Opérations de construction de spécification multi-vues UML et B

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

Opérations de construction de spécifications multi-vues UML et B

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

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

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

Résumé : *Les travaux sur l'intégration d'UML et B se sont surtout focalisés sur la définition d'un processus de transformation séquentiel et unidirectionnel d'UML en B ou de B en UML. Aucun effort n'a été entrepris sur la définition d'une démarche de construction simultanée qui permette une plus grande flexibilité et la possibilité de faire évoluer la spécification tout en maintenant la cohérence et la traçabilité entre les deux représentations. L'évolution d'une spécification résulte de l'application d'une ou plusieurs opérations de construction. Celles-ci caractérisent l'expression des raisonnements suivis pour construire les différents aspects du système. Dans ce cadre, l'utilisation de plusieurs formalismes pour décrire un système ne peut être attractive et bénéfique pour le concepteur que si les outils qu'il exploite lui proposent non seulement des notations, mais aussi une démarche pour leur utilisation. La mise en place de telles fonctions nécessite la prise en compte de plusieurs facteurs comme par exemple le contexte de modélisation, la sémantique et la syntaxe des formalismes utilisés. Le travail présenté dans ce papier vise à apporter des éléments de réflexion sur la définition des opérations de construction de spécifications multi-vues UML et B.*

Mots clés : *B, UML, spécifications multi-vues, intégration UML et B, opération de construction.*

1 Motivations

On souhaite construire des spécifications en combinant les formalismes UML et B. Une telle démarche consiste à représenter les connaissances du domaine d'application à l'aide de diagrammes UML et à formaliser ces diagrammes en B, puis à mettre en œuvre des raisonnements sur ces représentations. Les principaux enjeux de cette démarche sont notamment la définition d'une sémantique B pour les diagrammes UML, la clarté architecturale, la facilité de lecture offerte par les diagrammes et l'utilisation des outils de preuve supportés par la méthode B.

Parmi les travaux s'attachant à utiliser conjointement UML [18, 19] et B [1], deux grands courants se distinguent. Le premier consiste à concevoir la spécification en UML et à la traduire ensuite en B [10, 11, 7]. Le second concerne la génération de diagrammes UML à partir de spécifications B [22] : l'utilisateur construit la spécification en B et celle-ci est ensuite traduite en diagrammes UML. Ces travaux ont permis de définir un processus de transformation séquentiel et unidirectionnel d'UML en B ou de B en UML, mais pas les deux à la fois. Aucun effort n'a été entrepris sur la définition d'une démarche de construction simultanée qui permette une plus grande flexibilité et la possibilité de faire évoluer la spécification tout en maintenant la cohérence et la traçabilité entre les deux représentations.

1.1 Limites de l'intégration d'UML et B

Les limites de l'intégration d'UML et B sont à la fois dues à la transformation entre les deux formalismes et au processus de construction induit.

1.1.1 Limites liées à la transformation

Elles sont dues au fait que UML et B sont de nature différente. Cette différence implique l'existence d'incompatibilités entre les deux formalismes :

- la structure des spécifications B obtenues par génération est très éloignée de celle des spécifications qu'on aurait écrites directement en B,
- la transformation d'UML en B reste souvent très syntaxique en fournissant des notations équivalentes, mais pas de démarche pour leur utilisation.

1.1.2 Limites liées au processus de construction

Les limites liées au processus de construction proviennent en partie du fait que la transformation systématique rompt très rapidement le lien UML/B et oblige à de nombreux travaux manuels de mise en cohérence et de complétion des spécifications B séparément de leur représentation en UML :

- l'assistance à la construction n'est quasiment pas explorée : la plupart des efforts ont porté sur la transformation systématique des formalismes,
- le processus de développement est séquentiel : concevoir le modèle UML, le traduire en B, poursuivre le processus avec la complétion et le raffinement de squelettes de spécifications B générées, intégrer une activité de validation. Un tel processus ne permet pas de garantir la cohérence des spécifications B produites avec le modèle UML initial à cause du manque de liens dynamiques entre les deux représentations. Dans ce contexte, Laleau et Polack proposent dans [8] une démarche pour inférer la transformation de B vers UML afin d'assurer la traçabilité des éléments UML impliqués dans les obligations de preuves non déchargées par le prouveur fournis par les outils de B [21, 2, 3].

  La totalité des approches du couplage d'UML et les méthodes formelles Z[5, 6], B[16, 13, 9, 7], LOTOS[24] ou CSP [15] suivent une démarche de développement séquentiel.

1.2 Solution

Pour surmonter ces limites nous proposons une nouvelle approche d'intégration des deux formalismes basée sur la structuration en *vues*. L'idée consiste à offrir à l'utilisateur la possibilité de construire une

spécification avec deux notations différentes et complémentaires UML et B. 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 [26] ont montré que l'utilisation de vues et de notations adaptées à chaque aspect de l'application pouvait clarifier la spécification. Pour construire cette spécification dans le cadre d'UML et B, l'utilisateur dispose des *opérations de construction* qui font évoluer la spécification sur toutes les parties dépendantes. Les tactiques définies par composition des opérations de construction guident l'utilisateur dans le développement de spécifications.

Ce papier est structuré comme suit. La section 2 décrit le principe général de la construction de spécifications multi-vues UML et B. La section 3 est dédiée à la définition de la notion d'opération de construction et à celle de tactique. La section 4 présente quelques exemples de tactiques. La section 5 revient sur les différentes approches d'intégration UML et B actuelles et dresse une comparaison avec la démarche proposée. Puis nous concluons et nous donnons quelques perspectives dans la section 6.

2 Principe de la solution : structuration en vues UML et B

La démarche de *construction de spécifications multi-vues UML et B* [17, 4] est une extension des travaux sur la traduction de diagrammes UML en B [16, 14, 13, 9, 7]. Elle consiste à considérer les représentations en UML et en B comme deux vues d'une même spécification. Pour obtenir une spécification conforme aux exigences de l'utilisateur, la construction de celle-ci résulte de l'utilisation d'opérations de construction qui ajoutent, suppriment ou modifient des éléments, non du fait de la transformation systématique d'une notation en une autre. La figure 1 présente l'architecture générale du système visé.

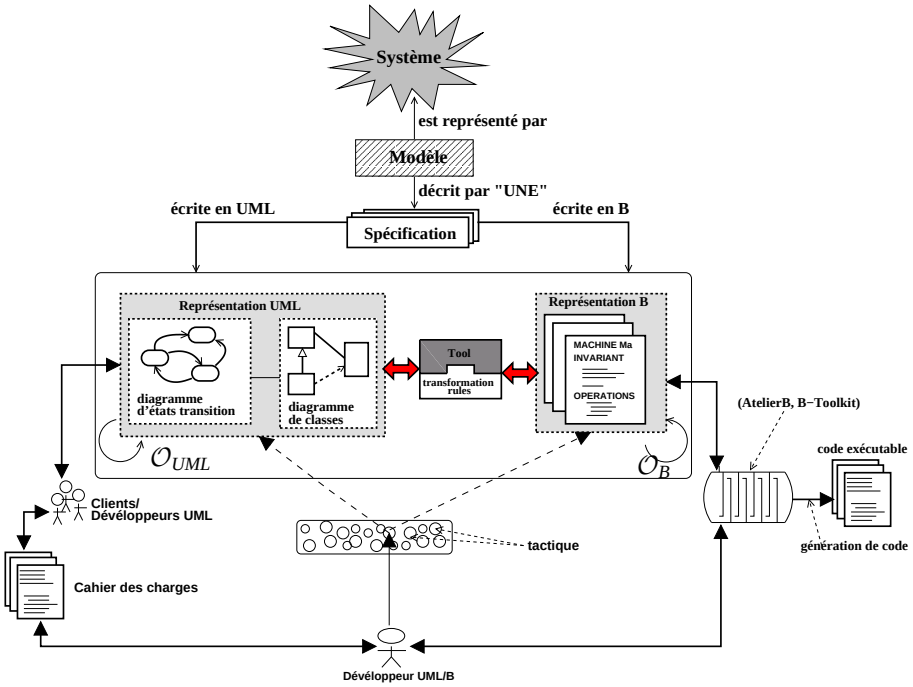

FIG. **1. Assistance à la construction de spécifications multi-vues UML et B**

A l'heure actuelle, nous disposons d'un outil *ArgoUML+B* [12] qui permet de traduire les diagrammes UML et les annotations OCL [25] en B. Le travail effectué dans notre étude porte sur l'enrichissement de cet outil. Il vise à :

1. Modéliser en termes de tactiques et d'*opérations de construction* sous-jacentes les raisonnements mis en œuvre et les choix effectués par l'utilisateur lors de la construction. Leur automatisation

doit garantir le suivi des évolutions et le maintien de la cohérence globale. Cette dernière découle de la capacité de l'outil à modifier plusieurs fois une description sans entraîner la perte des constructions existantes. Ainsi, pour ajouter un attribut, une classe ou pour changer la signature d'une opération, le développeur utilisera une opération de construction à travers l'application d'une tactique. L'outil modifie systématiquement toutes les représentations dépendantes. C'est sur ce point que l'approche montre son originalité par rapport aux techniques d'intégration existantes. Avec une implantation appropriée, l'approche permet d'assurer trois niveaux de cohérence par construction :

   (a) une gestion de cohérence entre les représentations des deux formalismes afin de répercuter les changements des éléments selon toutes les vues et sur toutes les parties dépendantes,
   (b) un contrôle de validité des informations saisies pour éviter la construction de spécifications incorrectes,
   (c) une traçabilité sur les évolutions de la spécification. La traçabilité permet de préserver la sémantique d'un élément et de séparer les parties introduites manuellement par le spécifieur des parties issues de la modélisation par génération. Elle permet de gérer des modifications délicates telles que le renommage, la création ou la destruction d'éléments.

2. Favoriser la construction incrémentale de manière à fournir une aide active au spécifieur en l'aidant à comprendre et à valider au fur et à mesure les choix de modélisation adoptés.

3. Exécuter automatiquement des actions implicites, comme par exemple la déclaration d'une variable avant son utilisation.

A terme, l'outil *ArgoUML+B* aura pour fonction principale de fournir au concepteur une véritable interaction, lui signalant les incohérences entre les propriétés exprimées et le contexte dans lequel elles sont modélisées et lui proposant des formulations alternatives mieux adaptées.

3 Opérations de construction de spécifications multi-vues UML et B

Dans cette section, nous présentons la notion d'*opération de construction*. Avant de définir cette notion, nous précisons le concept de *tactique* qui correspond à une composition particulière d'opérations de construction.

3.1 Tactique

Une tactique est définie comme une abstraction de haut niveau qui décrit la logique d'utilisation et les commandes d'édition d'éléments relatives au développement multi-vues en termes d' *opérations de construction*. L'idée fondamentale est de fournir aux concepteurs, entre autres, des commandes d'édition relatives à leurs tâches pour résoudre des problèmes dans un contexte donné et dont les mécanismes internes assurent la cohérence entre toutes les représentations induites à chaque itération du développement.
Les tactiques visent à guider systématiquement le développeur dans le choix des éléments et des opérations. Par exemple en lui fournissant la liste des éléments liés au concept manipulé, les conditions de leur modélisation, les opérations susceptibles de les manipuler et le moment de leur application ainsi que le contexte sous-jacent. Les paramètres d'une tactique proviennent de l'état courant du développement et de l'interaction avec l'utilisateur.

Forme générale d'une tactique

Une tactique est composée de diverses clauses qui ne font pas partie des notations UML et B.

Nom_de_la_Tactique

Description. Description en langue naturelle des buts et des effets attendus de la tactique.

Parameters. Détermine les éléments (les noms) requis par la tactique utilisée. Cette clause fournit deux types de paramètres : *In* et *Result*. Ils sont optionnels.
- **In.** Désigent les noms d'éléments sur lesquels les modifi cation vont être effectuées.
- **Result.** Désignent les noms d'éléments à être construits à partir des paramètres *In*. Les *Result* non spécifi és sont par défaut les mêmes que les paramètres *In*.

Pre-condition. Défi nit les conditions pour lesquelles une tactique peut-être appliquée.

Definition. Indique quelles opérations de construction utiliser et dans quel ordre elles doivent être exécutées.

Invariant de construction

A chaque tactique est associée la notion d'*invariant de construction* qui permet de garantir deux niveaux de contrats élémentaires en matière de développements logiciels : un qui concerne la syntaxe et un qui concerne la cohérence entre les différentes représentations.

- Le niveau syntaxique rend explicite le fait que les spécifications conçues doivent toujours être syntaxiquement correctes par rapport au langage utilisé.

- Les contrats de cohérence permettent de capturer les contraintes de développement garantissant les trois niveaux de cohérence évoqués dans la section 2. Sur ce point, l'exploitation des règles de transformation d'UML en B et de B en UML doit permettre de vérifier que la syntaxe et les propriétés sémantiques des spécifications B obtenues par génération restent cohérentes par rapport à celles obtenues par utilisation d'opérations de construction.

3.2 Opération de construction

Les tactiques de construction de spécifications multi-vues UML et B peuvent être vues comme des séquences d'exécution d'opérations de construction. Une opération est définie comme une action effectuée par l'utilisateur pour faire évoluer la spécification. Lorsque l'on définit une opération de construction, on utilise les concepts de haut niveau indépendant des notations utilisées comme *component, data, type, behaviour* ou *relation*. On ne se préoccupe pas de la forme concrète qui peut leur être donnée au niveau des langages. Ce rôle est celui de l'outil qui calcule les éléments syntaxiques propres à chaque notation par rapport à la tactique utilisée et au contexte de modélisation courant. Chaque opération de construction est subdivisé en deux sous-ensembles, $\mathcal{O}_{UML}$ et $\mathcal{O}_B$ (e.g. figure 3) d'opérations ou d'actions atomiques relatives à la construction des représentations UML et B.

opération de construction	*Description*
new	Introduction d'un nouvel élément par ajout successif des caractéristiques qui défi nissent la syntaxe ou schéma de construction de cet élément.
change	Modifi cation d'un élément. Elle correspond à la modifi cation d'une ou plusieurs propriétés qui le caractérisent.
remove	Suppression d'un élément. Elle correspond à la suppression de chacune de ses caractéristiques.
$\mathcal{O}$	Désigne toutes les autres opérations disponibles dont *skip* (opération sans effet).

FIG. **2. Opérations de construction de base**

Les opérations de $\mathcal{O}_{UML}$ et $\mathcal{O}_B$ peuvent contenir d'autres opérations ou des actions atomiques. Du point de vue de la spécification, une opération de $\mathcal{O}_{UML}$ ou de $\mathcal{O}_B$ correspond à une séquence d'actions atomiques qui ajoute, modifie ou supprime successivement des caractéristiques définissant la syntaxe ou schéma de construction d'un élément. Par exemple, l'adjonction d'une variable correspond à l'utilisation de l'opération *newVariable* qui se traduit par les actions *addName*, *addInvariant* et *addInitialisation*. Du point de vue de l'utilisateur, ces actions correspondent à la déclaration, à la modélisation de l'invariant

et à l'initialisation de la variable à modéliser. Les opérations de $\mathcal{O}_{UML}$ et de $\mathcal{O}_B$ travaillent en parallèle sur les deux représentations ; ce sont des transformations qui font évoluer la spécification d'un état S_{old} à S_{new} (e.g. figure 3). Les paramètres d'une opération sont ceux de la tactique qui l'incorpore. On distingue différents types d'opérations de construction de base qui définissent les tactiques, figure 2.

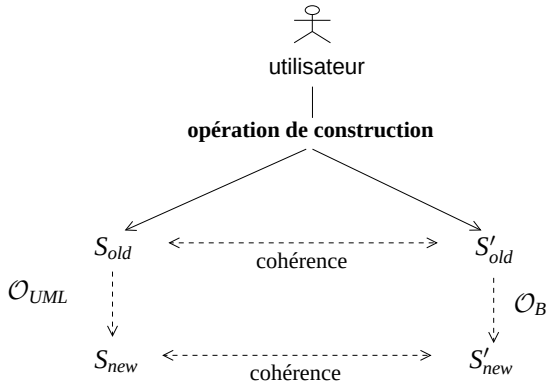

FIG. **3. Décomposition d'une opération de construction**

4 Exemples de tactiques et d'opérations de construction

Dans cette section, nous décrivons trois exemples de tactiques pour illustrer la caractérisation d'opérations de construction. Ce sont des tactiques relatives à l'introduction et à la composition de composants B et à l'introduction de *données* dans un composant B. Ces tactiques sont ensuite instanciées sur un petit exemple.

4.1 Introduction de composants

Description informelle

Cette tactique présente le processus d'introduction systématique d'un composant (machine abstraite) dans une spécification B. Les utilisateurs doivent indiquer le nom *Ma* du composant à introduire. L'opération *newComponent* est utilisée pour modifier la spécification qui s'enrichit du nouveau composant *Ma*. L'application de *newComponent* sur *Ma* induit les modifications suivantes :

1. **Sur le vue B**
   (a) La première application de cette tactique donne lieu à deux machines :
   - **Types** qui est une machine spéciale dans laquelle sont modélisés :
     - toutes les données partagées,
     - *OBJECTS*, l'ensemble de tous les objets possibles.
     - *MA*, l'ensemble de tous les objets possibles de la classe UML représentant la machine *Ma*.
   - **Ma** qui est une machine représentant le composant à introduire et dans laquelle sont modélisés :
     - un lien de composition avec la machine *Type*,
     - une variable *ma*, l'ensemble de tous les objets effectifs de la classe UML représentant la machine *Ma*.

   (b) Toute autre utilisation de cette tactique entraîne non seulement l'introduction de la machine correspondante, mais également la modification de la machine *Type* en y ajoutant l'ensemble de toutes les instances possibles de la classe UML correspondante.
2. **Sur la vue UML**
   - Pour la première application de cette tactique, un type *OBJECTS* qui représente l'ensemble de tous les objets possibles est créé [1].

[1] Pour des raisons de lisibilité, *OBJECTS* ne sera pas représenté graphiquement sur les diagrammes UML.

– Pour toute machine abstraite B *Ma* introduite, une classe *Ma* correspondante est générée [2]

Parameters

– **In**
  – *Ma* : componentIdentifier

Pre-condition

– $\forall\, Ma, Ma_i \in \mathcal{N} : Ma \neq Ma_i \wedge Ma \in (\text{'A'..'Z'})(\text{'a'..'z'}|\text{'A'..'Z'}|\text{'\_'}|\text{'0'..'9'})^+$.

Definition

operation de construction	$\mathcal{O}_B$	$\mathcal{O}_{UML}$
newComponent	newMachine *addName* *addCompositionLink*[a] *newDataType*[b] *newVariable*[c]	newClass *addName*

[a] utilisé lors de la création de toute machine hormis la machine *Type*.
[b] permet d'introduire les ensemble *OBJECTS* et *MA* dans la machine *Types*.
[c] variante de l'opération de la tactique 4.3 qui introduit la variable *ma* dans la machine *Ma*.

FIG. **4. Définition de l'introduction d'un composant**

La précondition de la tactique indique que :
– les identificateurs des machines B dans $\mathcal{N}$ sont des chaînes de caractères deux à deux disjointes de *a..Z, 0..9,* _ et dont le premier caractère appartient à *A..Z* où :
  – $\mathcal{N}$ est l'ensemble de tous les noms valides (UML et B) respectant les conventions de nommage de B où chaque nom d'éléments est une chaîne de caractères de deux ou plusieurs caractères.

Dans la démarche de génération de spécifications B à partir d'UML, la solution adoptée est de traduire une classe par une machine abstraite B. Toutefois, le lien sémantique fondamental entre ces deux concepts reste la notion d'ensemble. Ce qui explique l'introduction dans les machines B des ensembles *MA* et *ma* (e.g. *CLIENT et client* dans les machines *Types* et *Client* de la figure 5) qui modélisent les noms des objets de la classe. Toute modification portée à l'un de ces ensembles est implicitement une modification de la classe *Ma* correspondante. L'introduction d'un composant resulte de l'utilisation de l'opération *newComponent* obtenue en composant les operations B *newMachine* et UML *newClass* (voir Fig. 4).

Application de la tactique : introduction de composants

Prenons l'exemple de la phrase suivante. Nous considérerons cet exemple dans toute la suite :

> Une société propose à ses clients une gamme de produits. Les achats d'un client donné ne peuvent être considérés que si le montant global des produits achetés est inférieur ou égal à un montant limite donné.

Le point de départ de la modélisation de cette propriété consiste à décrire les entités *Société*, *Client* et *Produit*. Une instanciation de la tactique ci-dessus pour l'entité *Client* consiste à définir ses paramètres et à appliquer systématiquement les opérations proposées dans la figure 4.

[2] Il faut noter qu'une classe étant défi nie comme un ensemble d'objets partageant les mêmes propriétés et les même comportements, la création d'une classe est aussi implicitement celle d'un type d'objets disponible dans toute la spécifi cation.

Introduction de composants

Parameters

- **In**
  - *Client*

Les représentations UML et B induites sont présentées dans la figure 5. L'application de cette tactique pour les classes *Société* et *Produit* donne les machines et classes *Société* et *Produit* et les modifications de la machine *Types* de la figure 5 [3].

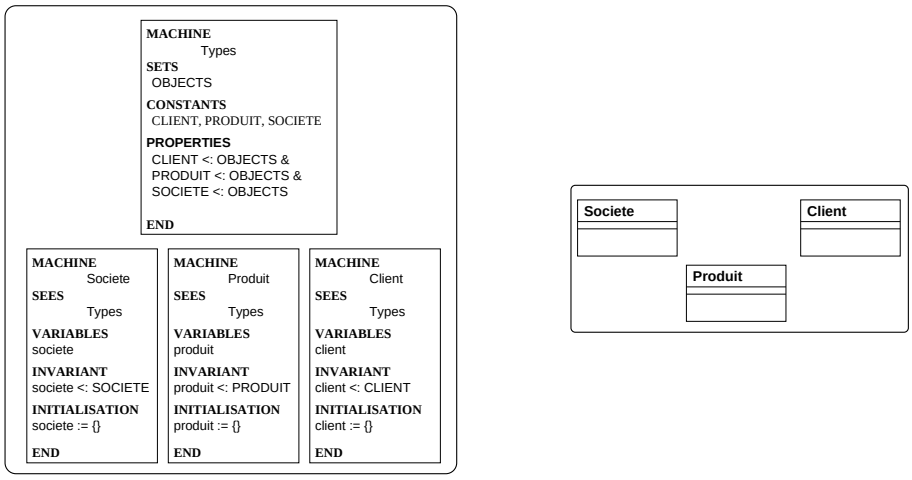

FIG. **5. Instantiation de la tactique d'introduction de composants abstrait**

4.2 Composition de composants B

Description informelle

Cette tactique décrit l'établissement d'un lien logique entre deux composants. A partit d'un composant Ma_1, l'utilisateur doit désigner le composant Ma_2 à lier à Ma_1, la clause de composition B *CL* souhaitée et spécifier si nécessaire une contrainte $\mathcal{I}$ qui définit les conditions de composition entre les ensembles d'instances de classes UML correspondant aux deux composants.

1. Sur la vue B les modifications suivantes sont effectuées :
   (a) introduction d'une clause de composition $CL \in$ {SEES, USES, INCLUDES, EXTENDS}, dans le composant Ma_1 sur lequel la tactique est appliquée,
   (b) spécification d'une contrainte $\mathcal{I}$ de la forme : $v \in ma_1 \sim ma_2$ où
   – $\sim \in \{\rightarrow, \nrightarrow, \leftrightarrow, \twoheadrightarrow, ...\}$,
   – ma_1 et ma_2 sont des ensembles d'instances de classes UML correspondant aux composants Ma_1 et Ma_2,
   – v est une variable modélisée dans Ma_1.
2. Sur la vue UML, les modifications suivantes sont possibles :
   – introduction d'une association entre les classes UML Ma_1 et Ma_2 représentant les ensembles ma_1 et ma_2. L'extrémité de cette association du côté de la classe correspondant à ma_2 possède un nom de rôle qui est celui de v.
   – introduction d'un attribut v de type Ma_2 dans la classe Ma_1 représentant l'ensemble ma_1. C'est le cas où l'association est représentée par un attribut.
   Le type de représentation est déterminé par une interaction avec l'utlisateur.

[3] Dans ce cas, il s'agit d'une variante de cette tactique car la machine *Types* existe déjà

Parameters

- **In**
  - Ma_1 : componentIdentifier
  - Ma_2 : componentIdentifier
  - CL : compositionLink
  - $\mathcal{I}$: constraint

Pre-condition

- Ma_1 et Ma_2 doivent exister,
- il ne doit exister aucun lien de composition entre Ma_1 et Ma_2 au préalable,
- $\forall v \in \mathcal{N}_{Ma_1}, v_i \in \mathcal{N}_{Ma_2} : (Ma_1\ CL\ Ma_2) \Rightarrow (v \neq v_i) \wedge v, v_i \in$ *('a'..'z'|'_')('a'..'z'|'A'..'Z'|'_'|'0'..'9')*$^+$.

Definition

operation de construction	$\mathcal{O}_B$	$\mathcal{O}_{UML}$
newRelation	newCompositionLink *setCompositionLink* *addToLinkComponent* | addInvariant [a]	newAssociation *newAssociationEnds* *addRoleName* *addMultiplicity* *addNavigability* | addAttribute [b]

[a] modélise une contrainte (invariant) entre les ensembles d'instances de classes UML représentant les composants à composer.

[b] utilisée pour représenter une association par un attribut. Cette opération est similaire à celle de la tactique 4.3

FIG. **6. Définition de l'introduction d'une relation entre deux composants B**

La précondition de la tactique indique que :
- les deux entités participantes à la relation doivent exister. Si les deux entités ou l'une d'entre elle n'existe pas, une interaction avec l'utilisateur permettra de les créer pour que la tactique puisse être appliquer,
- il ne doit y avoir qu'une et une seule primitive de composition qui lie les deux composants,
- les identificateurs apparaissant dans deux composants ayant un lien de composition *CL* doivent être deux à deux disjoints. Ces identificateurs sont des chaînes de caractères de *a..Z, 0..9,* _ d'au moins deux caractères et dont le premier caractère appartient à *a..z.*

En UML, un utilisateur peut choisir de représenter une relation par
- une association pour mettre en avant le lien conceptuel entre les classes participantes,
- un attribut pour mettre en avant les choix d'implantation. Ce choix entraîne la perte de la mise en évidence du lien conceptuel entre les deux classes : il faut regarder chacune des classes pour découvrir que parmi leurs attributs un ensemble de pointeurs sur l'une d'entre elles s'y trouve.

Les opérations de construction qui permettent de mettre en oeuvre ces deux cas sont présentées dans la figure 6.

Application de la tactique : composition de composants B

On souhaite établir une relation entre les composants *Produit* et *Client* afin de modéliser les achats effectués par un client dans le but de vérifier si le montant total des achats effectués ne dépasse pas la limite imposée. Ceci implique :

– l'introduction d'une clause de composition (e.g. *USES*, figure 7) entre les machines *Client* et *Produit* dans la machine *Client* pour avoir accès aux données de la machine *Produit* (la variable *produit*, figure 7),
– la modélisation d'une relation *produits_achetes*, entre les ensembles *client* et *produit* d'instances des deux classes afin de pouvoir déterminer pour un client donné, l'ensemble des produits associés.

Le choix de la clause de composition et de la modélisation du type de la relation *produits_achetes* dépend des choix de l'utilisateur et des exigences du système à modéliser. Les représentations UML et B induites par l'instanciation de la tactique 4.2 sont présentées dans la figure 7.

Composition de composants

Parameters

– **In**
  – *Client*
  – *Produit*
  – *USES*
  – *produits_achetes* $\in$ *client* $\leftrightarrow$ *produit*

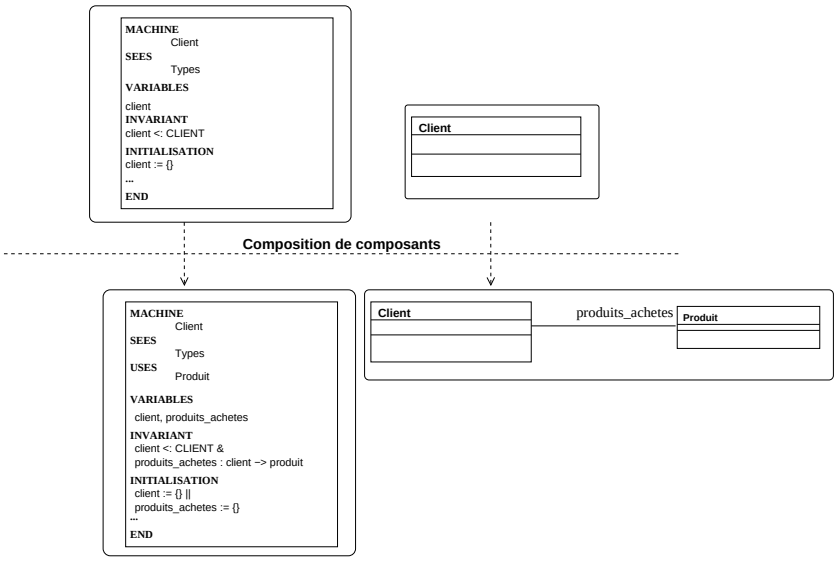

FIG. **7. Introduction d'une relation**

Une représentation UML alternative à celle de la figure 7 serait la représentation de l'association entre les deux classes par un attribut comme illustré dans la figure 8. C'est une variante de la tactique 4.2 qui modélise une association à partir de l'introduction d'une donnée B.

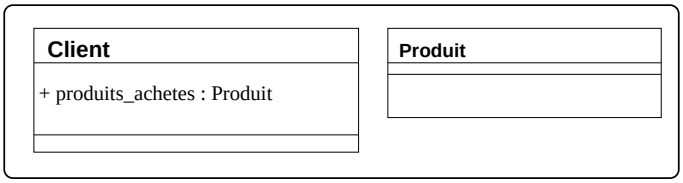

FIG. **8. Représentation d'une association par un attribut**

4.3 Introduction de données

Description informelle

Cette tactique introduit de manière systématique une donnée dans un composant B et un attribut dans une classe UML. L'utilisateur doit désigner le composant *Ma* dans lequel la donnée doit être modélisée, entrer le nom *v*, le type *T* et la valeur initiale *U*[4]. Les changements suivants sont effectués :

1. Sur la vue B, introduction d'une variable ou une constante caractérisée par :
   - son nom *v*,
   - son invariant de typage $\mathcal{I}_T(v)$,
   - sa valeur initiale *U* (pour les variables) ou sa valeur constante *U* (pour les constantes, celle-ci est optionnelle dans les machines abstraites).
2. Sur la vue UML, pour $\mathcal{I}_T(v)$ de la forme : $v \in T$ ou $v \in ma \rightarrow T$, *v* donne lieu à :
   - à un attribut *v* de type *T* et de valeur initiale *U*. Si *v* est modélisé en B comme une constante, l'attribut *v* est munie d'une propriété additionnelle *{frozen}*.

Parameters

- **In**
  - *Ma* : componentIdentifier
  - *v* : dataIdentifier
  - *T* : dataType
  - *U* : dataInitialValue

Pre-condition

- $\forall\, v, v_i \in \mathcal{N}_{Ma} : v \neq v_i \wedge v \in$ *('a'..'z'|'_')('a'..'z'|'A'..'Z'|'_'|'0'..'9')*$^+$

Definition

opération de construction	$\mathcal{O}_B$	$\mathcal{O}_{UML}$
newData	newVariable *addName* *addInvariant*[a] *|addProperty*[b] *addInitialisation*[c]	newAttribute *addName* *addType* *addInitialValue* *addMultiplicity* *addProperty*

[a] valable pour les variables et les constantes
[b] valable pour les constantes
[c] valable pour les variables

FIG. **9. Définition de l'introduction de données**

La précondition de la tactique indique que :
- les identificateurs apparaissant dans un composant B doivent être deux à deux disjoints. Ces identificateurs sont des chaînes de caractères de *a..Z, 0..9,* _ d'au moins deux caractères et dont le premier caractère appartient à *a..z*.

Les attributs d'une classe sont les variables de la machine B correspondante. Ils modélisent l'état de celle-ci. Le type d'un attribut correspond au domaine de valeurs de la variable B correspondante définie par l'invariant de typage de celle-ci. La valeur initiale d'un attribut correspond à l'initialisation définie dans un prédicat de substitution de la variable correspondante. Les opérations qui permettent de les construire sont présentées dans la figure 9.

[4] obligatoire pour les variables et optionnelle pour les constantes. Pour ces dernières, *U* est interprétée comme une valeur constante.

Application de la tactique : introduction de données

Une instanciation de cette tactique est proposée dans la figure 10. Elle illustre l'introduction de la constante *montant_limit* qui modélise le montant total des achats autorisé pour un client donné.

Introduction d'une donnée

Parameters

- **In**
  - *Client* /* nom de la machine qui modélise la donnée */
  - *montant_limit* /* nom de la donnée */
  - *NAT* /* type de la donnée */

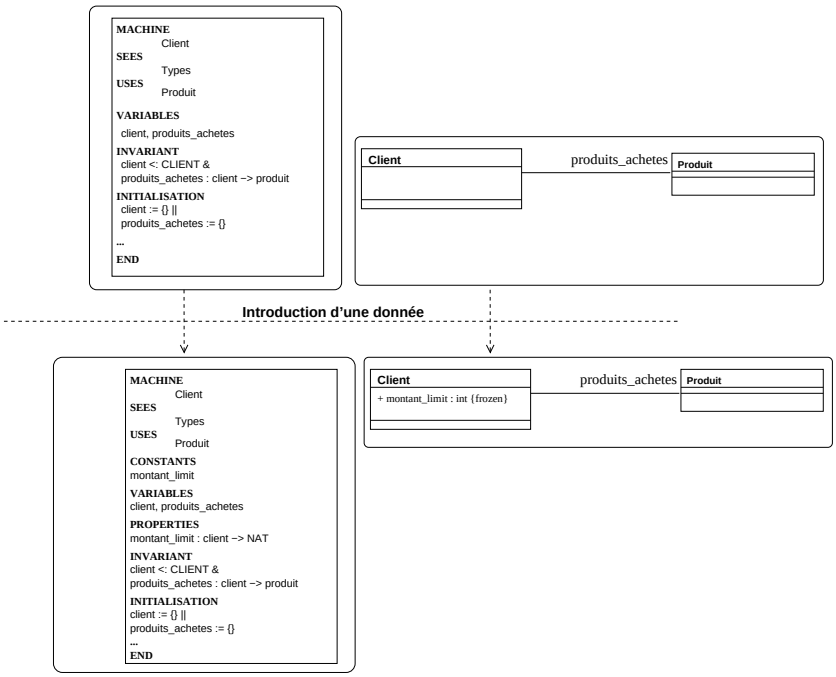

FIG. **10. Introduction d'une donnée**

5 Comparaison avec des approches similaires

Il existe plusieurs approches d'intégration d'UML et B, notamment la génération de spécifications B à partir de diagrammes UML et la génération de diagrammes UML à partir de spécifications B. Une autre concerne l'extension de la notation UML aux concepts B. Cette section présente ces différents courants.

5.1 Intégration UML et B par la génération de spécifications B à partir de diagrammes UML

La première démarche d'intégration d'UML et B s'inscrit dans le domaine de la transformation de formalismes. Elle consiste à générer les squelettes de spécifications B à partir d'un modèle UML et d'annotations OCL [10, 11]. Les spécifications générées sont complétés et raffinés manuellement. L'*atelierB* [21] peut ensuite être utilisé pour analyser les spécifications modifiées pour vérifier les obligations de preuve correspondantes. Les objectifs visés par cette approche sont les suivants :

- 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 de la méthode B.

Après la transformation, une grande partie du travail consiste à compléter et à adapter les squelettes de spécifications B obtenus. Il devient alors intéressant d'étudier les changements causés par l'évolution de la spécification B de telle sorte qu'ils soient toujours cohérents avec les représentations UML initiales. Sur ce point, la démarche de transformation d'UML en B possède une limite majeure :

– Le manque de liens dynamiques entres les représentations UML et B conduit à ce que les modifications opérées sur la spécification B générée ne sont pas prises en compte au niveau UML ; ce qui pose le problème de la cohérence entre les deux représentations.

5.2 Intégration UML et B par la génération de diagrammes UML à partir de spécifications B

Le second courant consiste à considérer UML comme moyen de documentation graphique de spécifications B. Dans cette démarche, l'utilisateur construit entièrement la spécification en B et celle-ci est ensuite traduite en diagrammes UML. Cette approche a été proposée par Bruno Tatibouët et Ahmed Hammad dans [22]. Ces travaux ont donné lieu à un outil, *jBTools* [23], de transformation de machines B en fichier HTML. L'approche reste encore exploratoire. L'approfondissement de cette étude pour traiter des propriétés plus complexes d'un projet B complet (composition de machines, raffinement, implantation) s'avère difficile. Cette difficulté est due au fait que UML et B sont de nature différente, ce qui implique l'existence d'incompatibilités. Dans cette perspective, la spécification UML obtenue par la transformation d'une spécification B complète sera très éloignée d'une spécification telle qu'un spécifieur UML l'aurait conçue.

La transformation de B en UML peut devenir plus délicate encore si, par exemple, il y a d'autres modifications telles que le renommage d'une machine. Dans ce cas, la régénération de B en UML risque de ne pas reconnaître qu'il s'agit de la même machine et construira un modèle UML comprenant deux classes correspondant respectivement à l'ancienne et à la nouvelle machine.

5.3 Intégration UML et B par définition de <<profils>> B pour UML

Cette technique à été proposée dans [20] et vise la définition d'un *«profil»* UML pour B noté *UML-B*. En UML les *«profils»* sont des mécanismes d'extension servant à spécialiser la notation UML pour une utilisation particulière. La technique est intéressante dans la mesure où elle permet de définir une sémantique B depuis UML en définissant des *«tag»* et des *stéréotypes* B pour chaque élément UML. Par exemple, pour définir une classe correspondant à un raffinement B, on écrira *class «Refinement»*. Cette approche présente l'inconvénient de fournir une spécification UML difficile à lire pour les non connaisseurs de B à cause des *stéréotypes* B attribués à chaque élément UML. Il se pose alors le problème de la sémantique objet de ces *stéréotypes*.

5.4 Intégration UML et B par la structuration en vues

Le quatrième courant, que nous proposons, consiste à considérer UML et B comme deux langages de modélisation complémentaires. Dans une telle approche, l'utilisateur ne travaille plus sur deux spécifications indépendantes d'un même système, mais sur une des deux représentations d'une même spécification. Avec une telle approche, l'outil garantit une cohérence totale entre les deux représentations tout au long des modifications et des itérations du processus de construction. Cette démarche est basée sur la synchronisation totale UML/B. Il n'existe aujourd'hui à notre connaissance aucune approche qui vise une intégration dynamique d'UML et B.

Grâce à son aspect dynamique tant du point de vue de la représentation que du point de vue des mécanismes de la construction des raisonnements sur un système, cette démarche est une solution avantageuse pour les concepteurs. Elle pose les bases d'une nouvelle technique d'intégration d'UML et B et constitue une réponse aux problèmes (cohérence, traçabilité) liés à l'évolution individuelle de différents documents de spécification d'un système. Elle sous entend une démarche de construction qui requiert que tout ce qui relève de la description du modèle (par exemple définir une classe, une association entre

deux classes, une machine) soit fait au niveau de la construction et tout ce qui relève de la vérification (preuve) le soit au niveau des outils appropriés.

Lorsque l'on est en phase de modélisation ou que l'on est face à des cas simples de génération, les quatre approches peuvent sembler comparables. Cependant, dès que des itérations entre UML et B interviennent, dès que des incréments de spécification sont introduits, les bénéfices de la construction simultanée apparaissent clairement. La simultanéité dans la construction d'une spécification avec plusieurs formalismes est un facteur important qui facilite l'appropriation d'un système et autorise la validation directe de la modélisation adoptée par l'expert non spécialiste du langage. Une telle dynamique permet de suivre pas à pas les raisonnements réalisés afin de valider les représentations déduites.

6 Conclusion et perspectives

Dans cet article, nous avons présenté des éléments de réflexion sur la caractérisation d'opérations de construction pour définir les tactiques dans la démarche de construction de spécifications multi-vues UML et B. L'idée de combiner différents formalismes pour décrire un système n'est pas nouvelle en informatique. Elle se heurte cependant au problème du maintien de la cohérence entre les représentations générées lorsque des changements interviennent sur telle ou telle représentation. La notion de *"tactique"* et ses mécanismes internes, les *opérations de construction* présentés dans ce papier constituent une réponse à ce problème. Les exemples de tactiques montrent comment les deux notations peuvent *"collaborer"* et être intégrées de façon dynamique. Ce travail constitue un point de départ à la mise en œuvre de mécanismes de modélisation avec plusieurs formalismes.

Les perspectives d'évolution de ce travail sont notamment l'identification et la caractérisation d'un ensemble complet de tactiques utiles à la spécification d'un système complet. Il doit inclure aussi bien les tactiques pour modéliser les aspects statiques que dynamiques. Cet ensemble de tactiques doit ensuite être implanté sur la plateforme *ArgoUML+B*.
Une autre suite importante à ce travail est l'étude de la validation de la cohérence des représentations UML et B produites.

Références

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

[2] B-Core(UK) Ltd, Oxford(UK). *B-Toolkit User's Manual*. 1996.

[3] ClearSy. B4free. Disponible à l'adresse : http ://www.B4free.com/index.php.

[4] J. Souquières D. Okalas Ossami and J-P. Jacquot. Construction de spécifications multi-vues UML et B. *Journal en ligne : Informations, Savoirs, Décisions & Médiations (ISDM)*, February 2004.

[5] S. Dupuy. *Couplage de notations semi-formelles et formelles pour la spécification des systèmes d'information.* Thèse de doctorat, Université Joseph Fourier-Grenoble 1, France, septembre, 2000.

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

[7] R. Laleau and A. Mammar. An overview of a method and its support tool for generating b specifications form UML notations. In *The 15st IEEE Int. Conf. on Automated Software Engineering, Grenoble, France, September 11-15*, 2000.

[8] R. Laleau and F. Polack. Coming and going from UML to B : A proposal to support traceability in rigorous is development. In *ZB'2002 – Formal Specification and Development in Z and B*, pages 517–534, 2002.

[9] H. Ledang. *Traduction Systématique de Spécifications UML vers B.* Thèse de doctorat, LORIA -Université Nancy2, France, novembre, 2002.

[10] H. Ledang and J. Souquières. Derivation Schemes from OCL Expressions to B, LORIA May 2002. Technical report, A02-R-04, 2002.

[11] H. Ledang and J. Souquières. Integration of UML and B Specification Techniques : Systematic Transformation from OCL Expressions into B. In *Proceedings of APSEC 2002 , © IEEE Computer Society, Gold Coast, Queensland, Australia, December 4-6,*, 2002.

[12] H. Ledang, J. Souquières, and S. Charles. ArgoUML+B : Un outil de transformation systématique de spécifications UML vers B. In *Proceedings of AFADL'2003, Rennes, France, January 15-17.* © INRIA, 2003.

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

[14] E. Meyer and J. Souquières. A systematic approach to transform OMT diagrams to a B specification. *FM'99 : World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September*, 1999.

[15] M.Y. Ng and M. Butler. Tool Support for Visualizing CSP in UML. In *George, C. and Miao, H. K., Eds. Proceedings of International Conference on Formal Engineering Methods(ICFEM), Shanghai, China*, pages 287–298, 2002.

[16] H. P. Nguyen. *Dérivation de Spécifications Formelles B à Partir de Spécifications Semi-Formelles.* Thèse de doctorat, Conservatoire National des Arts et Métiers, Paris, France, décembre, 1998.

[17] D. Okalas Ossami, J. Souquières, and J-P. Jacquot. Assistance à la construction de spécifications multi-vues UML et B. poster. In *MAJECSTIC'03. Marseille , France, 29-31 octobre*, 2003. Disponible à l'adresse : http ://www.loria.fr/ okalas.

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

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

[20] C. Snook, M. Butler, and I. Oliver. Towards a UML profile for UML-B. Technical report, DSSE-TR-2003-3, Electronics and Computer Science, University of Southampton, 2003.

[21] STERIA. *Manuel de référence du langage B.* -ClearSy-, novembre, 1998.

[22] B. Tatibouet and A. Hammad. Génération de diagrammes de classes uml à partir de machines abstraites b. In *Actes des Journées d'Informatique pour l'Entreprise, JIEO1'02, Université de Blid, 4-6 mars*, pages 6–17, 2002.

[23] B. Tatibouët. jbtools. 2003.

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

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

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