HAL
open science

Concepts importants à la construction de spécifications multi-vues UML et B

Dieu Donné Okalas Ossami, Jeanine Souquières, Jean-Pierre Jacquot

To cite this version:

Dieu Donné Okalas Ossami, Jeanine Souquières, Jean-Pierre Jacquot. Concepts importants à la construction de spécifications multi-vues UML et B. Informations, Savoirs, Décisions et Médiations [Informations, Sciences for Decisions Making] , 2004, 13, 14 p. inria-00107756

HAL Id: inria-00107756

https://inria.hal.science/inria-00107756v1

Submitted on 19 Oct 2006

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

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

Concepts importants à la construction de spécifications multi-vues UML et B

Dieu Donné Okalas Ossami Jeanine Souquières Jean-Pierre Jacquot

LORIA - UMR 7503
Campus scientifique, BP 239
54506 Vandœuvre-lès-Nancy Cedex - France

E-mail: {souquier, okalas, jacquot}@loria.fr

Abstract

La technique d'intégration d'UML et B est encore susceptible d'être améliorée. Nous pensons notamment à la gestion du va et vient entre les deux représentations induites. On sait aujourd'hui transformer les diagrammes UML et certaines expressions OCL en B, mais pas la contraire. Le manque de retour de B vers UML laissent 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 et pouvoir d'expression plus outils de preuve pour B. Ce papier présente les concepts et la démarche générale de construction de spécification multi-vues UML et B.

mots clés: *B, UML, méthodes intégrées, spécification multi-vues.*

1 Motivations

De nos jours, le développement du logiciel devient de 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'oeuvre de plusieurs spécialistes (analystes, développeurs, testeurs, etc.) qui interagissent à travers divers documents (cahier des charges, spécifications, documents de conception, plans de tests, programmes,...). 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, LOTOS, 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 font d'UML une méthode populaire et très utilisée. La difficulté liée à la conception de modèles UML fiables réside dans le manque d'outils de validation mathématique des propriétés modélisées par ces modèles. Conscients de cet handicap, les industriels se sont sensibilisés à la nécessité de mettre au point et d'intégrer les méthodes formelles de vérification comme éléments à part entière du cycle de développement de logiciels fiables. Car, l'expérience a montré qu'une erreur de spécification aussi infime soit elle détectée plus tard à l'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 de contrôle de conformité de logiciels avec leurs spécifications. Elles possèdent une notation ayant une syntaxe et sémantique précises et sont souvent supportées par des outils d'animation de preuves automatique ou interactive. Mais le fait que leur notation soit à base des 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 juste titre que de nombreuses recherches ont été menés sur le couplage d'UML et les méthodes comme Z, B ou LOTOS. 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 d'UML d'une part, avec la fiabilité, le pouvoir d'expression et la sémantique des méthodes formelles (Z, B) d'autre part. Ces travaux ont permis de mettre au point un certains nombre d'outils (citer les outils) de génération de spécifications B, Z, etc. à partir de diagrammes UML.

Aujourd'hui ces outils permettent de transformer les diagrammes UML et certaines expressions OCL en B, mais pas la 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 outils d'aide au développement conjoint, ne dépend pas seulement de sa capacité à projeter un langage source en un autre 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 tentons à étudier 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, permettra au concepteur de faire usage du meilleur des deux: clarté architecturale pour UML et pouvoir d'expression plus 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 en B actuel, en l'occurrence la traduction systématique de diagramme UML en B en général et inversement. 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 3.2 dresse un retour d'expérience sur l'utilisation conjointe d'UML et B, en l'occurrence la transformation systématique d'UML en B. Les avantages et les limites de cette démarche y sont présentés. 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 Brève présentation d'UML et B

2.1 UML

UML [8][10, 9] est une technique de modélisation standardisée et très répandue dans l'industrie du logiciel. UML 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 diagrammes représentant chacun un aspect particulier du système (aspects fonctionnel, statiques, dynamiques, etc.)
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.

Éléments du diagramme de classes UML

Un diagramme de classes UML consiste à un ensemble de classes reliées entre elles par des associations et des hiérarchies de généralisation (c.f. figure 2.1). 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'aggrégation. 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.

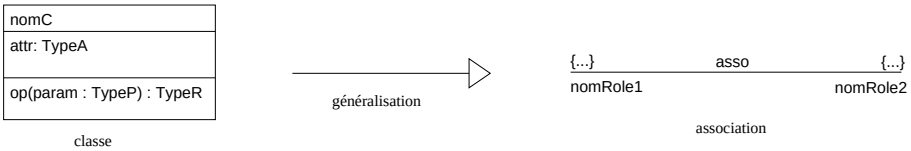


FIG. **1 –.** *Éléments structuraux du diagramme de classes UML*

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'états transition). Chaque diagramme constitue une des représentations partielles (vue) du modèle UML.

2.2 La méthode B

La méthode B[4] est une méthode de spécification formelle qui comprend toutes les phases de développement logiciel, de la spécification jusqu'à l'implémentation. La méthode B permet d'exprimer des propriétés sous la forme de prédicats 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 2.2 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 structuré 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 inclue 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, constants, ensembles) et de contraintes (c.f. 2.2, $\mathcal{C}$, $\mathcal{P}$, $\mathcal{I}$, etc.) que ces données doivent toujours vérifiées. La partie opérationnelle est constituée de l'initialisation et des opérations. Elle est basée sur le langages de substitutions généralisées.

MACHINE
  Ma(p) / nom et liste des paramètres de la machine */*
CONSTRAINTS
  $\mathcal{C}$(p) / contrainte sur les paramètres */*
CONSTANTS
  c /· liste des constants ·/
SETS
  s / liste des ensembles différés ou génériques */*
  s' = {a, b, c, ..} / liste des ensembles énumérés */*
PROPERTIES
  $\mathcal{P}$(c, s, s') / propriétés sur les constants et les ensembles */*
VARIABLES
  v /· liste des variables ·/
INVARIANT
  $\mathcal{I}$(p, c, s, s', v) / propriétés d'invariance */*
INITIALISATION
  $\mathcal{INIT}$ /· initialisation ·/
OPERATIONS */* liste des opérations */*
  Op = PRE P THEN $\mathcal{S}$ END
  ...
END

FIG. **2 –.** *Structure d'une machine abstraite B*

Sur la figure 2.2, les clauses *CONSTRAINTS*, *PROPERTIES*, *INVARIANT*, etc. regroupent un ensemble de prédicats ($\mathcal{C}$, $\mathcal{P}$, $\mathcal{I}$, 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. $\mathcal{X}(j)$ signifie que la donnée *j* apparaît dans le prédicat $\mathcal{X}$. Les prédicats $\mathcal{C}$, $\mathcal{P}$ et $\mathcal{I}$, *INIT*, *Pre* et *Post* doivent être écrits de telle sorte qu'ils garantissent la consistance interne de la machine [12] (chapitre 2).

3 Intégration d'UML et B: limites et avantages

3.1 Avantages de l'intégration 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 et/ou augmenté de contraintes OCL et de traduire ces diagrammes en B, puis de mettre en oeuvre 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,
- la transformation permet d'établir une sémantique B des modèles UML

L'utilisation des outils d'animation automatique de preuve 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 pre 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 à petits pas, mais aussi de maintenir les propriétés de processus tout au long du développement. Pour ce dernier point, il s'agit de s'assurer qu'il n'existe pas de contractions entre les différentes abstractions du système.

3.2 Limites de l'intégration UML et B

Malgré ses 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 (i.e figure 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.

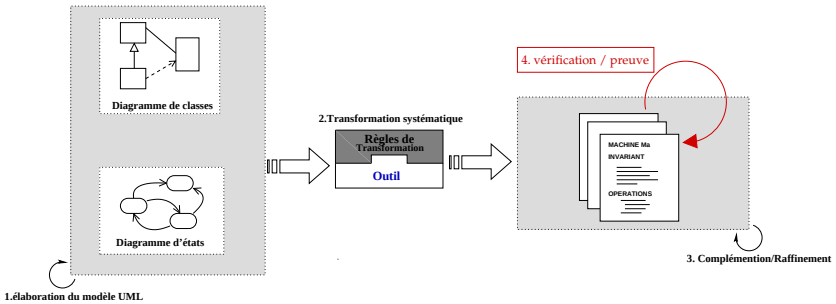


FIG. **3 –.** *Processus de dérivation UML vers B*

La majorité des approche du couplage d'UML et les méthodes formelles Z[6][16], B[13][14] ou LOTOS[7] suivent le processus présenté par la figure 3. Les travaux autour du couplage d'UML avec Z et B ont donnés lieu à des outils de génération de spécifications Z[2] ou B [**?**, **?**][17][11] à partir de diagrammes UML. Le mode opératoire de ces outils est celui de la figure 3.

3.3 Vers une structuration en vues UML et B

Pour surmonter les limites actuelles de la traduction d'UML en B (c.f. paragraphe 3.2), on pourrait penser à continuer 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 limiter pour une raison essentielle:

- la notation B contrairement à celle d'UML n'est pas orientée objets. Cette différence implique inévitablement l'existence d'incompatibilités entre les deux formalismes. La spécification B déduite automatiquement d'UML et inversement, 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 ne sont plus déduites systématiquement, mais dépendent de l'aspect décrit du contexte de modélisation. Dans un telle approche, le concepteur ne travaille plus sur deux spécifications indépendantes, mais sur une des deux représentations d'une même spécification. Les vues permettront au concepteur de faire usage du meilleur des deux: clarté architecturale pour UML et pouvoir d'expression plus outils de preuve pour B. Nous appelons cette démarche la *construction de spécifications multi-vues UML et B*.

4 Notre approche: construction de spécification multi-vues UML et B

La *construction de spécifications multi-vues UML et B* est une extension des travaux de Hung [**?**] et de Meyer [13] 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 les vues ouvertes (diagrammes de classes, diagramme d'état-transition et spécification B induite) de celle-ci. 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 de 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* (c.f figure 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 modélisés (classe, attribut, opération, variable, machine, raffinement, etc.). L'outil modifie alors systématiquement les deux représentations en fonctions des informations saisies et des contraintes de propagation définies par les règles de transformation.

Nous pensons en effet 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 (i.e *clients* sur la figure 4), de la modélisation adoptée. Une telle dynamique permet donc de suivre, pas à pas, les raisonnements réalisées afin de valisualiser la représentation adoptée. La figure 4 présente l'architecture générale de cette démarche.

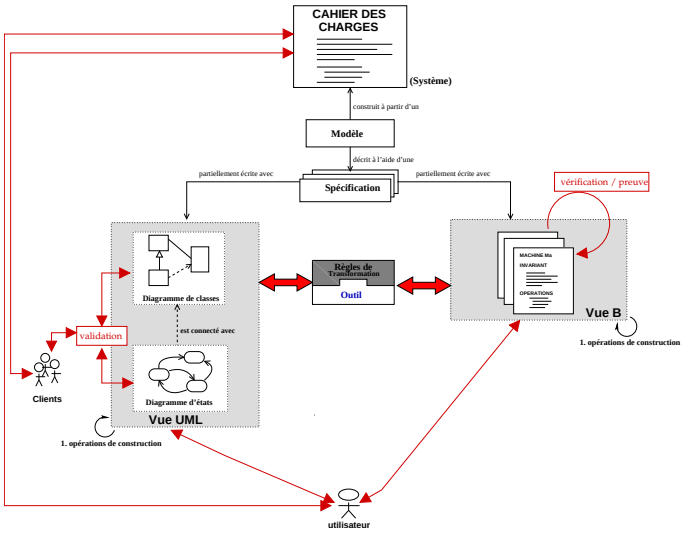


FIG. **4 –.** *Assistance à la construction de spécifications multi-vues UML et B extrait de [5]*

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

- utiliser UML et B de façon complémentaire: faire usage du meilleur des deux formalismes,
- proposer un cadre de transformation et un environnement de développement interactif de spécifications par combinaison de plusieurs formalismes (ici UML et B),
- 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) définit chacun par une syntaxe et une sémantique données,
2. un ensemble de propriétés *Prop* d'un modèle *m* à construire,

Determiner:

1. un ensemble d'opérations $\mathcal{O}$ qui permettent de construire les éléments des langages utilisés et permettre ainsi de décrire les propriétés du modèle (*Prop*) entièrement ou partiellement sur une ou sur les deux vues de $\mathcal{V}$ dans une dynamique de développement simultané UML et B,
2. un ensemble de règles $\mathcal{T}$ 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 de base fournis par la construction de spécifications multi-vues UML et B

Ce paragraphe est consacré à l'ensemble essentiel des concepts qui doivent être supportés par la construction de spécifications multi-vues UML et B. Ces concepts sont relatives aux notions de *vues*, de *spécification hétérogène multi-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 [18] ont montré que l'utilisation de vues et de notations adaptées à chaque aspects de l'application pouvait clarifier la spécification. UML intègre déjà 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 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 natures différentes. 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-modèle respectifs des deux formalismes. Par exemple, le méta-modèle UML [1] définit la syntaxe et la sémantique des constructions UML et le schéma de construction et la sémantique des concepts B sont décrits dans la littérature relative à la méthode B [3, 4].

Définition 1 *(Vue)*

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

Où

– $v_i ::= V_U \mid V_B$ 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. Toute fois, nous en présenterons à chaque que c'est nécessaire.

5.2 Spécification hétérogŁne multi-vues UML et B

Dans [15] Paige définit une spécification hétérogène comme un ensemble de descriptions écrites avec deux ou plusieurs formalismes. L'interprétation de cette définition dans la cadre d'UML et B conduit à considérer la spécification hétérogène multi-vues UML et B comme un ensemble de descriptions écrites pour partie en UML et pour partie en B sur deux vues de représentation différentes.

Définition 2 *(Spécification hétérogène multi-vues UML)*

Une spécification hétérogène multi-vues UML et B peut être comme un tuple (m_U, m_B)

L'interprétation de la définition 2 est que si deux constructions partielles e_i et e_j décrivent un même aspect sur deux vues de représentation différentes, alors (e_i, et_j) est la spécification hétérogène multi-vues de cet aspect.
Les éléments e_i et e_j entretiennent deux types de relations: les relations calculées T et les simples correspondances C.
La figure 5) présente ces différents liens. Elle présente une classe *Customer* qui modélise le fichier client d'une société. Cette classe possède trois attributs *name*, *category* et *discount*. Le *name* caractérise le nom du client, *category* modélise le type de client (particulier, abonné, autres,...) et *discount* spécifie le pourcentage de réduction attribué à chaque catégorie de clients. Dans toute la suite, nous allons utiliser cette classe pour illustrer notre propos. L'exemple est petit, mais il est suffisant pour illustrer les idées présentées dans ce papier.

1. les relations calculées (notées T sur la figure 5) sont déduites automatiquement par l'application systématique des règles de transformation tel que : $T = \{(r_i, r_j) \mid r_i = (e_i, e_j) \wedge r_j = r_i^{-1}$ avec $e_i \leftrightsquigarrow e_j\}$ est l'ensemble de couples de règles (r_i, r_j) qui projetent une ou plusieurs descriptions partielles UML $e_i \in m_U$ vers une ou plusieurs descriptions complémentaires B $e_j \in m_B$ et inversement.
2. les simples correspondances (notées 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 reférence à l'opération correspondante afin d'en assurer la traçabilité.

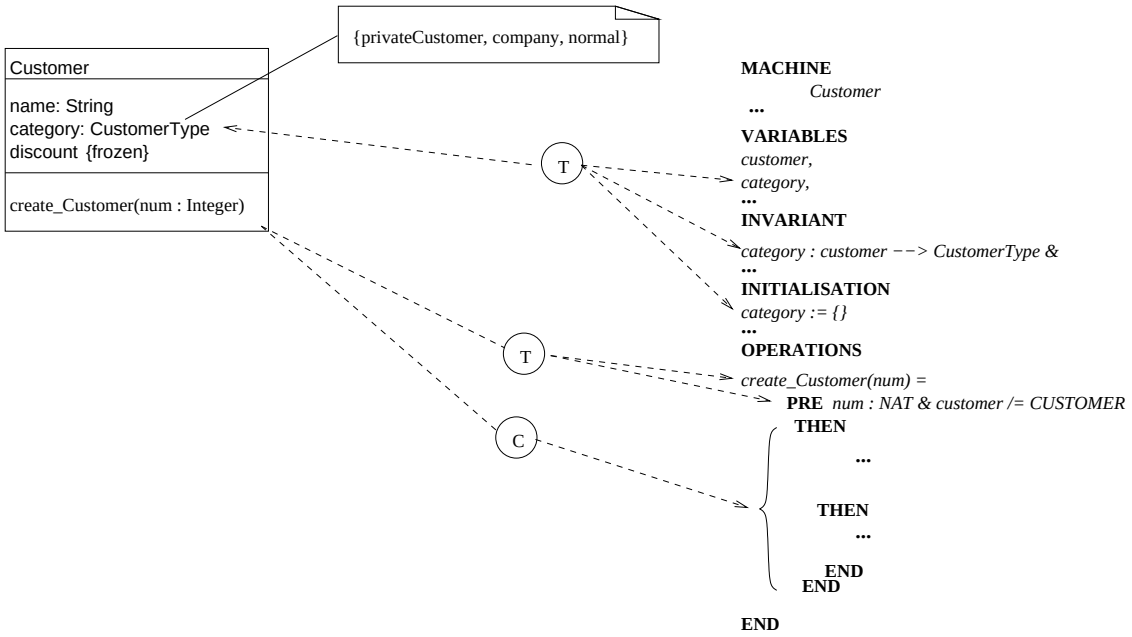


FIG. **5 –.** *Exemple de relations entre* m_U *et* m_B

5.3 Opération de construction de spécifications multi-vues UML et B

L'activité de développement peut être modéliser à travers un ensemble d'opérations atomiques: *opérations de construction*. Ces opérations caractérisent la logique du développement de l'utilisateur. Un des objectifs de la construction de spécifications multi-vues UML et B est d'identifier et de caractériser ces opérations.

Le support essentiel à l'identification des opérations de construction est l'analyse des propriétés constitutives de chaque élément de modélisation (dans le méta-modèle UML et le schéma de construction des données B).

Nous utilisons la notation UML pour caractériser les propriétés les éléments de modélisation. Les propriétés d'un élément sont modélisées comme des attributs de celui-ci ou comme un lien de composition avec d'autres éléments. Par exemple la figure 6 présente un élément de type *Association* qui est caractérisé par une propriété, *name*. Elle possède un lien de composition avec un autre élément *AssociationEnd* qui est caractérisé par un ensemble de propriétés. Pour chaque propriété, on peut définir un ensemble d'opérations qui permettent de la manipuler dans un certain contexte.
Dans cette section, nous une définition d'opérations de construction dans le cadre de notre étude.

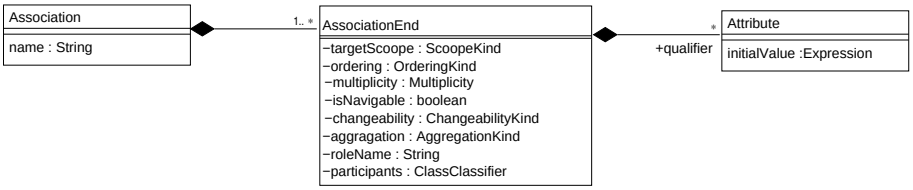


FIG. **6 –.** *Caractérisation de propriétés d'une association UML*

Définition 3 *(Opération de construction)*

*Une **Opération de construction** $\mathcal{O}$ est une action élémentaire effectuées par l'utilisateur pour décrire un aspect donné du système.*

L'identification d'opérations de construction à travers les propriétés d'éléments de modélisation permet d'affiner le processus de construction et de définir le niveau de granularité des transformations nécessaire à la projection d'un élément source vers un ou plusieurs éléments d'une vue cibles. Nous regroupons les opérations de construction dans une méta-opération *change* qui modélise le changement de vues et donc l'évolution de la spécification.

Exemple. 1 (Opération de construction sur une association)
La figure 6 illustre l'évolution de la spécification par la manipulation d'une association. Cette évolution se traduit par le changement la vue UML qui modélise l'association manipulée en effectuant une ou plusieurs opérations de construction (c.f. figure 5.3) atomiques sur une ou toutes les propriétés de celle-ci.

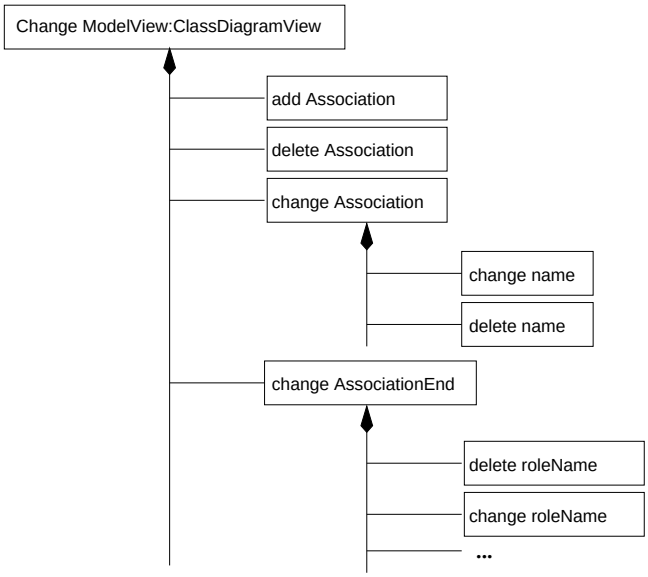


FIG. **7 –.** *Exemple schématique d'opérations de construction atomiques d'une association*

L'évolution de la spécification se résume à l'utilisation des opérations $\mathcal{O}_0$,...,$\mathcal{O}_i$ (*add, change, delete, etc.* sur les propriétés p_o,...,p_i d'un élément donné. Les changement causés par une opération se situent à deux niveaux:

- au niveau de la vue où elle est exécutée (UML ou B)
- au niveau de toutes les autres représentations liées à l'élément manipulé (autres diagrammes UML, spécification B).

Dans ce contexte, dessiner une association par exemple est implicitement équivalent à la déclaration, à l'écriture d'un invariant et à l'initialisation d'une variable B modélisée comme une relation entre les ensembles de toutes

les instances des classes participantes à l'association. Le changement de l'une des propriétés de cette association ou de la donnée B correspondante implique systématiquement le changement de toutes les représentations liées à l'association ou à la donnée B.

- Soient $e \in V_i$ et $d \in V_j$ deux éléments de modélisation (ou expressions) décrits respectivement sur deux vues de représentation différentes V_i et V_j,
- soit $\mathcal{P}_e$ l'ensemble des propriétés qui caractérisent e,
- on désigne par *changed(p_i)* une fonction qui renvoie vraie si la propriété p_i a été changée (supprimer, modifier, etc.).

Définition 4 *(Changement d'un élément)*

Un élément e est considéré comme changé si:
- $changed(p_i), \forall\, p_i \in \mathcal{P}_e \vee$
- $changed(p_j) \wedge e \leftrightsquigarrow d, \forall\, p_j \in \mathcal{P}_d$.

Où
- $e \leftrightsquigarrow d$ exprime une relation calculée entre **e** et **d**,
- $\mathcal{P}_d$ est l'ensemble des propriétés qui caractérisent **d**,

La propagation des effets de changements d'un élément est assurée par la transformation.

5.4 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, cette transformation doit être bidirectionnelle pour garantir la cohérence par construction entre les deux représentations. 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 déduites 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 dédié à la présentation de ces deux notions.
La transformation est un processus de traduction automatique d'une ou plusieurs constructions d'un modèle m_1 vers une ou plusieurs constructions d'un modèle m_2. La transformation est modélisée par une collection de règles de transformation $r_0, ..., r_i$.

Définition 5 *(Règles de transformation (r))*

Les règles de transformation sont des mécanismes qui montrent comment une ou plusieurs constructions d'un langage L_1 (i.e UML ou B) sont projetées vers une ou plusieurs constructions d'un langage L_2 (i.e UML ou B). Une règle de transformation r_i est un tuple (e_i, e_j) qui consiste à:
- *un élément source $e_i \in m_i$ (i.e m_U ou m_B),*
- *un élément cible $e_j \in m_j$,*

Définition 6 *(Transformation bidirectionnelle)*

La transformation entre deux éléments e_i et e_j est bidirectionnelle si pour $e_i ::= [p_0,...p_i]$ et $e_j ::= [p_0,...p_j]$, il existe un ensemble de règles (r_i, r_j) pour lesquelles il existe au moins une propriété p_i et p_j tel que les relations suivantes soient vraies:
- *$e_i.p_i \rightsquigarrow e_j.p_j$ satisfait par r_i et*
- *$e_j.p_j \rightsquigarrow e_i.p_i$ satisfait par r_j*

L'application d'une règle de transformation est une conséquence logique de l'utilisation d'une ou plusieurs opérations de construction. Dans un développement multi-vues, une règle n'est applicable que si les conditions de modélisation des éléments source et cible sont satisfaites. Par exemple, la traduction d'une variable B en un attribut n'est possible que s'il n'existe pas un attribut de même nom dans la classe correspondante à la machine dans laquelle la variable est modélisée. La structure et la forme des règles de passage de B à UML ne sont pas traitées dans ce papier.
Dans une démarche de construction simultanée, la transformation bidirectionnelle est un pont entre les différentes représentations. Elle doit donc être munie d'un système de contrôle de consistance capable de garantir la cohérence par construction et la qualité des descriptions produites. Dans ce cadre, nous envisageons les mécanismes de contrôle de consistance suivants:

1. **Consistance et contrôle de cohérence par construction**
   - *consistent(M,m_i)* assure que les constructions m_i UML ou B induites sont valides par au langage dans lequel elles sont écrites

– *coherent(v_i,e_i)* assure que la spécification induite reste toujours cohérente au regard des manipulations opérées sur tout aspect e_i: les modifications opérées sur une vue v_i (vue UML ou vue B) sont automatiquement propagées sur toutes vues ouvertes de la spécification.

2. **Flexibilibilité**
pour modéliser une donnée en B, l'utilisateur est parfois contraint de saisir certaines informations qu'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 prévoit dans ce cadre un mécanisme de spécification partielle de données, comme le mécanisme d'attributs dérivés.
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:

(a) 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. La figure **??** illustre ce propos.

MACHINE
Customer
...

SETS
TypeClient; *OBJECTS*

CONSTANTS
discount, *CUSTOMER*

PROPERTIES
$CUSTOMER \subseteq OBJECTS \wedge$
$discount \in \perp$
...

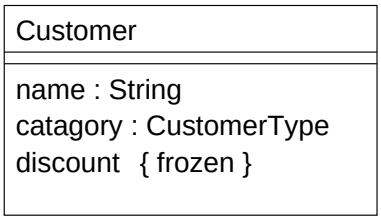
VARIABLES
customer, *category*, *name*
...

INVARIANT
$customer \subseteq CUSTOMER \wedge$
$category \in customer \rightarrow TypeClient \wedge$
$name \in customer \rightarrow string$
...

INITIALISATION
$customer := \perp \,||$
$name := \varnothing \,||$
$categoty := \perp$
...

END

(a) Machine **Customer**



(b) Class **Customer**

$\perp$ 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).

(b) 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 projeté en UML comme en témoigne la propriété de la constante *discount* sur la figure 8(g). Le lien existant entre cette propriété et l'attribut est une simple correspondance.

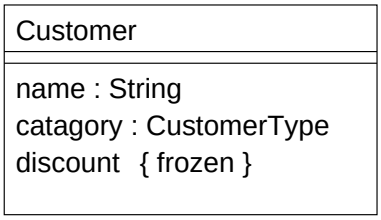
MACHINE
Customer
...

SETS
TypeClient; *OBJECTS*

CONSTANTS
discount, *CUSTOMER*

PROPERTIES
$CUSTOMER \subseteq OBJECTS \wedge$
$discount \in (TypeClient \rightarrow 0..100)$
...
END

(c) Machine **Customer**



(d) Class **Customer**

3. **Persistence des informations:**
quand un modèle a été augmenté d'informations suppléménetaires 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 suppléménetaires de v_j. Par exemple, on souhaiterais renommer le type *CustomerType* de l'attribut *category* en *CATEGORY*. L'utilisateur voudrait affiner la description B de la constante *discount* en associant à chaque catégorie de clients, un pourcentage de réduction précis (c.f. figure 8(g)). 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.

MACHINE
Customer
...

SETS
$CATEGORY = \{privateCustomer, company, normal\};$
$OBJECTS$

CONSTANTS
$discount, CUSTOMER$

PROPERTIES
$CUSTOMER \subseteq OBJECTS \wedge$
$discount \in CATEGORY \rightarrow 0..100 \wedge$
$discount = \{privateCustomer \mapsto 20, company \mapsto 40, normal \mapsto 5\}$
...

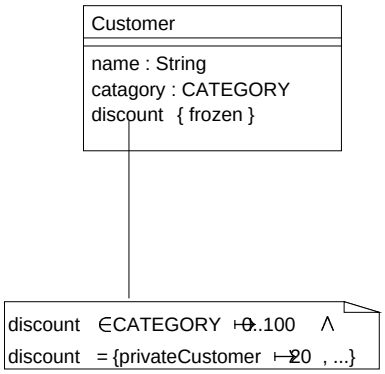
VARIABLES
$customer, category, name$
...

INVARIANT
$customer \subseteq CUSTOMER \wedge$
$category \in customer \rightarrow CATEGORY \wedge$
$name \in customer \rightarrow string$
...

INITIALISATION
$customer := \varnothing \mid\mid$
$name := \varnothing \mid\mid$
$categoty :=$ **normal**
...

END

(e) Machine **Customer**



(f) Class **Customer**

4. **traçabilité:**
permet de localiser et de visualiser les liens d'utilisation d'un élément donné sur toutes les vues où il est référencé.

MACHINE
Customer
...

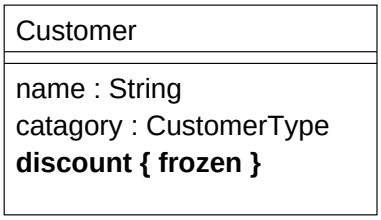
SETS
$CATEGORY; OBJECTS$

CONSTANTS
discount,
$CUSTOMER$

PROPERTIES
$CUSTOMER \subseteq OBJECTS \wedge$
discount $\in CATEGORY \rightarrow 0..100 \wedge$
discount $= \{privateCustomer \mapsto 20, company \mapsto 40, normal \mapsto 5\}$
...

END

(g) Machine **Customer**



(h) Class **Customer**

5. **assistance au développement:**
L'assistance au développement est mécanisme par lequel l'outil propose une rétroaction à l'utilisateur comme par exemple :
   (a) lui proposer la liste d'éléments qui peuvent être liés au concept manipulé. Par exemple, la déclaration d'une donnée dans une machine B est opération qui est implicitement liée à l'écriture d'une propriété et/ou d'un invariant et/ou à l'initialisation relatif à cette donnée,
   (b) lui indiquer les effets des actions qu'il opère sur la spécification,
   (c) lui signaler les endroits où son attention est particulièrement attendue,
   (d) lui indiquer 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 encore 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.

Références

[1] Object Management Group. *http://www.omg.org*.

[2] http://www-lsr.imag.fr/les.groupes/pfl/roz/index.html. February 22, 2000.

[3] *Manuel de référence du langage B.* -ClearSy-, Novembre 1998.

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

[5] D. Okalas Ossami, J-P. Jacquot et J. Souquières. Assistance à la construction de spécifications multi-vues UML et B, Marseille (F), 29-31 ocotobre, 2003. Avaible at http://www.loria.fr/ okalas.

[6] S. Dupuy. *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.

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

[8] The Object Management Group(OMG). *OMG Unified Modeling Language Specification.* June 1999. Version 1.3.

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

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

[11] R. Laleau and A. Mammar. 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*, Yok(UK), August/September 2000.

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

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

[14] H. P. Nguyen. *Dérivation de Spécifications Formelles B à Partir de Spécifications Semi-Formelles.* PhD thesis, Conservatoire National des Arts et Métiers, décembre 1998.

[15] R. F. Paige. A meta-method for formal method integration. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313, pages 473–494. Springer-Verlag, 1997.

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

[17] C. Snook and M. Buttler. U2B: a tool for combining UML and B. Avaible at http://www.ecs.soton.ac.uk/ cfs/U2Bdownloads/U2Bdownloads.htm.

[18] P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions of Software Engineering and Methodology*, 2(4):379–411, 1993.