CONSTRUCTION DE SPECIFICATIONS MULTI-VUES UML ET B

Dieu Donné Okalas Ossami,
Doctorant en Informatique
okalas@loria.fr +33 3.83.59.20.16

Jeanine Souquières,
Professeur en Informatique - Université Nancy 2
Jeanine.Souquieres@loria.fr +33 3.83.59.20.12

Jean-Pierre Jacquot,
Maître de conférences en Informatique – Université Nancy 1
Jean-Pierre.Jacquot@loria.fr + 33 3.83.59.20.10

Adresse professionnelle
LORIA campus scientifique ★ BP 239 ★ F-54506 Vandœuvre-Lès-Nancy Cedex

Résumé : On sait aujourd'hui transformer les diagrammes UML et les expressions OCL en B, mais pas le contraire. Le manque de retour de B vers UML laisse penser que l'évolution individuelle des deux représentations pourrait conduire à ce qu'elles expriment des exigences contradictoires. Par ailleurs, le fait que UML et B appartiennent à deux paradigmes de modélisation différents fait que la transformation systématique de B en UML ne peut se faire sans perte d'informations. Pour surmonter ces problèmes, nous proposons de structurer la spécification en vues: une vue UML et une vue B. Dans une telle approche, l'utilisateur ne travaille plus sur deux spécifications indépendantes, mais sur une des deux représentations d'une même spécification. La structuration en vues, permettra au concepteur de faire usage du meilleur des deux: clarté architecturale pour UML, pouvoir d'expression et outils de preuve pour B. Ce papier présente les concepts et la démarche générale de construction de spécifications multi-vues UML et B.

Summary: UML diagrams and OCL expressions can be transform into B, but not the opposite. The lack of return from B towards UML lets think that the individual evolution of both representations could lead to express contradictory requirements. Moreover as UML and B belong to two different modeling paradigms, the systematic transformation of B into UML cannot be done without loosing information. To overcome these problems, we propose to structure the specification into views: A UML view and a B view. In such an approach, the user does not work any more on two independent specifications, but on one of the two representations of the same specification. By structuring into views, the specifier is able to use the best of both formalisms: architectural clearness for UML, expression power and proof tools for B. This paper presents the concepts to construct UML and B multi-views specifications.

Mots clés : B, UML, intégration UML et B, spécifications multi-vues, méthodes formelles.

Keywords : B, UML, UML and B integration, multi-views specifications, formal methods.

Construction de spécifications multi-vues UML et B

1 - MOTIVATIONS

Le développement du logiciel devient de plus en plus complexe. Cette complexité est due à la diversité des contraintes (temps réel, sûreté, etc.) à prendre en compte, à l'hétérogénéité des données manipulées, à la variété des fonctionnalités (flexibilité, maintenabilité, interopérabilité, etc.). Pour prendre en compte tous ces facteurs, la construction d'un logiciel est souvent l'œuvre de plusieurs spécialistes (analystes, programmeurs, testeurs, décideurs, etc.) qui interagissent à travers divers documents (cahier des charges, spécifications, documents de conception, plans de tests, programmes, etc.). Ces documents définissent les différentes facettes du futur logiciel et sont rédigés pour partie en langue naturelle et pour partie dans des langages artificiels dont la syntaxe et la sémantique sont définies formellement, comme par exemple les formalismes UML, B, Z, etc.

De manière générale, la construction de spécifications occupe une place importante dans le cycle de développement du logiciel. Dans la pratique industrielle, la construction des documents de spécification fait souvent appel à des notations semi-formelles UML (diagrammes de classes, diagrammes d'état-transition, diagrammes de cas d'utilisation, etc.). Ces notations combinent une structure à base de graphismes avec des contraintes OCL et des annotations en langue naturelle sous forme de commentaires. La rapidité de conception, la facilité de lecture et la clarté architecturale des modèles UML en font une méthode populaire et très utilisée. La difficulté liée à la conception de modèles UML fiables réside dans l'immaturité ou le manque d'outils de validation mathématique des propriétés modélisées. Conscients de cet handicap, les industriels sont sensibilisés à la nécessité de mettre au point et d'intégrer les méthodes formelles comme éléments à part entière du cycle de développement de logiciels fiables. L'expérience a montré qu'une erreur de spécification aussi infime soit elle détectée en cours d'exploitation, peut non seulement entraîner des surcoûts de correction considérables, mais aussi provoquer des dégâts dont les conséquences sont parfois irréversibles sinon dramatiques comme la perte de vie humaines.

Les méthodes formelles sont basées sur une approche rigoureuse et répondent au double objectif non seulement de qualité et de sûreté de fonctionnement, mais aussi de faciliter le contrôle de conformité de logiciels avec leurs spécifications. Elles possèdent une notation avec une syntaxe et une sémantique précises et sont souvent supportées par des outils de preuves automatiques ou interactives. Mais le fait que leur notation soit à base de concepts mathématiques et de la logique a conduit à ce que les spécifications écrites avec ces notations sont difficiles à lire et à comprendre quand elles ne sont pas accompagnées d'une bonne documentation. Dans ces conditions, leur utilisation pour construire des systèmes de grande taille est coûteuse en temps et n'est réservée qu'aux seuls initiés. C'est à ce titre que de nombreuses recherches ont été menées sur le couplage d'UML et les méthodes formelles comme Z ou B. Le but de ces recherches est d'étudier des mécanismes de modélisation automatisables capables de combiner la rapidité de conception, la facilité de lecture et la clarté architecturale des modèles UML d'une part, avec la fiabilité, le pouvoir d'expression et la sémantique des méthodes formelles d'autre part. Ces travaux ont permis de mettre au point un certain nombre d'outils de génération de spécifications B [Ledang 2003, Snook 2002, Laleau 2000], Z [RoZ], etc. à partir de diagrammes UML.

Ces outils permettent de transformer les diagrammes UML et certaines expressions OCL en B, mais pas le contraire. Le manque de retour de B, Z, etc. vers UML laisse penser que l'évolution individuelle des deux représentations pourrait conduire à ce qu'elles expriment des exigences contradictoires. Par ailleurs, la qualité d'un outil d'aide au développement, ne dépend pas seulement de sa capacité à projeter un langage source en un langage cible, mais aussi de sa faculté à analyser la pertinence des constructions induites et à fournir une assistance à

l'utilisateur non spécialiste du langage. C'est dans ce cadre que nous étudions une nouvelle démarche d'intégration d'UML et B basée sur la structuration en vues. Dans une telle approche, l'utilisateur ne travaille plus sur deux spécifications indépendantes, mais sur une des deux représentations d'une même spécification. La structuration en vues, permet au concepteur de faire usage du meilleur des deux: clarté architecturale pour UML, pouvoir d'expression et outils de preuve pour B.

L'objectif de ce papier est double. D'une part, il s'agit de dresser une analyse sur le processus d'intégration d'UML et B, c'est à dire la traduction systématique de diagrammes UML en B. D'autre part, il s'agit d'apporter en s'appuyant sur cette analyse, des éléments de réflexion sur notre nouvelle façon de penser l'intégration de ces deux formalismes (la décomposition de la spécification en vues UML en B). Pour ce dernier point, notre but est de montrer l'intérêt et la nécessité d'une telle démarche. Le travail présenté dans le papier décrit la démarche générale de construction de spécifications multi-vues UML et B.

Le reste du papier est structuré comme suit. Dans le paragraphe 2 nous présentons brièvement les formalismes UML et B. Le paragraphe 3 présente les avantages et les limites de cette démarche. Le paragraphe 4 présente l'idée générale de notre approche. Le paragraphe 5 est dédié à la présentation des concepts importants supportés par notre approche de construction de spécifications multi-vues UML et B. Finalement, nous concluons et donnons des suites à ce travail dans le paragraphe 6.

2 – BREVE PRESENTATION D'UML ET B

2.1 - UML

UML [OMG 2001, Rumbaugh 1991, Rumbaugh 1998] est une technique de modélisation standardisée et très répandue dans l'industrie du logiciel. Elle propose un ensemble de notations graphiques de modélisation par objets. Ces notations permettent de visualiser, de construire et de documenter les systèmes à l'aide de diagrammes. UML possède neuf types de diagrammes représentant chacun un aspect particulier du système (aspects fonctionnels, statiques, dynamiques).

Parmi les différents types de diagrammes UML, nous nous intéressons plus particulièrement aux diagrammes de classes et d'état-transition. Ces deux types de diagrammes se sont montrés suffisants pour représenter la structure et la dynamique d'un système. Dans ce papier nos efforts se focalisent sur le diagramme de classes.

2.1.1 – Eléments du diagramme de classes UML

Un diagramme de classes UML est constitué d'un ensemble de classes reliées entre elles par des associations et des hiérarchies de généralisation (cf. Fig. 1). L'association modélise comment sont liés les objets des classes participantes. Elle est caractérisée par un nom et deux ou plusieurs extrémités sur lesquelles figurent les cardinalités et les noms de rôle. Il existe deux cas particuliers pour les associations: la composition et l'agrégation. La généralisation est une relation dans laquelle une des classes impliquées est identifiée comme classe générale et les autres comme des spécialisations de celle-ci.

L'élément central dans la modélisation orientée objet avec UML est la classe. Une classe est en général représentée par un rectangle divisé en trois compartiments. Le compartiment du haut spécifie le nom de la classe et les compartiments du centre et du bas définissent respectivement la liste des attributs et les opérations. Selon le niveau de détails souhaité, les compartiments du centre et du bas peuvent être omis.

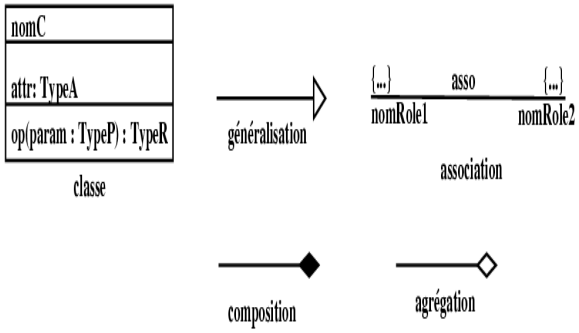

Fig. 1 : Eléments structuraux du diagramme de classes UML

L'ensemble des diagrammes UML servant à décrire les différents aspects d'un système forment le modèle de celui-ci (dans notre cas, diagramme de classes, d'état-transition). Chaque diagramme constitue une des représentations partielles (vue) du modèle UML.

2.2 – La méthode B

La méthode B [Abrial 1996] est une méthode de spécification formelle qui comprend toutes les phases de développement logiciel, de la spécification jusqu'à l'implémentation. Elle permet d'exprimer des propriétés sous la forme de prédicats de la logique de premier ordre. Les opérateurs utilisés par la notation B sont ceux de la théorie des ensembles de Zermelo-Frankel. Les propriétés d'un système à prouver en B sont essentiellement des propriétés d'invariance. Contrairement à UML, la notation B n'est pas orientée objets.

Structure d'une machine B

L'élément de granularité de base d'une spécification B est la machine abstraite. La figure 2 présente la structure syntaxique d'une machine abstraite B. Cette notion est similaire au concept de classe ou de module dans les langages de programmation classiques. La machine abstraite peut être successivement raffinée (*raffinement*) jusqu'à l'obtention d'un composant implantable (*implantation*).

Le concept central étant l'encapsulation, le changement d'état d'une machine ne peut se faire qu'au travers des opérations. Une machine B est structurée en trois parties principales: l'entête, la partie déclarative et la partie opérationnelle. L'entête spécifie le nom et la liste des paramètres éventuels de la machine. Elle inclut la partie composition de la machine qui permet de décrire les différents liens entre les machines. Ces liens se traduisent par la déclaration de noms de machines dans les clauses *SEES, USES, EXTENDS, IMPORTS* et *INCLUDES*. Chaque clause possède une sémantique et des règles de visibilité précises. La partie déclarative modélise l'état de la machine au travers de divers types de données (variables, constantes, ensembles) et de contraintes (cf. Fig. 2, *C, P, I, etc.*) que ces données doivent toujours vérifier. La partie opérationnelle est constituée de l'initialisation et des opérations. Elle est basée sur le langage des substitutions généralisées. Une substitution généralisée est une construction mathématique abstraite construite à partir d'une substitution basique ***x := e***, qui correspond à un changement d'état, par l'intermédiaire d'opérateurs comme CHOICE S_1 OR S_2, PRE *P* THEN *S* , etc.

MACHINE
Ma(p) /*nom et liste des paramètres de la machine */
CONSTRAINTS
C(p) /*contraintes sur les paramètres*/
CONSTANTS
c /*liste des constantes */
SETS
/*ensembles différés ou génériques*/
s ; s'={a,b,c,...}
PROPERTIES
/*propriétés sur les constantes et les ensembles*/
P(c, s, s')
VARIABLES
v /*liste des variables*/
INVARIANT
I(p, c, s, s', v) /*invariant*/
INITIALISATION
Init /*initialisation*/
OPERATIONS
/*liste des operations */
Op = **PRE** *pre* **THEN** *S* **END**
...
END

Fig. 2 : Structure d'une machine abstraite B

Dans la figure 2, les clauses *CONSTRAINTS*, *PROPERTIES, INVARIANT*, etc. regroupent un ensemble de prédicats (*C, P, I, etc.*). Le prédicat minimal est le prédicat de typage qui permet de déterminer le type (le domaine de valeurs) de chacune des données de la machine. *X(j)* signifie que la donnée *j* apparaît dans le prédicat *X*. Les prédicats *C*, *P* et *I*, *Init*, *pre* et *S* doivent être écrits de telle sorte qu'ils garantissent la consistance interne de la machine [Lano 96].

3 – INTEGRATION D'UML ET B : LIMITES ET AVANTAGES

3.1 – Avantages de l'intégration d'UML et B

L'intégration d'UML et B est une technique qui consiste à représenter les connaissances d'un domaine d'application à l'aide de diagrammes UML augmenté de contraintes OCL [Warmer 1999] et de traduire ces diagrammes en B, puis de mettre en œuvre des raisonnements sur ces représentations. Les avantages d'une telle démarche sont nombreux:

- masquer la modélisation formelle B à l'utilisateur par la manipulation de graphismes,
- utiliser UML comme point de départ de la modélisation B des modèles orientés objets,
- valider les modèles UML à l'aide des outils de preuve (*atelierB*, *bToolKit*) de la méthode B,
- donner une sémantique aux modèles UM à l'aide de B.

L'utilisation des outils d'animation automatique de preuves permet d'explorer les aspects fonctionnels du système modélisé. En terme de modélisation UML, cela signifie que la dynamique du système peut être analysée en terme de vérification de pré- et post-condition d'opérations et d'invariants sur les données (attributs) et les relations entre les objets. Le fait que B définisse un mécanisme de raffinement permet non seulement un développement incrémental, mais aussi de maintenir les propriétés de processus tout au long du développement. Pour ce dernier point,

3.2 – Limites de l'intégration d'UML et B

Malgré les nombreux avantages cités dans le paragraphe 3.1, la démarche de génération de spécifications B à partir de diagrammes UML possède quelques limites:

- on sait transformer UML en B, mais pas le contraire. Les modifications opérées sur la spécification B dérivée ne sont pas prises en compte au niveau UML, ce qui pose le problème de la cohérence entre les deux représentations,
- le processus de développement induit est séquentiel (cf. Fig. 3): concevoir le modèle UML (étape 1), le traduire en B (étape 2), poursuivre le processus avec la complétion et/ou le raffinement des squelettes de spécifications B induites (étape 3), intégrer une activité de validation (étape 4). Un tel processus ne permet pas de garantir la consistance des textes B produits avec le modèle UML initial à cause du manque de liens dynamiques entre les deux représentations.

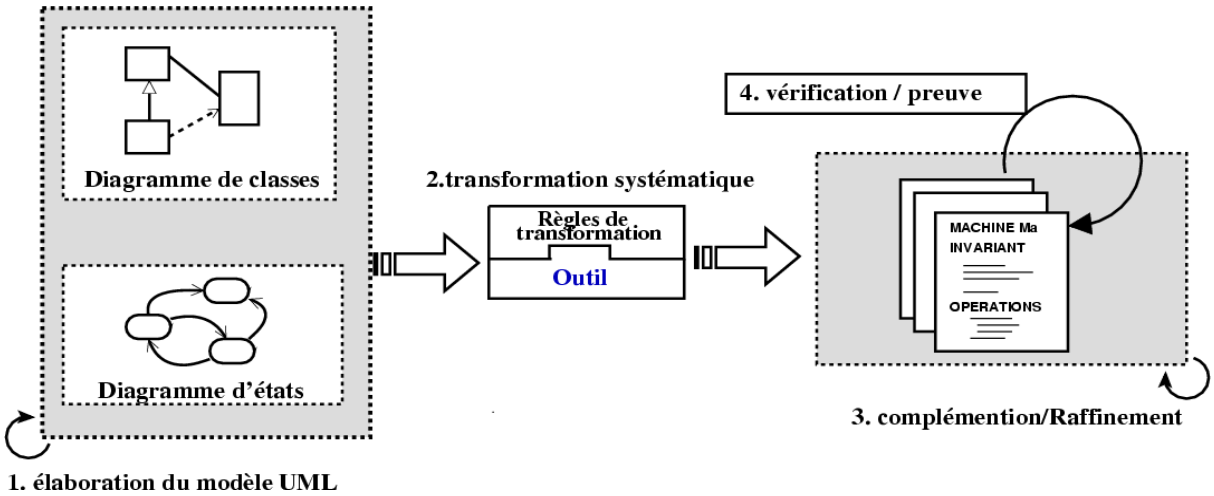

Fig. 3 : Processus de dérivation UML vers B

Il s'agit de s'assurer qu'il n'existe pas de contradictions entre les différentes abstractions du système.

La majorité des approches de couplage d'UML avec les méthodes formelles Z [Dupuy 2000, France 2000], B [Nguyen 1998, Meyer 2001] ou LOTOS [Wang 97] suivent le processus présenté dans la figure 3.

3.3 – Vers une structuration en vues UML et B

Pour surmonter les limites de la traduction d'UML en B, une idée consiste à poursuivre la construction de la spécification en B avec l'écriture des raffinements et en déduire la transformation inverse des textes B produits en diagrammes UML. Mais dans un développement à large échelle, cette solution peut vite s'avérer limitée pour une raison essentielle:

- la notation B contrairement à celle d'UML n'est pas orientée objets. Cette différence implique l'existence d'incompatibilités entre les deux formalismes.

La spécification B déduite automatiquement d'UML, est éloignée de celle qu'on aurait écrite directement en B ou en UML. Dans ces conditions, il est plus simple d'écrire directement les spécifications en B ou en UML.

Cette observation montre clairement que la transformation entre UML et B ne peut pas être un processus symétrique. Dans ce contexte, nous proposons de structurer la spécification en différentes vues (vue UML et vue B). Les transformations ne sont plus déduites systématiquement, mais dépendent de l'aspect décrit du contexte de modélisation. Dans une telle approche, le concepteur ne travaille plus sur deux spécifications indépendantes, mais sur une des deux représentations d'une même spécification. Nous appelons cette démarche : *construction de spécifications multi-vues UML et B*.

4 – NOTRE APPROCHE : CONSTRUCTION DE SPECIFICATIONS MULTI-VUES UML ET B

La *construction de spécifications multi-vues UML et B* est une extension des travaux de Ledang [Ledang 2002] et de Meyer [Meyer 2001] qui ont défini un cadre théorique et pratique à la dérivation des spécifications B à partir des diagrammes UML et d'annotations OCL. L'extension de ces travaux consiste à considérer UML et B comme deux vues partielles d'une même spécification et à étudier les mécanismes de consistance entre toutes ces différentes vues (diagrammes de classes, diagrammes d'état-transition et spécification B induite).

Pour construire cette spécification, l'utilisateur utilise les *``opérations de construction''* qui font évoluer la spécification sur les deux représentations (UML et B). La traçabilité et la cohérence entre les deux représentations sont assurées par des liens dynamiques existants entre les deux textes. Ces liens se traduisent par la transformation bidirectionnelle qui permet de ce fait une plus grande flexibilité et la possibilité de faire évoluer la spécification tout en maintenant la cohérence entre les deux vues.

Cette démarche pose les bases d'une nouvelle technique d'intégration d'UML et B. Elle constitue une réponse aux problèmes (incohérence, traçabilité, etc.) liés à l'évolution individuelle de différents documents de spécification d'un système. L'idée consiste à composer les étapes de *l'élaboration du modèle UML* et de la *complétion des spécifications B* (cf. Fig. 3) en une seule et unique étape: *la construction de la spécification*.

Le scénario d'usage envisagé est le suivant. Le spécifieur se positionne sur une vue et construit les représentations de cette vue en effectuant les opérations de spécifications sur les objets qu'il souhaite modéliser (classe, attribut, opération, variable, machine, raffinement, etc.). L'outil modifie alors systématiquement les deux représentations en fonction des informations saisies et des contraintes de propagation définies par les règles de transformation.

Nous pensons que la construction simultanée et multi-vues d'une spécification dans un développement conjoint est un facteur important permettant d'une part de faciliter l'appropriation d'un système, d'autre part, d'autoriser une validation directe par l'expert non spécialiste du langage, de la modélisation adoptée. Une telle dynamique permet de suivre pas à pas les raisonnements effectués afin de

valider la représentation adoptée. La figure 4 présente l'architecture générale de cette démarche.

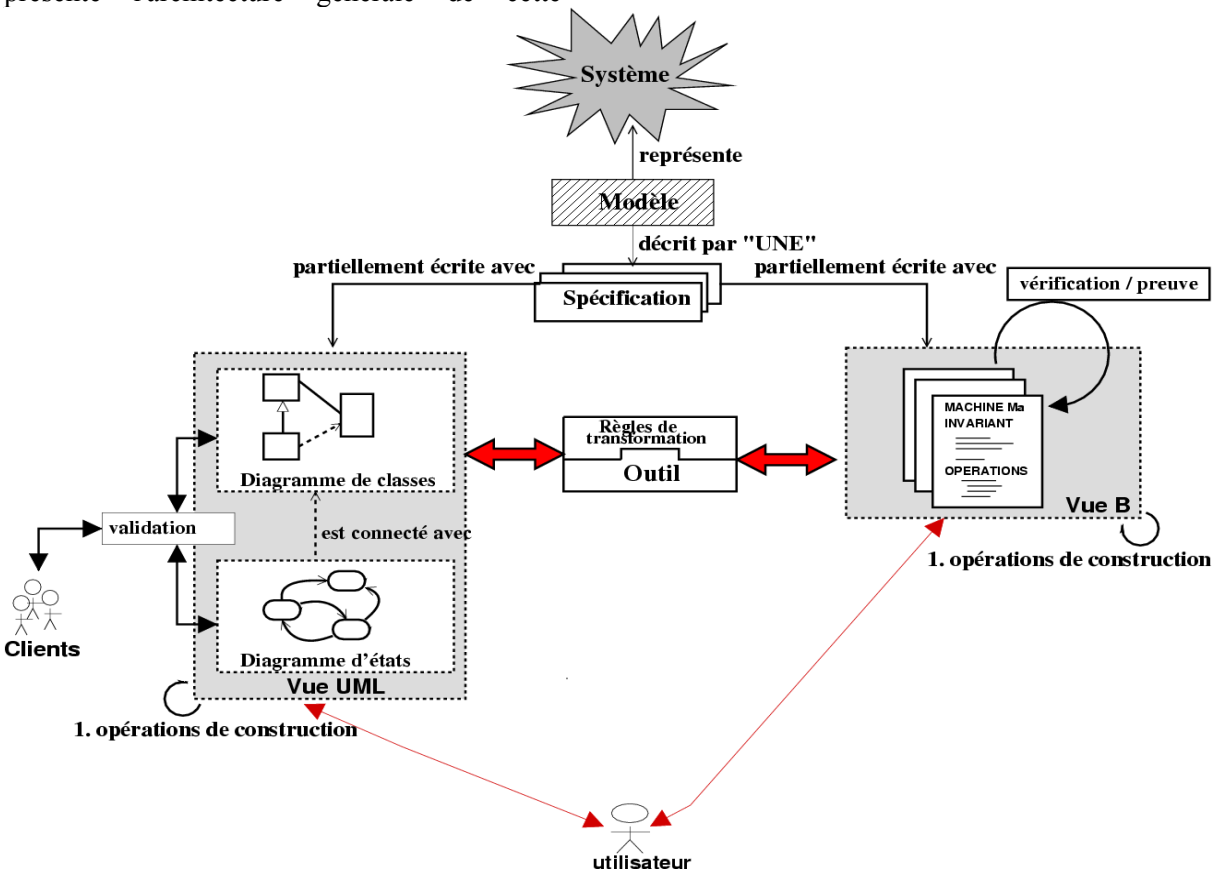

Fig. 4 : Assistance à la construction de spécifications multi-vues UML et B extrait de [Okalas 2003]

La construction de spécifications multi-vues UML et B vise à :

- utiliser UML et B de façon complémentaire,
- proposer un cadre de transformation et un environnement de développement interactif de spécifications par combinaison de plusieurs formalismes,
- faciliter la diffusion de B,
- documenter dynamiquement les projets B,
- apporter une réponse aux problèmes (incohérence, traçabilité, etc.) liés à l'évolution individuelle de différents documents de spécification d'un même système,
- concevoir un outil d'aide au développement de spécifications multi-vues UML et B.

La problématique générale à laquelle nous nous intéressons peut se traduire ainsi:

étant donné

1. un ensemble de langages (UML et B dans notre cas) définis chacun par une syntaxe et une sémantique données,
2. un ensemble de propriétés *Prop* d'un modèle *m* à construire,

déterminer

1. un ensemble d'opérations *O* qui permettent de décrire les propriétés *Prop* du modèle entièrement ou partiellement sur une ou sur les deux vues dans une dynamique de développement simultané UML et B,
2. un ensemble de règles permettant de calculer dynamiquement les correspondances entre les différentes représentations induites (i.e. UML et B) de la spécification,

3. un mécanisme de contrôle de cohérence par construction entre les différentes vues.

5 – CONCEPTS IMPORTANTS POUR LA CONSTRUCTION DE SPECIFICATIONS MULTI-VUES UML ET B

Ce paragraphe est consacré à l'ensemble des concepts qui doivent être supportés par la construction de spécifications multi-vues UML et B. Ces concepts sont relatifs aux notions de *vues*, *d'opérations de construction* et de *transformation*. Les concepts sont illustrés par des exemples. En parallèle, les constructions UML et B sont présentées.

5.1 – Les vues

La structuration en vues UML et B s'inscrit dans une tendance générale en matière de développement logiciel. Il s'agit de maîtriser la complexité de la spécification en décrivant certains aspects particuliers d'un système dans un langage dédié. Jackson et Zave [Jackson 1993] ont montré que l'utilisation de vues et de notations adaptées à chaque aspect de l'application pouvait clarifier la spécification. UML intègre la notion de décomposition en vues en offrant un éventail de diagrammes pour représenter différents aspects d'un même système.

La difficulté de la structuration en vues dans le cadre d'un développement conjoint semi-formel et formel (i.e. UML et B) réside dans l'utilisation de deux techniques de représentations différentes (graphique et textuelle) avec deux notations différentes appartenant à deux paradigmes de modélisation différents. Une telle combinaison est très intéressante. En effet, la décomposition d'un système en plusieurs vues ne peut-être réellement bénéfique que si les vues décrivent des concepts de nature différente. Le contraire serait une simple juxtaposition de notations.

Dans notre travail, une vue est considérée comme un moyen de construction et de visualisation des constructions partielles UML et B induites. Ces constructions sont réalisées au travers d'éléments dont la syntaxe et la sémantique sont définies par les méta-modèles respectifs des deux formalismes.

> **Définition** (Vue)
>
> Dans la construction de spécifications multi-vues UML et B, une vue v_i est considérée comme un tuple ***(M, m)***

Où

- v_i *:* := V_U | V_B désigne les vues UML et B,
- ***M*** est le méta-modèle (langage) dans lequel la description (modèle) partielle est écrite. Dans notre contexte, ***M*** désigne le méta-modèle UML ou le langage B. Chaque modèle ***m*** est construit avec les outils syntaxiques et sémantiques offerts par son méta-modèle respectif,
- $M ::= m_U \mid m_B$ désigne respectivement le modèle UML et la spécification B. m_U et m_B correspondent respectivement aux représentations partielles UML et B sur lesquelles l'utilisateur travaille.

Nous avons volontairement omis de présenter le détail de chacun des modèles m_U et m_B pour des raisons de lisibilité. Toutefois, nous les présenterons à chaque fois que c'est nécessaire.

Les éléments des modèles m_U et m_B entretiennent deux types de relations: les relations calculées et les simples correspondances. La figure 5 présente ces différents liens.

1. Les relations calculées (***T*** sur la figure 5) sont déduites automatiquement par l'application systématique des règles de transformation qui permettent de passer systématiquement d'UML à B et inversement,
2. Les simples correspondances (***C*** sur la figure 5) sont des relations non déductibles automatiquement. Par exemple, la post-condition d'une opération ne peut être déduite automatiquement. Par contre, elle doit faire référence à l'opération correspondante afin d'en assurer la traçabilité.

La figure 5 présente une classe *Customer* qui modélise le fichier client d'une société. Cette classe possède trois attributs *name*, *category* et *discount*. *name* caractérise le nom du client, *category* modélise le type de client (particulier, professionnel, etc.) et *discount* spécifie le pourcentage de réduction attribué à chaque catégorie de clients. Dans la suite, nous

utiliserons cette classe pour illustrer notre propos. L'exemple est petit, mais il est suffisant pour illustrer les idées présentées dans ce papier.

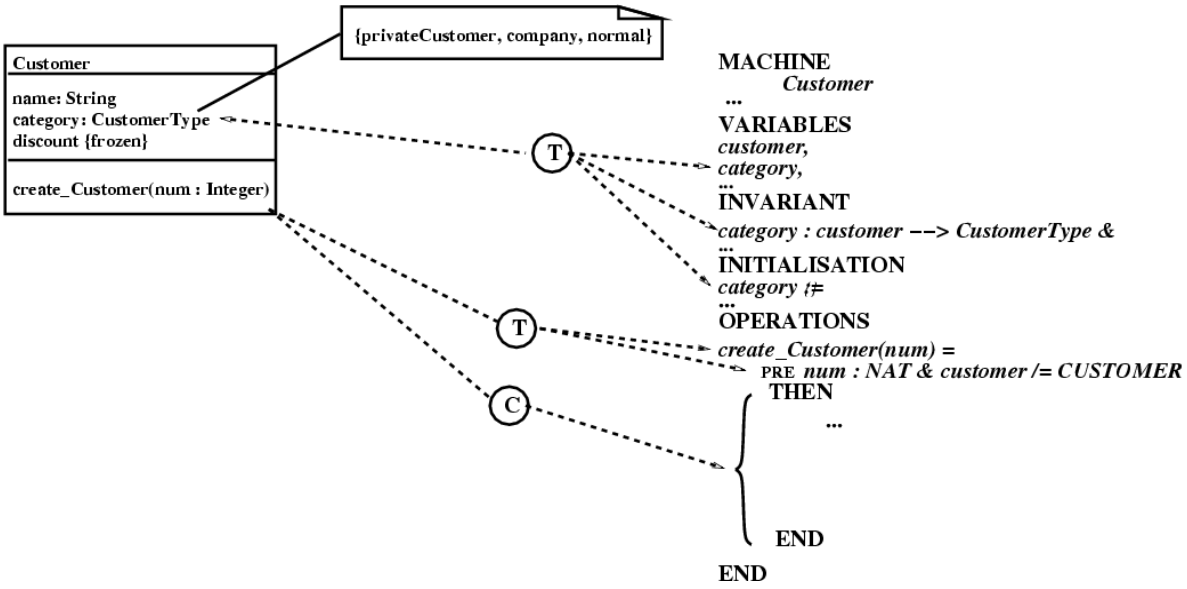

Fig. 5 : Exemple de relation entre m_U et m_B

5.2 – Opérations de construction

L'activité de construction d'une spécification peut être modélisée à travers un ensemble d'opérations atomiques: les *opérations de construction*. Ces opérations caractérisent la logique du développement de l'utilisateur. Un de nos objectifs est d'identifier et de caractériser ces opérations. En terme de modélisation, effectuer une opération de construction sur un ou plusieurs éléments d'un modèle m (m_U ou m_B dans notre cas) correspond à la modification de ce dernier. Il passe alors de m_{old} à m_{new} où m_{old} et m_{new} sont respectivement l'ancienne et la version modifiée de m. Dans un développement conjoint, cette modification est implicitement celle de toutes les représentations liées à l'élément édité. Dans cette perspective, la recherche sur la caractérisation des opérations de construction d'une spécification multi-vues peut se ramener à :

- un problème de calcul de la différence Δ entre deux versions d'un modèle m sur lequel l'utilisateur travaille:

  $\Delta = m_{new} \setminus m_{old}$

- l'étude de la propagation T de Δ sur les autres vues de la spécification

Pour traiter ces problèmes de façon générique, nous proposons de mettre en œuvre une stratégie de construction dont la première étape consiste à définir systématiquement tous les éléments (ou endroits) à créer, modifier ou supprimer sur la vue source pour accomplir la description du concept en cours d'édition. Dans le contexte de modélisation d'une variable en B par exemple, ce sera la clause VARIABLES pour la déclaration de la variable, la clause INVARIANT pour l'écriture de l'invariant relatif à la variable et la clause INITIALISATION pour l'initialisation de celle-ci. La deuxième étape repose sur la propagation des modifications sur la ou les vues cibles. Par exemple le diagramme de classes UML si la modification est effectuée en B. Il s'agit de déterminer systématiquement à chaque saisie de l'utilisateur, le type de ou des éléments UML ou B correspondant afin d'en déduire les représentations UML ou B adéquates. L'idée sous-jacente à la propagation réside dans le principe que l'exécution d'une opération de construction O_i sur une vue correspond à l'exécution systématique de zéro, une ou plusieurs opérations $O_0', \ldots, O_i'$ qui

ajoutent, suppriment ou modifient les éléments des différentes représentations de la spécification.

5.2.1 – Caractérisation de Δ

Dans [Alanen 2003] Alanen et Pores présentent le résultat Δ de la différence entre deux versions d'un modèle UML en terme d'opérations et non comme un ensemble d'éléments. Une des raisons de ce choix est que la différence entre deux modèles n'est pas toujours un modèle, tout comme le résultat de la différence entre deux entiers positifs n'est pas toujours un entier positif. Il est donc plus intuitif de représenter Δ en termes d'opérations qui ajoutent, suppriment ou modifient les éléments d'un modèle. Nous adoptons cette représentation pour caractériser les opérations de construction génériques qui permettent de construire simultanément les différentes représentations UML et B de la spécification. Dans notre étude, nous distinguons trois types d'opérations de construction de base qui définissent Δ:

1. les opérations de création, désignées par *new(e, t)* : création d'un élément *e* de type *t*, tel que:

$new(e, t) = [new(e, t) \mid e \in m_{new} \setminus m_{old}]$

2. les opérations de suppression, désignées par *delete(e,t)*: suppression d'un élément *e* de type *t*, tel que:

$delete(e, t) = [delete(e, t) \mid e \in m_{old} \setminus m_{new}]$

3. les opérations de modification, désignées par *modify(e,e')*: modification d'un élément *e* qui devient *e'*.

La modification d'un élément est implicitement celle du modèle, puisqu'un modèle est constitué d'éléments respectant certaines propriétés. Il existe plusieurs façons pour modifier un modèle. Nous en distinguons trois principales:

a. ajout, suppression ou modification *(add/delete/modify)* des données comme par exemple les attributs, les variables, les ensembles ou les constantes B.

b. ajout, modification ou suppression *(add/delete/modify)* des opérations des classes UML, des composants B (machines, raffinements, etc.).

c. ajout, modification ou suppression *(add/delete/modify)* des classes, association ou des composants B (machines, raffinements, etc.).

Sur la base de ces opérations de base, on peut définir Δ = *[new, delete, modify, O]* comme la collection d'ensembles d'opérations atomiques *add, remove, set, etc.* qui sont des sortes de raffinement de *new, delete, modify* et *O*. Les opérations *new, delete, modify* ajoutent, suppriment ou modifient un élément de façon globale. Tandis que *add, remove, set, etc*. ajoutent, modifient ou suppriment une caractéristique particulière d'un élément. *O* désigne toutes les autres opérations dont *skip* (opération sans effet).

Exemple (édition d'une variable B)

Du point de vue d'un utilisateur B, éditer une variable consiste à effectuer une saisie clavier pour la déclarer, écrire un invariant correspondant au domaine de valeurs qu'elle peut prendre et affecter une valeur initiale à celle-ci. Du point de vue de l'outil, ces banales saisies clavier sont interprétées comme une exécution de l'opération *modify* qui modifie la machine dans laquelle la nouvelle variable doit être ajoutée. Une telle opération peut se raffiner en trois opérations atomiques *add* permettant de construire les éléments nécessaires à la description complète d'une variable en B comme illustré dans la figure 6. Elle peut entraîner la génération de un ou plusieurs éléments UML (cf. Fig. 6). La déduction du type des ces derniers dépend de plusieurs facteurs (cet aspect n'est pas traité dans ce papier) comme par exemple l'invariant *I(v)* associé à *v,* les opérateurs (∈, =, >, ->, ≠, ≤, ≥, etc.) utilisés pour construire cet invariant, etc.

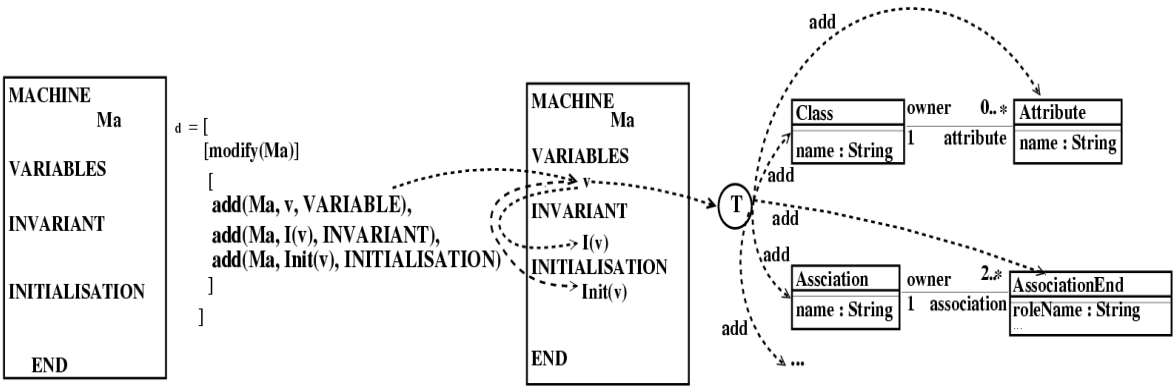

Fig. 6 : Ajout d'une variable dans une machine B et ses effets potentiels en UML.

Sur la figure 6 d = Δ

L'exécution d'une opération $O_i \in \Delta$ est conditionnée par le contexte de modélisation et les règles de construction relatives aux concepts des deux formalismes UML et B. Par exemple, une variable ou un attribut ne peut être ajouté à un composant (classe ou machine, etc.) si celui-ci contient déjà une donnée de même nom.

5.3 – La transformation

Pour être efficace, la construction de spécifications multi-vues UML et B a besoin de la transformation. Dans une dynamique de développement simultané, elle est induite par les opérations de construction. Le rôle de la transformation est double. D'une part, elle permet de calculer et de maintenir les liens dynamiques entre les deux représentations. Ces liens sont déduits à partir des règles de passage de B à UML et inversement. D'autre part, il s'agit de fournir un mécanisme de contrôle de consistance garantissant la qualité et la cohérence par construction des descriptions produites. Cette section est dédiée à la présentation des propriétés que doit avoir la transformation dans le cadre du développement de spécifications multi-vues UML et B.

La transformation est un processus de traduction automatique d'une ou plusieurs constructions d'un modèle m_i vers une ou plusieurs constructions d'un modèle m_j. Elle est modélisée par une collection de règles r_0 ,...,r_i qui associe un ou plusieurs éléments $e_i \in m_i$ avec un élément ou plusieurs éléments $e_j \in m_j$.

5.3.1 – Propriétés de la transformation

Bidirectionnalité:

C'est permettre un va et vient dynamique entre les différentes représentations afin de maintenir la cohérence et la traçabilité entre les éléments de ces représentations.

Consistance:

Le contrôle de consistance concerne deux points:

1. assurer que les constructions UML ou B induites sont respectivement valides par rapport aux deux langages,
2. assurer que la spécification induite reste toujours cohérente au regard des manipulations opérées sur tout aspect $\boldsymbol{e}_i$: les modifications opérées sur une vue $\boldsymbol{v_i}$ sont automatiquement propagées sur toutes les vues de la spécification: *cohérence par construction*.

Flexibilibilité:

Pour modéliser une donnée en B, l'utilisateur est parfois contraint de saisir certaines informations dont il ne dispose pas toujours.

Par exemple, une variable B *v* exige outre sa déclaration, un invariant (invariant de typage au minimum) et une valeur initiale. L'absence de l'une de ces informations induit une erreur. UML autorise dans ce cadre la modélisation partielle d'un attribut sans en préciser le type ou la valeur initiale. C'est par exemple le cas

des attributs dérivés qui sont des données calculées.

Dans ce cadre, un des objectifs de nos recherches est de proposer des mécanismes de construction capables de faire le lien entre la perception du développement UML de l'utilisateur et les exigences du développement formel B. Nous proposons dans ce contexte, de prédéfinir des mécanismes flexibles de construction permettant d'une part de conserver la facilité d'utilisation d'UML, et d'autre part de tenir compte de la rigueur de B. Prédéfinir dans ce contexte peut par exemple vouloir dire:

1. Prévoir des mécanismes de construction par défaut permettant de substituer aux valeurs manquantes des valeurs par défaut tout en prenant le soin de l'indiquer au spécifieur. Une telle substitution permet de ne pas contraindre l'utilisateur à spécifier des concepts non encore élucidés. Un tel mécanisme doit être capable de contrôler la pertinence des constructions déduites, d'en interdire ou de les compléter si besoin; l'utilisateur ne décrit pas nécessairement les informations implicites et/ou considérées comme évidentes (voir figures 7 et 8.

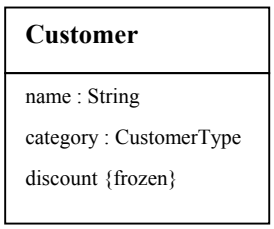

Fig. 7 Classe Customer

MACHINE

Customer

SETS

TypeCustomer; OBJECTS

CONSTANTS

discount, CUSTOMER

PROPERTIES

$CUSTOMER \subseteq OBJECTS \wedge discount \in \square \wedge \ldots$

VARIABLES

customer, category, name, …

INVARIANT

customer $\subseteq$ CUSTOMER $\wedge$

category $\in$ customer $\rightarrow$ TypeCustomer $\wedge$

name $\in$ customer $\rightarrow$ String $\wedge$ …

INITIALISATION

customer := $\square$ || *name* := *0* || *category* := $\square$

…

END

Fig. 8 Machine Customer

$\square$ désigne non seulement des valeurs par défaut, mais aussi les endroits que l'utilisateur devra expliciter lors d'une exploitation concrète de la spécification (utilisation du prouveur par exemple).

2. Prévoir des mécanismes de représentation partielle des aspects n'ayant pas de forme de représentation équivalente sur telle ou telle vue. Par exemple, l'invariant d'un attribut peut parfois être une formule complexe qui ne peut toujours pas être projetée en UML comme en témoigne la propriété de la constante *discount* sur la figure 9 où l'on souhaite attribuer une remise allant de 0 à cent pour cent à chaque catégorie de client. Le lien existant entre cette propriété et l'attribut *discount* (cf. Fig.7) est une simple correspondance.

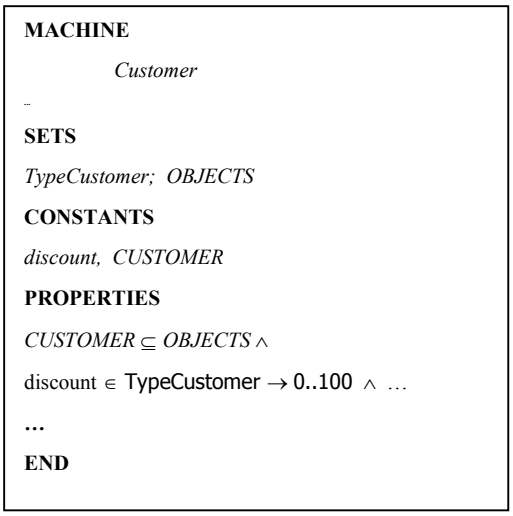

Fig. 9 Machine Customer

Persistance des informations:

Quand un modèle a été augmenté d'informations supplémentaires sur une vue cible v_j, après la modification de l'information initiale sur la vue source v_i, la projection de cette modification sur v_j ne doit pas entraîner la perte des informations supplémentaires de v_j.

Par exemple, on souhaiterait renommer le type *CustomerType* de l'attribut *category* en *CATEGORY* (cf. Fig. 10). L'utilisateur voudrait affiner la description B de la constante *discount* en associant à chaque catégorie de clients, un pourcentage de réduction précis (cf. Fig. 11). Cette modification ne doit pas entraîner la perte de la contrainte déjà exprimée sur *discount*. Grâce au mécanisme de contrôle de consistance et de cohérence par construction, cette modification doit être propagée partout où *CustomerType* est utilisé.

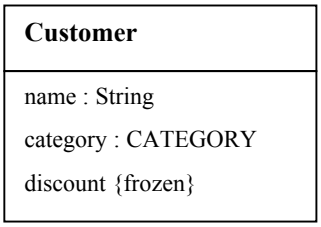

Fig. 10 Classe Customer modifiée

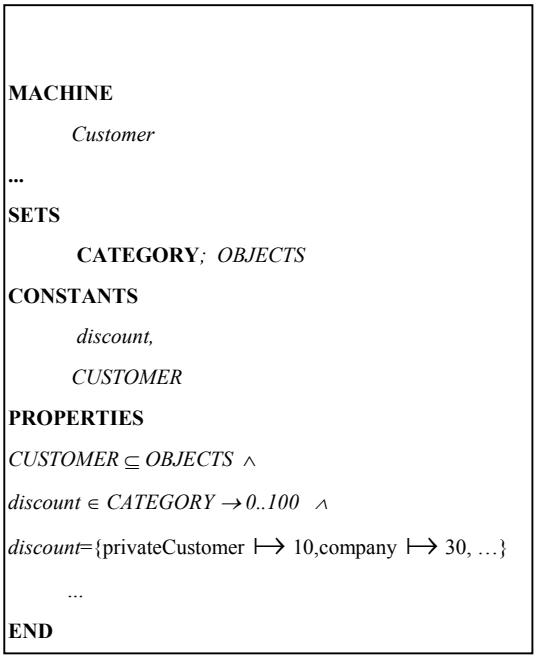

Fig. 11 Machine Customer modifiée

Assistance au développement :

L'assistance au développement est le mécanisme par lequel l'outil propose une rétroaction à l'utilisateur comme par exemple, il

1. lui propose la liste d'éléments qui peuvent être liés au concept manipulé. Par exemple, la déclaration d'une donnée dans une machine B est une opération qui est implicitement liée à l'écriture d'une propriété et/ou d'un invariant et/ou à l'initialisation relative à cette donnée,
2. lui indique les effets des actions qu'il opère sur la spécification,
3. lui signale les endroits où son attention est particulièrement attendue,
4. lui indique les tâches nécessaires à exécuter pour atteindre tel ou tel but.

6 - CONCLUSION ET PERSPECTIVES

Le processus d'intégration de B dans la construction de spécifications orientées objets UML et les mécanismes de transformation automatiques d'UML vers B sont susceptibles d'être améliorés malgré les avancées considérables de la recherche dans ce domaine. En effet, en plus de la formalisation d'UML en

B, il est actuellement important d'explorer les mécanismes flexibles et performants qui rendent cette formalisation transparente et attractive.

Dans ce papier, nous avons proposé une nouvelle forme d'intégration de la méthode B dans le processus de construction de spécifications orientées objets UML par construction systématique de spécifications multi-vues. Nous en avons présenté les concepts essentiels. Dans ce cadre, plusieurs aspects doivent encore être étudiés et approfondis. Nous pensons notamment à la formalisation des actions de spécification pouvant être prises en charge dans un tel processus. Une telle formalisation pose de solides bases à l'analyse des différents liens intra et extra-vue pour chaque aspect modélisé. Nous nous intéressons également à l'approche incrémentale afin de permettre un prototypage rapide par aspects.

BIBLIOGRAPHIE

Abrial J.R. *B Book –Assigning Programs to Meanings-,* Cambridge University Press, ISBN 0-521-49619-5, 1996

Alanen M., and Porres I. *Difference and Union of Models*. In <<UML>> 2003 Conference. Vol. LNCS 2863. Springer, San Francisco, California, USA, October 20-24, 2003.

Dupuy S. *Couplage de notations semi-formelles et formelles pour la spécification des systèmes d'information*. PhD thesis, Université Joseph Fourier-Grenoble 1, Grenoble(F), septembre 2000.

France R.B., Grant E. and Bruel J-M. *UMLtranZ: An UML-Based Rigorous Requirements Modeling Technique.* Technical report, Colorado State University, Ft. Collins, Colorado, January 2000.

Warmer J., A. Kleppe *The Object Constraint Language: Precise Modelling with UML.* Addison-Wesley 1999, ISBN 0-201-37940-6.

Okalas O. D., Souquières J., Jacquot J-P. Assistance à la *construction de specifications multi-vues UML et B*. Poster. In MAJECSTIC'03, Marseille (F), 29-31 octobre, 2003.

Wang E.Y., Richter H.A. and Cheng B.H.C. *Formalizing and integrating the dynamic model within OMT**. In ICSE'97:19th International Conference on Software Engineering Boston (USA), July 1997.

OMG The Object Management Group. *OMG Unified Modeling Language Specification, version 1.3*, March 2001.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorensen W. *Object-Oriented Modeling and Design.* Prentice Hall Inc. Englewood Cliffs, 1991.

Rumbaugh J., Jacobson I and Booch G. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1998. ISBN 0-201-30998-X.

Laleau R., Mammar A. *A Generic Process to Refine a B Specification into a Relational Database Implementation.* In ZB 2000: Formal Specification and Development in Z and B, LNCS 1878. Springer-Verlag York(UK), August/September 2000.

Lano K. *The B Language and Method: A Guide to Practical Formal Development.* FACIT. Springer-Verlag, 1996. ISBN 3-540-76033-4.

Ledang H., Souquières J. and Charles S. *ArgoUML+B: Un outil de transformation systématique de spécifications UML vers B*. Actes de la conférence AFADL'2003, INRIA, janvier 15-17, 2003, Rennes, France.

Ledang H. *Traduction Systématique de Spécifications UML vers B.* PhD thesis, LORIA -Université Nancy2, novembre 2002.

Meyer E. *Développements formels par objets: utilisation conjointe de B et d'UML*. PhD thesis, LORIA -Université Nancy2, mars 2001.

Nguyen H.P. *Dérivation de Spécifications Formelles B à Partir de Spécifications Semi-Formelles*. PhD thesis, Conservatoire National des

Arts et Métiers (CNAM), décembre 1998.

Snook C., M. Butler M. *Tool-Supported Use Of UML for Constructing B specifications*. DSSE Technical Report, April 2002. Avaible at http://www.ecs.soton.ac.uk/~mjb/

Jackson M., Zave P. *Conjunction as composition*. ACM Transactions of Software Engineering and Methodology. Vol 2(4), p. 379-411, 1993.

RoZ Disponible à l'adresse: *http://www-lsr.imag.fr/Les.Groupes/PFL/RoZ/index.html*