et modèle. En conclusion (section 5), nous faisons un bilan des besoins que fait apparaître une telle approche.

2. Paradigmes de modélisation et langage naturel

Ainsi que nous l'avons suggéré en introduction, le couple métamodèle-modèle est insuffisant dans la mesure où il ne rend pas compte du processus de modélisation dans son ensemble : tout le travail initial, mené pour la définition du paradigme de modélisation, est perdu. Par exemple, Price *et al.* [PRI 99a] présentent une extension d'UML pour la modélisation des aspects temporels dans les systèmes d'information géographiques. Cette extension est tout d'abord longuement motivée par une étude des modèles de temps du projet TAU [TAU 97]. Mais ces modèles, qui ne sont pas traduits en métamodèles UML, n'apparaissent pas dans une architecture de métamodélisation standard. De même pour l'extension de Robbins *et al.* [ROB 98] pour le langage C2 de description des architectures logicielles qui s'appuie fidèlement sur les travaux de Medvidovic *et al.* [MED 96, MED 99].

Le risque d'une prolifération de métamodèles – avec comme conséquence une difficulté accrue pour l'interopérabilité des systèmes d'information – est d'autant plus important qu'il est difficile de retrouver ou d'exploiter les métamodèles existants. Une première façon d'éviter cette prolifération de métamodèles est de fournir un jeu d'extensions standardisées pour les différents domaines d'application non couverts par le métamodèle standard d'UML. Cette approche a été choisie par l'OMG avec les « profiles » [OMG 98, OMG 99]. Un autre mode de contrôle – plus souple – consiste à organiser les métamodèles en une hiérarchie [COO 99, FAL 94]. Chaque extension est alors positionnée comme descendant d'un métamodèle existant dans la hiérarchie. La qualité d'un tel contrôle est alors fortement dépendant de la qualité de cette hiérarchie : il faut garantir que les liens entre métamodèles sont précisément transcrits dans la hiérarchie. Cela suppose que les équipes d'ingénierie puissent facilement positionner leurs besoins par rapport aux métamodèles existants.

Le principal point d'achoppement est donc le besoin d'une structuration des métamodèles qui soit précise tout en restant lisible (sans prérequis) par tout type d'utilisateur : on retrouve l'argumentaire de Heitmeyer [HEI 98a] pour des « light-weight formal tools », outils formels mais offrant des interfaces adaptées aux non-spécialistes. Notre proposition repose sur l'idée d'une double description (l'une formelle pour les lecteurs spécialistes et l'autre informelle pour tout public) avec un couplage fort entre les deux niveaux de description. Dans la suite de cette section, nous proposons trois exemples afin de montrer comment les métamodèles développés sont effectivement positionnés par rapport à des réflexions existantes sur le domaine d'application et nous soulignons dans chaque cas les langages utilisés.

2.1. *Les modèles de temps*

Dans le cadre du projet TAU [TAU 97], l'équipe du laboratoire Timelab a défini un modèle de temps. La définition, en anglais, de ce modèle de temps énonce les propriétés mathématiques de l'axe du temps[2], indique quel calendrier est utilisé[3], etc.

Kakoudakis *et al.* [KAK 96, SVI 97] ont repris ce modèle de temps afin d'en dériver une sémantique pour une extension temporelle d'UML appelée TUML. Ils définissent les stéréotypes nécessaires : ≪date≫, ≪time≫, ≪timestamp≫, ≪period≫, ≪timepoint≫, etc.

Price *et al.* [PRI 99b] reprennent la proposition de Kakoudakis et proposent une extension spatio-temporelle d'UML, appelée STUML, qui met en place – sous forme de stéréotypes – différents modèles de temps basés sur les propositions du projet TAU. Chaque stéréotype temporel est associé à une boîte de spécification qui précise le modèle de temps utilisé, l'unité, le type d'interpolation, et les dimensions. Par exemple

2. Projet TAU : "*The time axis is considered to be discrete, linear, totally ordered and bounded at both ends.*"
3. Projet TAU : "*The model adopts the Gregorian calendar and supports its default granularities.*"

la densité de population a un attribut temporel dont la boîte de spécification indique que le temps est monodimensionnel, que l'interpolation suppose les valeurs statiques entre deux mesures, que le temps est linéaire et régulier (avec une fréquence de mesure annuelle), que l'unité de temps est instantanée (et non de type intervalle). Cette proposition – bien qu'exprimée aussi en terme de stéréotypes – est plus abstraite que celle de Kakoudakis.

2.2. *Héritage et synchronisation*

La notion d'anomalie d'héritage dans les mécanismes de synchronisation a été identifiée par Matsuoka *et al.* [MAT 93] et décrite comme une rupture d'encapsulation résultant d'un conflit entre héritage et concurrence. Matsuoka *et al.* ont donc proposé de considérer les schémas de synchronisation comme des détails d'implémentation qui, à ce titre, ne sont pas transmis aux sous-classes lors de l'héritage. Leur présentation, en anglais, est placée dans le cadre de la programmation orientée objet[4] puis illustrée par du code C++.

De Miguel *et al.* [DEM 97] proposent une extension d'UML qui modifie la signification de l'héritage. Ils définissent un stéréotype ≪passive≫ pour les classes dont les instances ne changent d'état que sur exécution d'une méthode ou opération. Ces classes passives contiennent un protocole de synchronisation mais ne sont pas sujettes au problème d'anomalie d'héritage. Ils définissent aussi un stéréotype ≪protected≫ pour les classes dont les instances ne peuvent changer d'état que par des exécutions – en exclusion mutuelle – d'opérations atomiques. Ces classes ne contiennent pas de protocole de synchronisation. L'héritage est alors défini de façon adaptée à ces différentes classes.

Herrero *et al.* [HER 00] définissent dans UML un nouveau diagramme issu du Diagramme StateChart. Ce nouveau diagramme est construit à partir d'un stéréotype de classe appelé ≪synchronization≫. La principale différence entre le diagramme de synchronisation ainsi constitué et le Diagramme StateChart est la nécessité d'invoquer une méthode (qui prend en charge la synchronisation) lors des transitions. Chaque politique de synchronisation est définie sous forme d'une classe de synchronisation et chaque objet synchronisé appartient à une classe de synchronisation.

2.3. *Les langages ADL*

Un autre exemple significatif, que nous allons reprendre en détail dans la suite, concerne un langage, nommé C2[5], permettant de décrire des architectures basées sur les notions de composants et messages. Medvidovic *et al.* [MED 99] décrivent – en

4. Matsuoka *et al.* : "*…to separate and localize the synchronization schemes from the main bodies of methods, allowing fine-grained inheritance/overridding …*"
5. C2 est un ADL (Architecture Description Language).

anglais – les architectures traitées par le langage C2 en terme de composants qui s'échangent des messages (nommés « requests » et « notifications ») *via* des interfaces (nommées « top » et « bottom ») reliées par des connecteurs. Ils posent des contraintes telles que *les composants utilisent obligatoirement des connecteurs pour échanger des informations*. On peut noter qu'une description antérieure de C2 avait été donnée en langage Z, par Medvidovic *et al.* [MED 96].

Robbins *et al.* [ROB 98] reprennent – de façon fidèle – ces descriptions sous forme de stéréotypes UML. Nous utilisons comme exemple (en section 4) ces deux niveaux de description.

A travers ces quelques exemples, que nous pourrions multiplier, apparaît une tendance à décrire la sémantique d'un domaine d'application (comme pour les langages ADL de la famille composant-message) ou d'un domaine ontologique (comme pour les aspects temporels) en utilisant plusieurs langages. On peut ainsi distinguer plusieurs niveaux de description.

– Les descriptions intuitives, généralement exprimées en langage naturel. Elles ont un objectif pédagogique et visent une audience large.

– Les descriptions formelles, par exemple en langage Z. Elles permettent de décrire sans ambiguïté mais ne s'adressent qu'à un public restreint.

– Les descriptions concrètes, par exemple en C++. Elles ne sont pas toujours faciles à comprendre (il faut faire la part des choses entre les concessions à la syntaxe du langage et le contenu proprement dit). Ces descriptions concernent un public d'informaticiens avertis.

Les utilisations de ces différents langages sont le plus souvent imbriquées : une même caractéristique étant souvent décrite à la fois en langage naturel et dans un langage plus précis mais d'audience restreinte. Sur ces quelques exemples, on peut noter que la chronologie des descriptions (plus formelle, plus concrète ou plus intuitive d'abord) n'est pas la même d'un projet à l'autre.

3. Méthodes formelles et langage naturel

Avec le développement des modèles multivues, un des freins à l'utilisation des méthodes formelles dans le cadre de l'ingénierie du logiciel a en partie disparu. Dans un article de 1996, Clarke *et al.* [CLA 96] avaient déjà souligné le besoin d'appuyer l'utilisation des différentes méthodes formelles disponibles sur des notations adéquates : "*Given that no one formal method is likely to be suitable for describing and analyzing every aspect of a complex system, a practical approach is to use different methods in combination. When combining methods it is important to consider finding a suitable style for using different methods together...*". Depuis, sur la base d'UML entres autres, de nombreux travaux ont été présentés dans lesquels des méthodes formelles (basées pour la plupart sur la vérification symbolique de modèle) permettent d'améliorer les modèles, voire d'aider à leur construction puis à leur implémentation. Les méthodes

formelles cessent donc d'être un îlot de rigueur dans le déroulement « chaotique » d'un processus guidé en grande partie par le seul savoir-faire des intervenants. Ainsi que nous l'illustrons en figure 1, les méthodes formelles gagnent du terrain pour la capture des besoins utilisateurs et pour la génération automatique de code, voire le test du code. La validation formelle de modèles [CLA 96, OBE 00] permet de garantir que le modèle décrit est consistant, le raffinement de modèle [BAC 98] permet d'améliorer (au sens formel) un modèle considéré comme valide. La validation de modèles en fonction des demandes des utilisateurs [AND 99, DAS 00, JEF 01] est un premier pas vers la réduction de ce problème incontournable : « *Il est impossible de passer de l'informel au formel par des méthodes formelles* ». En amont, certains outils permettent de construire le métamodèle nécessaire à la description d'un modèle [CLA 01]. En aval, la génération automatique de tests à partir des modèles [AMM 99, LEG 01] permet de préparer le passage au code.

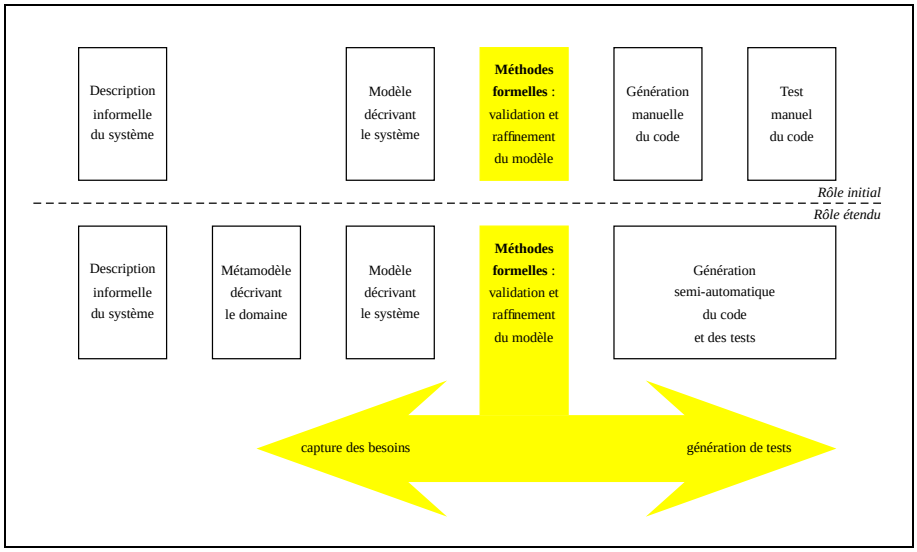

Figure 1. *Evolution du rôle des méthodes formelles dans le processus d'ingénierie du logiciel*

Dans les sections suivantes, nous présentons tout d'abord deux environnements qui incluent des méthodes formelles dans leur rôle étendu. L'environnement SCR, développé par l'équipe de C. Heitmeyer [BHA 99, BHA 00, HEI 98c, JEF 01], a pour domaine d'utilisation privilégié les systèmes demandant un fort niveau de sécurité et de fiabilité. L'environnement MIC (Model Integrated Computing) de Nordstrom *et al.* [KAR 00, LED 01, NOR 99b] vise plus particulièrement les systèmes pour lesquels l'environnement physique et les abstractions logicielles choisies (*e.g.*, synchronisation, variables en virgule flottante ou fixe) sont en interaction permanente. Un des domaine d'application est par exemple celui des produits industriels avec logiciel embarqué (robots, téléphones cellulaires) [KAR 00, HÖP 01]. Nous reprenons ensuite le

besoin d'un langage intermédiaire (qui pourrait inclure un langage naturel) à partir des « outils formels allégés » (light-weights formal tools) de C. Heitmeyer [HEI 98b].

3.1. *La méthode Software Cost Reduction*

Cette méthode a été développée dans les laboratoires de la marine des Etats-Unis depuis les années 70. Elle vise à travailler aussi longtemps que possible sur le seul modèle et à ne lancer la phase de génération de code que lorsque le modèle est validé. Les outils proposés à ce jour [HEI 02, LEO 02] permettent de mettre en œuvre la méthode de travail suivante.

– Spécification des besoins des utilisateurs à l'aide d'un éditeur de spécification.

– Détection des violations de propriétés indépendantes de l'application (*e.g.*, cas manquants, ambiguïtés, définitions circulaires) en utilisant un vérificateur de consistence de la spécification.

– Validation de la spécification au regard des attentes des utilisateurs. On effectue pour cela des simulations à partir de la spécification (*e.g.*, en faisant tourner sur la spécification des scénarios de tests).

– Validation de la spécification sur les propriétés inhérentes à l'application (*e.g.*, sécurité, fiabilité). On utilise pour cela soit un vérificateur symbolique de modèle soit un prouveur de théorèmes.

– Production de générateurs de code (générant du C) directement à partir de la spécification.

– Validation de code par la génération de tests directement à partir de la spécification.

Plusieurs applications (dans lesquelles les propriétés liées à la sécurité sont essentielles) ont été développées sur la base des outils proposés. Les articles rendant compte de ces applications montrent la complémentarité des différents niveaux de validation utilisés. Les outils formels proposés sont – si possible – occultés par des interfaces graphiques qui peuvent être facilement adaptées à un domaine d'application particulier [HEI 98d].

3.2. *L'environnement MIC*

Il a été développé à partir des travaux sur les architectures multigraphes (MGA), la synthèse de programme à partir de modèle (Model Integrated Program Synthesis) et les environnements génériques de modélisation (Generic Modeling Environments). La méthode proposée est itérative. Elle consiste à travailler – à l'aide de méthodes formelles – sur la spécification elle-même puis à générer automatiquement un modèle exécutable qui sera testé et permettra de modifier la spécification. Nordstrom a proposé

un environnement de développement d'application basé sur UML avec la méthode suivante[6].

– Traduction des besoins des utilisateurs et des connaissances d'experts du domaine en un ensemble de « composants » du modèle. Le langage d'écriture des composants de modèle est graphique (basé sur UML). L'environnement propose un éditeur graphique spécifique qui intègre directement certaines caractéristiques/contraintes du domaine de référence (*i.e.*, du métamodèle).

– Définition de conditions pour constituer le modèle complet à partir des composants de modèle.

– Interprétation (translation sémantique) de modèle afin de produire des modèles exécutables à l'aide d'interpréteurs de modèles (qui se basent sur une bibliothèque de classes C++).

La base de ces outils est un langage formel qui doit être assez ouvert pour permettre de faire à la fois de la preuve de théorèmes, de la vérification de modèles et de la génération de modèles exécutables. Nordstrom présente un exemple sur les langages ADL ainsi que plusieurs exemples en ingénierie électrique.

3.3. *Des outils formels allégés au langage naturel*

Dans un article intitulé « *On the need for Practical Formal Methods* » [HEI 98a], Constance Heitmeyer a souligné le besoin de rendre les méthodes formelles utilisables par tous les intervenants en ingénierie du logiciel. Elle considère deux types de problèmes posés par l'utilisation des méthodes formelles.

Le premier type de problème est inhérent aux méthodes formelles proprement dites. Par exemple pour vérifier un algorithme, il faut utiliser un prouveur de théorème. Les difficultés « théoriques » qui peuvent apparaître nécessitent de faire travailler (parfois pendant un temps assez long) une équipe de spécialistes des méthodes formelles. Dans certains cas (Heitmeyer cite le cas d'une vérification d'algorithme de division en virgule flottante) l'algorithme pourra être réutilisé et le coût de la vérification est acceptable. Dans d'autres cas, l'algorithme n'a pas d'intérêt en dehors du système pour lequel il a été conçu ; le coût de la vérification est alors rédhibitoire.

Le second type de problème est plutôt lié à « l'implémentation » de la méthode formelle. Par exemple, lorsque l'on utilise un vérificateur symbolique de modèle, les propriétés à vérifier doivent être exprimées en logique temporelle (difficile à maîtriser) et les contre-exemples détectés peuvent être exprimés sous une forme faisant intervenir des centaines d'états de la machine d'état finie. Ce type de difficulté a déjà été résolu dans plusieurs cas. Par exemple les BDD (binary decision diagrams) per-

6. Le vocabulaire utilisé dans la thèse de Greg Nordstrom [NOR 99a] n'étant pas compatible avec le nôtre (les niveaux d'abstraction sont décalés : il appelle métamodèle ce que nous appelons modèle et paradigme de modélisation ce que nous appelons description informelle du système), nous avons « traduit » dans notre vocabulaire sa présentation.

mettent d'entrer des informations sous forme de tables (faciles à maîtriser). Cela a permis de proposer des outils « pousse-boutons » dans le domaine de la vérification de consistence.

Une partie des problèmes liés à l'utilisation des méthodes formelles trouve donc sa solution dans l'utilisation d'un langage intermédiaire entre l'outil formel tel qu'il est proposé aux spécialistes et sa version dédiée aux utilisateurs non spécialistes. Nous pensons que la même démarche est nécessaire dans le cadre de la métamodélisation : il faut permettre aux intervenants du processus d'ingénierie qui ne sont pas familiers avec l'expression des métamodèles de pouvoir utiliser une architecture de métamodélisation. Le langage intermédiaire que nous proposons – au vu de la pratique des intervenants en UML, voir section 2 – doit inclure le langage naturel.

Dans la section suivante, nous présentons notre architecture de métamodélisation qui est un premier pas vers l'intégration du langage naturel comme support de communication entre intervenants dans le processus d'ingénierie du logiciel.

4. Notre architecture de métamodélisation

Notre architecture de métamodélisation repose sur une variante de l'architecture à quatre couches proposée par l'OMG dans laquelle nous organisons les deux couches les plus abstraites (méta-métamodèle et métamodèle) en une structure miroir. La couche méta-métamodèle contient les descriptions – informelles et multinotations – de paradigmes de modélisation. Ceux-ci décrivent la façon dont le monde réel est perçu. La syntaxe de description est très peu contraignante mais les paradigmes de modélisation doivent être organisés en un poset[7]. La couche métamodèle contient des descriptions – formelles et unilangage – de domaines d'applications : ce sont des instanciations du métamodèle UML. Les métamodèles sont organisés en une hiérarchie d'héritage qui est un miroir du poset des paradigmes de modélisation.

4.1. *La couche méta-métamodèle*

La couche méta-métamodèle de notre architecture décrit – sous forme de paradigmes de modélisation – la sémantique du monde réel. Nous avons choisi de décrire les paradigmes de modélisation en terme de concepts et de contraintes qui sont exprimés en utilisant éventuellement plusieurs langages tels que le langage naturel, la logique, la théorie des ensembles, etc. Un paradigme de modélisation est décrit par deux ensembles.

– Un ensemble $\mathcal{E}l^3(mp)$ de concepts élémentaires qui sont identifiés par un nom et dont la sémantique peut être exprimée au moyen des divers langages disponibles. Un concept peut être décrit de façon plus ou moins précise : selon les paradigmes de

7. Poset : ensemble muni d'un ordre partiel. Voir Grätzer [GRÄ 71].

modélisation une seule ou plusieurs variantes du même concept peuvent apparaître. Par exemple, le concept de *classe* peut être décliné en *classe interface*, *classe abstraite* et *classe d'implémentation*.

– Un ensemble $\mathcal{C}^3(mp)$ de contraintes portant sur les concepts de $\mathcal{E}l^3(mp)$. Les contraintes peuvent être plus ou moins précises (lorsqu'elles ne sont pas exprimées dans un langage formel). Par exemple, nous pouvons avoir la contrainte *chaque objet appartient à une et une seule classe* plutôt que *chaque objet appartient à une classe*.

Dans un cadre général (*i.e.*, en acceptant toute description de tout paradigme de modélisation possible) il serait difficile d'imposer à ces descriptions la moindre contrainte et, par conséquent, de garantir la moindre propriété exploitable. Falkenberg *et al.* [FAL 94] ont proposé d'organiser en une hiérarchie d'héritage des métamodèles : ils ont fourni une liste d'opérations permettant de mettre en place cette hiérarchie (*e.g.*, ajout, suppression, spécialisation de concepts). Il semble néanmoins que la construction d'une hiérarchie significative soit difficile lorsque les métamodèles sont constitués de concepts sans aucune référence sémantique commune.

Nous définissons – pour notre part – un sous-ensemble $Restrict_{MP}$ de la couche méta-métamodèle qui correspond aux paradigmes de modélisation compatibles avec la sémantique d'UML. $Restrict_{MP}$ contient le paradigme de modélisation général gmp (dont est issue la définition standard d'UML) ainsi que toutes les extensions possibles de ce paradigme de modélisation. Chaque nouveau paradigme de modélisation doit être défini comme une extension – plus ou moins lointaine – de gmp, ce qui est formalisé par une contrainte de subsomption entre paradigmes de modélisation. Un paradigme de modélisation mp_1 est *subsumé* par un paradigme de modélisation mp_2 (ce qui est noté par $mp_1 \preceq mp_2$) s'il y a à la fois *inclusion étendue des concepts* et *subsomption des contraintes*. L'inclusion étendue prend en compte la possibilité de retrouver une forme spécialisée d'un concept. La subsomption des contraintes prend en compte l'utilisation du mécanisme de preuve. Nous introduisons ainsi un ordre partiel entre les paradigmes de modélisation de $Restrict_{MP}$. Pour plus de détails, voir [TER 01b].

4.2. *Exemple de paradigme de modélisation*

Nous présentons en figure 3 un exemple de paradigme de modélisation (noté mp_4) pour le langage C2. L'objectif est d'obtenir une description du paradigme de modélisation de C2 à partir de la description en anglais de Medvidovic *et al.* [MED 99]. Nous allons montrer, sur une version simplifiée et partielle de cet exemple, la construction du paradigme de modélisation.

– Du texte "*In a C2-style architecture, connectors transmit messages between components while components maintain state, perform operations, and exchange messages with other components via two interfaces (named « top » and « bottom »). [...] Inter-component messages are either requests for a component to perform an operation, or notifications that a given component has performed an operation and changed*

state", on extrait les concepts de *connector*, *message*, *component*, *state*, *operation*, *interface*, ainsi que deux cas particuliers de messages (« request », « notification »).

– Les concepts *state* et *operation* existent déjà dans gmp, les autres concepts doivent être introduits dans le paradigme de modélisation mp_4. L'ensemble de concepts du paradigme de modélisation mp_4 est donc défini comme :

$$\mathcal{E}l^3(mp_4) = \mathcal{E}l^3(gmp) \cup \{\textit{connector}, \textit{component}, \textit{interface}, \textit{message}\}$$

– Du texte "*In the C2 style, components may not directly exchange messages ; they may only do so via connectors*", on extrait la contrainte $Contrainte_1$: *Les composants utilisent obligatoirement des connecteurs pour échanger des informations*.

– Du texte "*Request messages may only be sent « upward » through the architecture, and notification may only be sent « downward »*", on extrait les contraintes $Contrainte_2$: *Les requêtes sont transmises vers le bas de l'architecture* et $Contrainte_3$: *Les notifications sont transmises vers le haut de l'architecture*.

L'ensemble de contraintes du paradigme de modélisation mp_4 est alors défini comme :

$$\mathcal{C}^3(mp_4) = \mathcal{C}^3(gmp) \cup \{Contrainte_1, Contrainte_2, Contrainte_3\}$$

Le paradigme de modélisation mp_4 est donc défini directement à partir de gmp. Il appartient à $Restrict_{MP}$ et il y a une relation de subsomption entre mp_4 et gmp.

4.3. *La couche métamodèle*

Notre objectif est de construire la couche métamodèle comme un miroir du poset des paradigmes de modélisation. En conséquence, le paradigme de modélisation général gmp est instancié par le métamodèle UML (noté mm_{UML}), tout autre paradigme de modélisation mp étant instancié par une spécialisation du métamodèle UML. Le métamodèle résultant de l'instanciation, $mm = \mathcal{E}^{3,2}(mp)$, est donc défini par un ensemble $\mathcal{E}l^2(mm)$ de constructeurs UML et un ensemble $\mathcal{C}^2(mm)$ de contraintes.

A titre d'exemple, nous présentons l'extension, notée mm_4, proposée par Robbins *et al.* [ROB 98] pour le langage C2. Cette extension correspond au paradigme de modélisation mp_4. Le métamodèle mm_4 est construit comme héritier de mm_{UML}. Il est nécessaire, à partir de mm_{UML} et en utilisant les mécanismes d'extensions d'UML, d'instancier les concepts et contraintes introduites dans mp_4.

– Le **concept d'interface** existe dans gmp. Cependant la distinction entre interface « top » et « bottom » est propre aux interfaces C2. Les auteurs définissent donc un stéréotype des interfaces UML, noté $\ll C2 - interface \gg$, en ajoutant aux interfaces UML une valeur tag (de valeurs « top » et « bottom »).

– Le **concept de message** dans C2 est assez proche du concept d'opération dans gmp. Il y a cependant deux différences : un message C2 n'a pas de valeur résultat et on doit pouvoir distinguer les messages requêtes des messages notifications. Les auteurs

définissent donc un stéréotype des opérations UML, noté $\ll C2 - operation \gg$ en ajoutant aux opérations une valeur tag (de valeurs « request » et « notification ») et une contrainte OCL, notée OCL_4, pour interdire le renvoi d'un résultat.

– Le **concept de composant** de mp_4 n'existe pas dans gmp. La description des composants de C2 montre que les composants sont assez proches des classes mais 1) qu'ils n'ont pas d'attributs propres et 2) qu'ils ont exactement deux interfaces (une interface « top » et une interface « bottom »). Les auteurs définissent donc un stéréotype de class, noté $\ll C2 - component \gg$, avec les contraintes : OCL_5 pour interdire l'existence d'attributs propres, OCL_6 pour rendre obligatoire l'existence d'exactement deux $\ll C2 - interface \gg$ (avec des tag « top » et « bottom », respectivement), les contraintes OCL_2 et OCL_3 qui « traduisent » en OCL les contraintes $Contrainte_2$ et $Contrainte_3$ de mp_4.

– Le **concept de connecteur** de mp_4 n'existe pas dans gmp. Les connecteurs de C2 sont décrits comme étant assez proches des classes mais avec une limitation qui ne permet pas de les faire participer à des associations non binaires. Les auteurs définissent donc un stéréotype de *class*, noté $\ll C2 - connector \gg$, avec une contrainte OCL_7 pour limiter la multiplicité des associations portant sur un connecteur C2.

A ce point du travail, nous pouvons résumer dans le tableau donné en figure 2 les grandes étapes du passage du texte descriptif au métamodèle UML.

Concept de mp_4	est-ce un concept de gmp ?	stéréotype défini pour mm_4	constructeur UML utilisé	mécanisme d'extension utilisé
connector	non	$\ll C2 - connecteur \gg$	class	contrainte
message	non	$\ll C2 - operation \gg$	operation	contrainte et tag-value
component	non	$\ll C2 - component \gg$	class	contraintes
state	oui			
operation	oui			
interface	non	$\ll C2 - interface \gg$	class	tag-value

Figure 2. *Le passage du descriptif textuel au métamodèle UML*

4.4. *La structure miroir*

Nous définissons une fonction d'instanciation $\mathcal{E}^{3,2}$ qui associe à chaque paradigme de modélisation de $Restrict_{MP}$ son instanciation sous forme d'une spécialisation du métamodèle UML. Nous imposons au mécanisme d'instanciation de respecter l'ordre partiel des paradigmes de modélisation dans $Restrict_{MP}$. Nous garantissons que chaque métamodèle correspond au paradigme de modélisation à partir duquel il est instancié et que chaque lien d'instanciation correspond à un lien de subsomption. Nous appelons *règle de totale conformité de la fonction d'instanciation* cette contrainte forte qui nous permet de garantir l'effet miroir entre les couches méta-métamodèle et métamodèle. La règle de totale conformité a deux conséquences. Tout d'abord, il est nécessaire d'utiliser de l'héritage multiple pour construire la couche métamodèle afin

de pouvoir construire un métamodèle par double instanciation. Ensuite, il est nécessaire d'instancier tous les paradigmes de modélisation, y compris les paradigmes de modélisation qui sont ambigus. Afin de traiter ces problèmes d'ambiguïté, nous distinguons deux cas d'ambiguïté pour les paradigmes de modélisation : une ***ambiguïté initiale*** est une ambiguïté venant du paradigme de modélisation gmp lui-même (voir les travaux du groupe PreciseUML [EVA 98]), tandis qu'une ***ambiguïté d'extension*** provient d'une extension. Ces points ont été discutés dans [TER 01a, TER 01b]. La figure 3 présente un exemple de structure miroir dans laquelle :

– les descriptions du paradigme de modélisation mp_4 pour le langage C2 et de son métamodèle mm_4 (voir sections 4.2 et 4.3) sont illustrées. Deux flèches grises en trait gras représentent les instanciations, par la fonction $\mathcal{E}^{3,2}$, de gmp en mm_{UML} et de mp_4 en mm_4 ;

– les paradigmes de modélisation mp_3 et mp_7 correspondent respectivement aux propositions du projet TAU [KAK 96] et de Price *et al.* [PRI 99a] sur les modèles de temps (voir section 2.1). Le paradigme de modélisation mp_7 est subsumé par mp_3 et le métamodèle mm_7 est construit comme un héritier du métamodèle mm_3 ;

– le paradigme de modélisation mp_2 du projet UWE pour l'hypermedia (de Koch *et al.* [KOC 00]) décrit les principales caractéristiques d'un site web : espace de navigation, structure de navigation, etc. Dans leur proposition, l'espace de navigation décrit les classes (du système d'information sous-jacent) qui sont accessibles lors de la navigation ainsi que les associations permettant de naviguer d'une classe à l'autre. La structure de navigation décrit les types de navigation (tour guidé, index, etc.) offerts sur chaque lien navigable entre ces classes. Le paradigme de modélisation mp_2 est construit directement à partir de gmp. Le métamodèle correspondant, noté mm_2, est donc construit à partir de mm_{UML}. Divers stéréotypes sont définis comme par exemple les stéréotypes $\ll NavigationalClass \gg$, $\ll NavigationalLink \gg$, $\ll GuidedTour \gg$, etc. ;

– Fröhlich *et al.* [FRO 97] ont proposé un autre paradigme de modélisation pour l'hypermédia, noté mp_5. Ce paradigme de modélisation est similaire à celui du projet UWE mais il comprend plus de contraintes (explicites) : une restriction aux associations binaires entre classe de navigation, une limitation des types de navigation acceptés en fonction de la multiplicité des associations qui servent de support à la navigation, etc. Ce paradigme de modélisation mp_5 est donc subsumé par le paradigme mp_2 du projet UWE. La partie du métamodèle correspondant, noté mm_5, comprend une contrainte OCL pour garantir que les $\ll NavigationalLink \gg$ sont binaires, etc. ;

– Alatalo *et al.* [ALA 01] ont proposé un paradigme de modélisation pour l'hypermédia mobile. Nous notons par mp_6 ce paradigme de modélisation développé dans le cadre du projet OWLA. Le paradigme de modélisation mp_6 met en place un concept d'association conditionnelle afin de distinguer certains liens de navigations qui ne sont pas toujours « visibles » (dans certains cas, dépendant de la localisation de l'utilisateur, ces liens disparaissent de l'espace de navigation). Ce paradigme de modélisation mp_6 est lui aussi subsumé par le paradigme mp_2 du projet UWE. Le métamodèle correspondant mm_6 est construit à partir de mm_2. Il contient entre autres un stéréotype pour

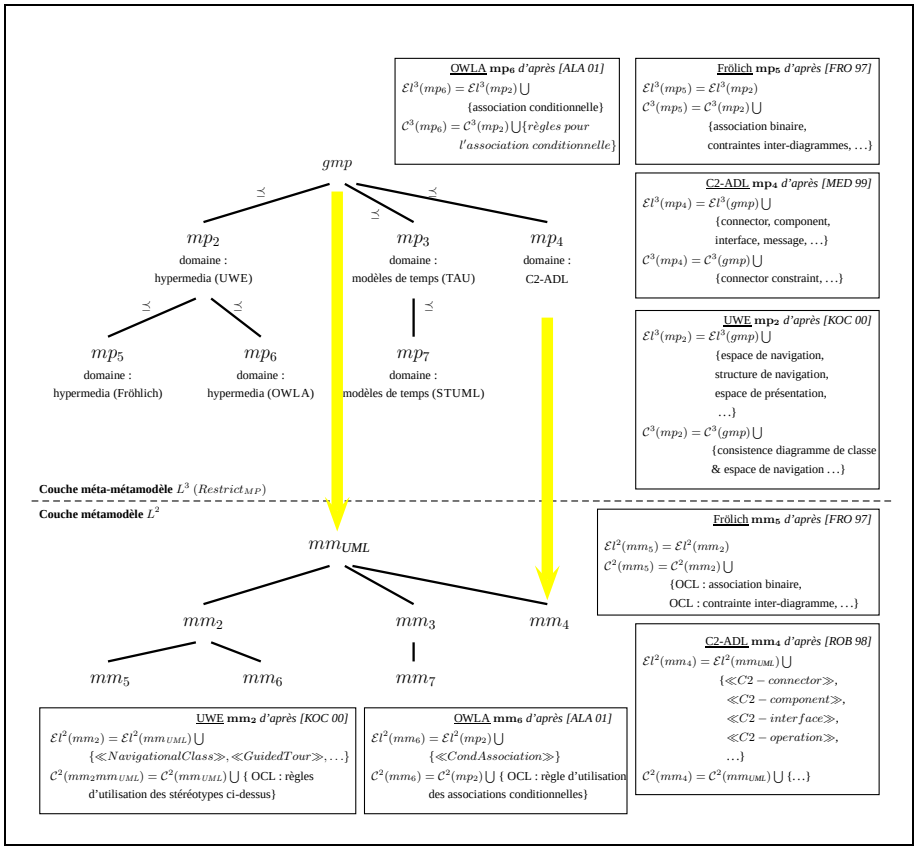

Figure 3. *Structure miroir des couches méta-métamodèle et métamodèle*

l'association conditionnelle qui est noté $\ll CondAssociation \gg$.

La structure miroir est ainsi mise en place en fonction des relations de subsomption entre paradigmes de modélisation. Ces dernières sont souvent données explicitement par les acteurs de l'ingénierie du logiciel qui décrivent leur paradigme de modélisation à partir d'un paradigme existant. Cependant un paradigme de modélisation peut-être décrit sans référence à un paradigme existant. La tâche de placement dans le poset des paradigmes de modélisation est alors plus délicate : il faut inventorier les concepts et les contraintes afin de déterminer les relations de subsomption existantes : cela suppose la prise en charge d'un mécanisme de preuve.

4.5. *Les enjeux*

En conclusion, cette structure miroir qui lie les paradigmes de modélisation (descriptions informelles et multinotations) aux métamodèles (descriptions formalisées

et unilangages) est un compromis entre la nécessité de proposer une description des domaines d'application qui soit lisible pour tous les utilisateurs et le besoin d'une organisation formellement exploitable des métamodèles. La capacité à maintenir un équilibre satisfaisant entre le degré de liberté laissé aux utilisateurs décrivant des paradigmes de modélisation et la qualité formelle de la couche métamodèle est un de nos premiers enjeux. Nous pensons que limiter – comme nous le faisons – les contraintes sur le niveau informel à une structuration en poset des descriptions (sans restreindre le ou les langages utilisés) est une base acceptable. Une telle liberté laissée pour l'expression des descriptions informelles introduit des besoins lourds en terme de mise en place de la structure miroir. Néanmoins, cette structure miroir présente – au-delà de son coût – l'avantage de permettre un travail *a posteriori* sur les résultats des opérations formelles sur les métamodèles. Dans la section suivante, nous discutons des besoins engendrés par une telle approche.

5. Conclusion : les besoins nouveaux

Nous allons, en conclusion, considérer deux types de besoins nouveaux liés à notre approche. Les besoins aval concernent les opérations nécessaires à l'exploitation de la couche métamodèle pour l'ingénierie des systèmes d'information. Les besoins amont concernent les outils nécessaires à la mise en place de la structure miroir.

5.1. *Les besoins aval*

Le mode de description choisi pour les métamodèles nous permet – une fois réglés les problèmes liés aux ambiguïtés des paradigmes de modélisation[8] – de définir des opérations formelles sur les métamodèles. Il s'avère que ces opérations même si elles sont formellement valides, ne sont pas toujours satisfaisantes en terme de sémantique de l'application. Par exemple, nous avons proposé une intégration formelle de métamodèles, notre objectif étant d'utiliser le métamodèle ainsi construit comme base d'agrément abstraite pour l'interopérabilité. Mais le métamodèle construit par intégration formelle peut être parfois trop éloigné d'un point de vue sémantique des systèmes d'information initiaux. Nous avons donc « étendu » l'intégration formelle en une intégration sémantique des métamodèles [TER 01b] et proposé une mesure de « distance sémantique » entre métamodèles [TER 01a]. En exploitant cette intégration sémantique des métamodèles, nous sommes à même de construire diverses plateformes pour l'ingénierie des systèmes d'information et du web sémantique [TER 02a, TER 02b].

L'approche MDA (Model Driven Architecture) a pour sa part fait apparaître de nombreux besoins en terme d'opérations sur les modèles et métamodèles, de gestion de bibliothèques de modèles et métamodèles, etc. Nous projetons d'étudier ces besoins dans le cadre de notre structure miroir.

8. Cette partie de notre travail est hors du champ de cet article, voir [TER 01a].

5.2. *Les besoins amont*

Mettre en place, dans des conditions satisfaisantes, la structure miroir des paradigmes de modélisation et des métamodèles suppose que nous puissions fournir un support pour l'analyse et l'exploitation des descriptifs de paradigme de modélisation, voire des descriptifs des mécanismes d'instanciation d'un paradigme de modélisation en extensions UML.

Ce support doit être susceptible de traiter des données multinotations (incluant le langage naturel) afin de retrouver les concepts (voire les contraintes) qui expriment le paradigme de modélisation. La complexité de cet aspect multinotation est cependant à mettre en balance avec le fait qu'il s'agit d'un langage technique et parfaitement délimité. Pour la composante langage naturel, il nous semble possible de partir vers une approche à base de tag (projets ATOLL ou XTAG [BAR 01, XTA]).

La plupart du temps, les concepts utilisés sont connus (soit dans les paradigmes de modélisation existants soit dans les descriptions du domaine d'application). Nous pouvons donc considérer qu'il existe un corpus quasi exhaustif à partir duquel nous pouvons construire un fond de ressources terminologiques. Nous en sommes à la phase d'étude de l'état de l'art [BOU 99] pour le choix de la forme à donner à ce fond de ressources.

De plus, il est possible de construire ce corpus de telle sorte que nous puissions à partir de termes clés déterminer des contextes d'utilisation. Nous pouvons donc utiliser l'approche proposée par Crispino *et al.* [CRI 99] pour la plateforme d'ingénierie linguistique Context.

5.3. *Conclusion*

Jusqu'à présent, nous avons travaillé à la mise en place de la structure miroir et à sa validation par rapport aux travaux existants. Nous considérons que cette phase de travail est terminée. Nous avons mis en place les outils formels pour la couche métamodèle et travaillé à l'extension sémantique de certains de ces outils. Ce mécanisme d'extension sémantique doit être à son tour validé sur des travaux existants. A l'issue de cette validation, une partie non négligeable des besoins aval sera couverte.

Notre principal chantier ouvert concerne maintenant les besoins amont, *i.e.*, la mise en place d'une plateforme pour l'ajout de nouvelles descriptions (paradigmes de modélisation et métamodèles) dans notre structure miroir : une telle plateforme doit prendre en compte – de façon efficace – des descriptions multinotations.

6. Bibliographie

[ALA 01] ALATALO T., PERÄAHO J., « Designing Mobile-aware Adaptive Hypermedia », *Proceedings of the Third Workshop on Adaptive Hypertext and Hypermedia*, 2001.

[AMM 99] AMMANN P., BLACK P. E., « Abstracting Formal Specifications to Generate Software Tests via Model Checking », *Proceedings of the 18th Digital Avionics Systems Conference, DASC'99, Saint Louis, Missouri, USA*, 1999, Available at URL www.nist.gov.

[AND 99] ANDROUTSOPOULOS K., « The Reactive System Development Support Tool », rapport, 1999, King's College, Department of Computing, Available at URL http ://www.dcs.kcl.ac.uk/pg/kelly.

[BAC 98] BACK R., BUTLER M., « Fusion and Simultaneous Execution in the Refinement Calculus », *Acta Informatica*, vol. 35, 1998, p. 921-949, IEEE.

[BAR 01] BARTHÉLEMY F., BOUILLER P., DESCHAMP P., KAOUANE L., DE LA CLERGERIE E. V., « Atelier ATOLL pour les grammaires d'arbres adjoints », *Actes de la conférence TALN'01, Tours, France*, 2001.

[BEZ 98] BEZIVIN J., « On Different Interoperability Modes in Software Engineering : the Case of Modeling Activities at OMG », *Proceedings of Software Engineering'98*, Paris, France, December 1998.

[BHA 99] BHARADWAJ R., HEITMEYER C., « Hardware/Software Co-Design and Co-Validation Using the SCR Method », *Proceedings of the International High Level Design Validation and Test Workshop, HLDVT'99*, November 1999, Available at URL chacs.nrl.navy.mil/SCR.

[BHA 00] BHARADWAJ R., HEITMEYER C., « Developing High Assurance Avionics Systems with the SCR Requirements Method », *Proceedings of the 19th Digital Avionics Systems Conference, USA*, October 2000, Available at URL chacs.nrl.navy.mil/SCR.

[BOU 99] BOURIGAULT D., SLODZIAN M., « Pour une terminologie textuelle », *Les cahiers du Rifal*, n° 19, 1999.

[BRE 98] BREU R., GROSU R., HUBER F., RUMPE B., SCHWERIN W., « Systems, Views and Models of UML », SCHADER M., KORTHAUS A., Eds., *The Unified Modeling Language, Technical Aspects and Applications*, Physica Verlag, 1998, p. 93-109, Available at URL http ://www.cs.york.ac.uk/puml.

[CLA 96] CLARKE E. M., WING J. M., « Formal Methods : State of Art and Future Directions », *ACM Computing Surveys*, 28(4), p. 626-643, ACM, December 1996.

[CLA 01] CLARK T., EVANS A., KENT S., SAMMUT P., « The MMF Approach to Engineering Object-Oriented Design Languages », *Proceedings of the Workshop on language, descriptions, Tools and Applications, LDTA01*, 2001.

[COO 99] COOK S., KLEPPE A., MITCHELL R., RUMPE B., WARMER J., WILLS A. C., « Defining UML Family Members Using Prefaces », MINGINS C., MEYER B., Eds., *Proceedings of "Technology of Object-Oriented Languages and Systems", TOOLS 32*, IEEE, November 1999, p. 102-114.

[CRI 99] CRISPINO G., HAZEZ S., MINEL J., « Architecture logicielle de Context plate-forme d'ingéniérie linguistique », *Actes de la conférence TALN'99, Cargèse, France*, 1999.

[DAS 00] DASKALOPULU A., « Model Checking Contractual Protocols », *Proceedings of the 13th Annual Conference JURIX'2000*, 2000, p. 35-47.

[DEM 97] DEMIGUEL M., ALONSO A., DE LA PUENTE J., « Object-Oriented Design of Real-Time Systems with Stereotypes », *Proceedings of the 9^{th} Euromicro Workshop on Real-Time Systems*, IEEE, June 1997, p. 216-223.

[EVA 98] EVANS A., BRUEL J.-M., FRANCE R., LANO K., RUMPE B., « Making UML Precise », *OOPSLA'98 Workshop on "Formalizing UML. Why and How ?"*, Vancouver, Canada, October 1998, OOPSLA'98, Available at URL http ://www4.informatik.tu-muenchen.de/papers.

[FAL 94] FALKENBERG E. D., HAN OEI J., « Meta Model Hierarchies from an Object-Role Modeling Perspective », *Proceedings of the 1^{st} International Conference on Object-Role Modeling, ORM-1, Magnetic Island, Australia*, 1994.

[FRO 97] FROHLICH P., HENZE N., NEJDL W., « Meta-Modeling for Hypermedia Design », *Proceedings of the Second IEEE Metadata Conference, MD97*, 1997.

[GRÄ 71] GRÄTZER G., *Lattice Theory, First Concepts and Distributive Lattices*, W.H. Freeman, 1971, ISBN 0-7167-0442-0.

[HEI 98a] HEITMEYER C., « On the Need for *Practical* Formal Methods », *Proceedings of the 5^{th} International Symposium on Formal Techniques in Real-Time and Real-Time Fault-Tolerant Systems, FTRTFT'98, Lyngby, Denmark*, September 1998, p. 18-26, Invited paper, available at URL chacs.nrl.navy.mil/SCR.

[HEI 98b] HEITMEYER C., « SCR : A PRACTICAL METHOD FOR REQUIREMENTS SPECIFICATION », *Proceedings of the 17^{th} AIAA/IEEE/SAE Conference on Digital Avionics System, DASC'98*, vol. 1, IEEE, Oct.-Nov. 1998, p. C44/1-C44/5.

[HEI 98c] HEITMEYER C., « Using the SCR Toolset to Specify Software Requirements », *Proceedings of the 2^{nd} IEEE Workshop on Industrial Strength Formal Specification Techniques, WIFT'98*, Boca Raton, FL, USA, October 1998, IEEE, p. 12-13.

[HEI 98d] HEITMEYER C., KIRBY J., LABAW B., BHARADWAJ R., « SCR : A Toolset for Specifying and Analyzing Software Requirements », *Proceedings of the 10^{th} Annual Conference on Computer-Aided Verification, CAV'98*, Vancouver, Canada, 1998, p. 526-531, Available at URL chacs.nrl.navy.mil/SCR.

[HEI 02] HEITMEYER C., « *Software Cost Reduction* », 2002, ISBN 0-471-02895-9.

[HER 00] HERRERO J. L., SANCHEZ F., LUCIO F., BONILLA M. T., « Changing UML Metamodel in Order to Represent Concern Separation », ECOOP'00 Workshop 14 on Defining a Precise Semantics for UML, Sophia Antipolis, France, June 2000.

[HÖP 01] HÖPLER R., MOSTERMAN P. J., « Model Integrated Computing in Robot Control to Synthesize Real-time Embedded Code », *Proceedings of the International Conference on Control Applications, CCA'01, Mexico*, IEEE, 2001, Abstract available at URL www.robotic.dlr.de.

[JEF 01] JEFFORDS R. D., HEITMEYER C., « An Algorithm for Strengthening State Invariants Generated from Requirements Specifications », *Proceedings of the 5^{th} International Symposium on Requirements Engineering, RE'01*, Toronto, Canada, August 2001, IEEE.

[KAK 96] KAKOUDAKIS I., « The TAU Temporal Object Model », MPhil Thesis, UMIST, Manchester, UK, 1996.

[KAR 00] KARSAI G., NORDSTROM G., LEDECZI A., SZTIPANOVITS J., « Towards Two-Level Formal Modeling of Computer-Based Systems », *Journal of Universal Computer Science*, vol. 6(11), 2000.

[KOC 00] KOCH N., BAUMEISTER H., HENNICKER R., MANDEL L., « Extending UML for Modeling Navigation and Presentation in Web Applications », *Proceedings of the Workshop Modeling Web Applications in the UML, UML'00*, 2000.

[LED 01] LEDECZI A., MAROTI M., KARSAI G., GARRETT J., THOMASON C., NORDSTROM G., SPRINKLE J., VOLGYESI P., « The Generic Modeling Environment », *Proceedings of the WISP'2001 Conference*, 2001.

[LEG 01] LEGEARD B., PEUREUX F., « Automatic Generation of Functional Test Sequences from B Formal Specifcations - Presentation and Industrial Case Study », *Proccedings of the 16th IEEE Conference on Automated Software Engineering, ASE'2001*, 2001.

[LEO 02] LEONARD E., HEITMEYER C., « *Program Synthesis from Formal Requirements Specifications using APTS* », Kluwer Academic Publishers, 2002.

[MAT 93] MATSUOKA S., TAURA K., YONEZAWA A., « Highly Effcient and Encapsulated Re-use of Synchronization Code in Concurrent Object-Oriented Languages », *Proceedings OOPSLA'93 Workshop on Object-Oriented Behavioral Semantics*, ACM, 1993, p. 109-126.

[MED 96] MEDVIDOVIC N., TAYLOR R. N., E. JAMES WHITEHEAD J., « Formal Modeling of Software Architectures at Multiple Levels of Abstraction », *Proceedings of the California Software Symposium 1996*, 1996, p. 28-40.

[MED 99] MEDVIDOVIC N., ROSENBLUM D. S., « Assessing the Suitability of a Standard Design Method for Modeling Software Architectures », *Proceedings of the 1st IFIP Working Conference on Software Architecture, San Antonio, Texas, USA*, 1999, p. 161-182.

[NOR 99a] NORDSTROM G., « Metamodeling-Rapid Design and Evolution of Domain Specific Modeling Environments », PhD thesis, Graduate School of Vanderbilt University, Tennessee, USA, 1999.

[NOR 99b] NORDSTROM G., SZTIPANOVITS J., KARSAI G., LEDECZI A., « Metamodeling-Rapid Design and Evolution of Domain-Specific Modeling Environments », *Proceedings of the IEEE Conference and Worshop on Engineering of Computer-Based Systems, ECBS'99*, 1999, p. 68–74, Available at URL computer.org/proceedings/ecbs/0028.

[OBE 00] OBER I., « Difficulties in Defining Precise Semantics for UML », ECOOP'00 Workshop 14 on Defining a Precise Semantics for UML, Sophia Antipolis, France, June 2000.

[OMG 98] « Roadmap for the Business Object Initiative : Supporting Enterprise Distributed Computing, OMG Report 98-10-09 », Available at URL http ://www.omg.org.

[OMG 99] « A UML Profile for CORBA, OMG Report 99-08-02 », 1999, Available at URL http ://www.omg.org, Version 1.0, August 2, 1999.

[PRI 99a] PRICE R., RAMAMOHANARAO K., SRINIVASAN B., « Spatiotemporal Extensions to Unified Modeling Language », *Proceedings of the 10 th International Worshop on Database and Expert Systems Applications*, IEEE, September 1999, p. 460-461.

[PRI 99b] PRICE R., SRINIVASAN B., RAMAMOHANARAO K., « Extending the Unified Modeling Language to Support Spatiotemporal Applications », MINGINS C., MEYER B., Eds., *Proceedings of TOOLS 32, Conference on Technology of Object-Oriented Languages and Systems*, IEEE, November 1999, p. 163-174.

[ROB 98] ROBBINS J. E., MEDVIDOVIC N., REDMILES D. F., ROSENBLUM D. S., « Integrating Architecture Description Languages with a Standard Design Method », *Proceedings of the 1998 International Conference on Software Engineering*, IEEE, April 1998, p. 209-218.

[SVI 97] SVINTERIKOU M., THEODOULIDIS B., « The Temporal Unified Modelling Language TUML », Technical Report n° TR-97-1, 1997, TimeLab, Department of Computation, UMIST, UK.

[TAU 97] « TAU project », rapport, 1997, Timelab, UMIST, UK, URL http ://www.co.umist.ac.uk/ timelab/projects/tau.html.

[TER 01a] TERRASSE M.-N., « A Metamodeling Approach to Evolution », BALSTERS H., DE BRUCK B., CONRAD S., Eds., *Database Schema Evolution and Meta-Modeling*, Springer-Verlag, LNCS 2065, ISBN 3-540-42272-2, 2001, 9^{th} International Workshop on Foundations of Models and Languages for Data and Objects, Schloss Dagstuhl, Germany, September 2000.

[TER 01b] TERRASSE M.-N., SAVONNET M., BECKER G., « An UML-metamodeling Architecture for Interoperability of Information Systems », *Proceedings of the International Conference on Information Systems Modelling, ISM'01*, 2001, Available at URL http ://www.fee.vutbr.cz/UIVT/ism/ISM_programElEdition.htm.

[TER 02a] TERRASSE M.-N., BECKER G., SAVONNET M., « *Metamodelisation and Interoperability of Web-based Information Systems* », chapitre 1, Idea Group Publishing, USA, A. Dahanayake and W. Gerhardt édition, 2002, ISBN 1-59140-041-4.

[TER 02b] TERRASSE M.-N., SAVONNET M., BECKER G., LECLERCQ E., « A UML-Based Meta-modeling Architecture with Example Frameworks », WISME 2002, Workshop on Software Model Engineering, Dresden, Germany, Available at URL http ://www.metamodel.com/wisme-2002/terrasse.pdf, 2002.

[UML 00] « OMG Unified Modeling Language Specification », March 2000, Version 1.3, Available at URL http ://www.omg.org.

[XTA] « The XTAG Project », Available at URL www.cis.upenn.edu.