# Patterns de raffinement – introduction d'une classe intermédiaire : `Class_Helper`

Boulbaba Ben Ammar
Université de Sfax
MIRACL, Sfax, Tunisie
Boulbaba.Ben-Ammar@live.fr

Mohamed Tahar Bhiri
Université de Sfax
MIRACL, Sfax, Tunisie
tahar_bhiri@yahoo.fr

Abdelmajid Ben Hamadou
Université de Sfax
MIRACL, Sfax, Tunisie
abdelmajid.benhamadou@isimsf.rnu.tn

**Résumé**

La spécification de systèmes complexes est une tâche difficile qui ne peut être accomplie en une seule étape. Dans les méthodes formelles, le concept de raffinement a donné lieu à de nombreux travaux dans lesquels la preuve de la correction entre les différents états de spécifications joue un rôle important. Ce papier préconise l'utilisation de la technique de raffinement afin d'établir des modèles UML de qualité c'est-à-dire corrects par construction, extensibles, réutilisables et efficaces. En outre, il plaide en faveur de l'utilisation conjointe UML (semi-formel) et B (formel). Pour cela, nous proposons des patterns de raffinement de diagrammes de classes UML/OCL afin de guider le concepteur lors de la modélisation statique de son application.

***Mots-clés :*** *Raffinement, Pattern de Raffinement, UML, B*

**Abstract**

The specification of complex systems is a difficult task that can not be accomplished in one step. In formal methods, the concept of refinement led to numerous studies in which the proof of the correction between the different states of specifications is important. This paper advocates the use of refinement technique to make UML models of quality that is to say, correct by construction, extensible, reusable and efficient. Furthermore, it advocates the joint use UML (semi-formal) and B (formal). For this, we propose refinement patterns of UML/OCL Class Diagrams to guide the designer when modeling static application.

***Keywords :*** *Refinement, Refinement Pattern, UML, B*

## 1 Introduction

D'une façon informelle, un raffinement est un processus permettant de transformer une spécification abstraite en une spécification concrète. Il vise le développement incrémental de systèmes corrects par construction. Le raffinement est défini d'une façon rigoureuse dans divers langages formels tels que B [1], Event-B, CSP, Z et Object-Z. Le langage UML ne supporte pas le concept du raffinement. Il offre une relation de dépendance stéréotypée «refine» permettant de relier un client (élément raffiné ou concret) à un fournisseur (élément abstrait). Cette relation est sujette à plusieurs interprétations [7] et n'offre pas une assistance méthodologique liée à la manière de raffiner un modèle UML existant. En outre, UML ne permet pas de vérifier si un modèle raffine un autre.

Le sens de raffiner dépend du formalisme utilisé. Par exemple dans la méthode B, raffiner est utilisé comme signifiant: affaiblissement de précondition, réduction de non déterminisme, changement de représentation des variables (raffinage de données) et changement de traitements (raffinage d'opérations). Dans des travaux antérieurs [3, 4], nous avons essayé de donner un sens précis à l'activité de raffinage dans le cadre d'UML. Notre idée centrale est d'identifier des schémas d'évolution généraux dits patterns de raffinement permettant de construire pas-à-pas un

diagramme de classes UML de qualité. Récemment dans [2], nous avons finalisé sept patterns de raffinement pour des diagrammes de classes UML et proposé une démarche de développement de ces diagrammes guidée par des patterns de raffinement. Ces patterns sont construits pour résoudre des problèmes récurrents lors de l'élaboration de la partie statique d'une application OO tels que : introduction d'une classe intermédiaire, réification d'un attribut, décomposition d'un agrégat et introduction d'une nouvelle entité. Dans ce papier, nous nous limitons à la présentation des aspects généraux du pattern : Introduction d'une classe intermédiaire (`Class_Helper`). Celui-ci revisite notre pattern décomposition chaînée (`Linked_Decomposition`) introduit dans [3] aussi bien sur le plan forme (nouveau canevas) que sur le plan fond (`Linked_Decomposition` est un cas particulier de `Class_Helper`).

# 2 Définition d'un pattern de raffinement

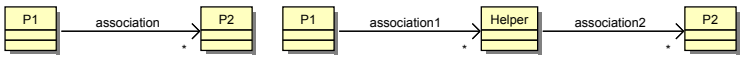
Contrairement aux patterns d'architecture [5], d'analyse et de conception [6], un pattern de raffinement possède un caractère dynamique. Appliqué à une description de niveau i, un pattern de raffinement produit une description de niveau i+1. Récemment, des patterns de raffinement commencent à apparaître visant des formalismes comme Event-B [8], KAOS [10] et B [9]. Un pattern de raffinement possède deux parties : Spécification et Raffinement. La partie Spécification décrit le modèle UML/OCL de niveau i. Et la partie Raffinement décrit le modèle UML/OCL de niveau i+1 produit en appliquant le pattern de raffinement sur le modèle de niveau i. Les patterns de raffinement proposés dans [2] sont décrits selon le même canevas comportant les six rubriques suivantes : Intention, Motivation, Solution, Vérification, Exemple et Voir Aussi. Dans ce qui suit, nous décrivons d'une façon partielle le pattern `Class_Helper` en se limitant aux rubriques : Intention, Solution, Vérification et Exemple.

## 2.1 Intention

Il permet d'introduire une classe **intermédiaire** `Class_Helper` entre deux classes jugées **importantes** vis-à-vis de l'étape de raffinement considérée. La relation directe entre les deux classes importantes est raffinée par un chemin liant ces deux classes en passant par la classe intermédiaire introduite. Le qualificatif "important" évoque un certain ordre hiérarchique dans la modélisation des concepts métier en allant progressivement, du général vers le spécifique.

## 2.2 Solution

Nous nous plaçons dans le cadre suivant : les deux classes principales (ou importantes) `P1` et `P2` sont reliées par une association dirigée de `P1` vers `P2`; les deux autres propriétés (attributs et opérations) de `P1` et `P2` ne sont pas prises en compte. Les deux parties Spécification et Raffinement de ce pattern sont données dans la Figure 1.




(a) Spécification (b) Raffinement

Figure 1: Pattern d'introduction d'une classe intermédiaire : `Class_Helper`

## 2.3 Vérification

La Figure 2 présente la machine abstraite `B_Class_Helper_a` correspondante à la transformation en **B** de la partie Spécification présentée par la Figure 1(a).

**MACHINE**
B_Class_Helper_a
**SETS**
OBJECTS = {p11, p12, p13, p21, p22, p23, h1, h2, h3}
**ABSTRACT_CONSTANTS**
P1, P2
**PROPERTIES**
P1 ⊆ OBJECTS ∧ P2 ⊆ OBJECTS ∧
P1 ∩ P2 = ∅ ∧ P1 = {p11, p12, p13} ∧
P2 = {p21, p22, p23}
**VARIABLES**
p1, p2, association
**INVARIANT**
p1 ⊆ P1 ∧ p2 ⊆ P2 ∧
association ∈ p1 ↔ p2
**INITIALISATION**
p1 := {p11,p12, p13} ||
p2 := {p21, p22, p23} ||
association := {p11↦p21, p11↦p22, p11↦p23, p12↦p21, p12↦p22, p12↦p23, p13↦p21, p13↦p22, p13↦p23}
**END**

Figure 2: Modélisation en **B** de l'état de spécification abstraite

L'ensemble OBJECTS est défini par extension. Il comporte des objets respectivement de type `P1 (p1i)`, `P2 (p2i)` et `Helper (hi)`. De même, les deux constantes abstraites `P1` et `P2` sont définies par extension. Elles mémorisent des objets potentiels de type `P1` et `P2`. Ceci permet d'initialiser (voir opération INITIALISATION) les variables `p1`, `p2` et `association`. Une telle initialisation correspond à un diagramme d'objets issu du diagramme de classes de la partie Spécification du pattern de raffinement proposé (voir Figure 1(a)).

La Figure 3 présente la traduction en **B** de l'état de la partie Raffinement du pattern donnée dans la Figure 1(b).

**REFINEMENT**
B_Class_Helper_r
**REFINES**
B_Class_Helper_a
**ABSTRACT_CONSTANTS**
HELPER
**PROPERTIES**
HELPER ⊆ OBJECTS ∧ HELPER ∩ P1 = ∅ ∧
HELPER ∩ P2 = ∅ ∧ HELPER = {h1, h2, h3}
**ABSTRACT_VARIABLES**
p1, p2, helper, association1, association2
**INVARIANT**
helper ⊆ HELPER ∧
association1 ∈ p1 ↔ helper ∧
association2 ∈ helper ↔ p2 ∧
ran(association1) = dom(association2) ∧
/* *Invariant de collage* */
dom(association) = dom(association1) ∧
ran(association) = ran(association2) ∧
ran(association1) = dom(association2) ∧
association = (association1;association2)
**INITIALISATION**
p1:={p11,p12,p13} || p2:={p21,p22,p23} ||
helper := {h1, h2, h3} ||
association1:={p11↦h1,p11↦h2,p11↦h3, p12↦h1, p12↦h2, p12↦h3, p13↦h1, p13↦h2, p13↦h3} ||
association2:={h1↦p21,h1↦p22,h1↦p23, h2↦p21, h2↦p22, h2↦p23, h3↦p21, h3↦p22, h3↦p23}
**END**

Figure 3: Modélisation en **B** de l'état de spécification concrète

La machine `B_Class_Helper_r` introduit une constante abstraite `HELPER` pour mémoriser les objets potentiels de type `Helper`. La clause PROPERTIES stipule que les constantes `P1`, `P2` et `HELPER` sont deux à deux disjointes (voir $\texttt{P1} \cap \texttt{P2} = \oslash$ venant de la machine `B_Class_Helper_a`). L'état variable de la machine `B_Class_Helper_r` est défini par les cinq variables `p1`, `p2`, `helper`, `association1` et `association2`. Les deux variables `p1` et `p2` proviennent de la machine abstraite `B_Class_Helper_a`. Tandis que les trois autres variables sont introduites par la partie Raffinement du pattern Class_Helper. La variable abstraite `association` a été supprimée. La clause INVARIANT de la machine `B_Class_Helper_r` définit le typage et les contraintes liées aux variables concrètes `helper`, `association1` et `association2`. Egalement, elle comporte l'invariant de collage suivant :

```
dom(association) = dom(association1)        (inv1)
ran(association) = ran(association2)        (inv2)
ran(association1)= dom(association2)        (inv3)
association = (association1;association2)   (inv4)
```

$$(inv\_collage\_Class\_Helper) \quad (1)$$

Cet invariant de collage garantit la correction du pattern de raffinement `Class_Helper`. Il pourrait être instancié et réutilisé avec profit dans un développement conjoint UML/B guidé par des patterns de raffinement.

## 2.4 Exemple

Dans un système de réservation aérienne, les deux classes concepts métier `Personne` et `Avion` peuvent être reliées par l'association `embarquement`. Une spécification pré/post de l'opération `reserver` est également fournie en OCL (voir Figure 4).

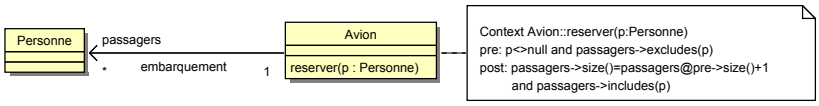



Figure 4: Relation abstraite `embarquement`

### 2.4.1 Application du pattern `Class_Helper`

L'introduction de la notion de Vol est matérialisée par une étape de raffinement en appliquant le pattern de raffinement `Class_Helper`. Ceci est illustré par la Figure 5.

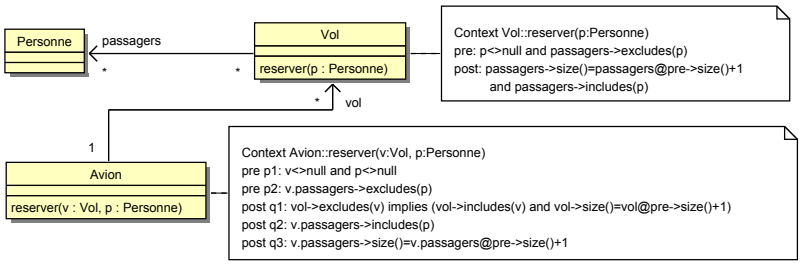



Figure 5: Introduction de la notion de `Vol`

La spécification pré/post en OCL de l'opération `reserver` de la classe `Avion` entraîne celle de l'opération `reserver` de la classe `Vol`. En fait, l'opération `reserver` de la classe `Avion` devrait appliquer `reserver` de la classe `Vol` sur l'occurrence `v` passée comme paramètre de `reserver` de la classe `Avion`.

### 2.4.2 Vérification de l'application du pattern

Afin de formaliser en **B** le raffinage entre la spécification et le raffinement fournis respectivement par les deux FIGURES 4 et 5, nous proposons l'architecture des machines **B** présentée par la Figure 6.

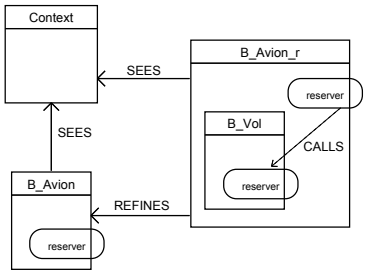



Figure 6: Dépendances des machines `Context`, `B_Avion`, `B_Avion_r` et `B_Vol`

Les deux machines `Context` et `B_Avion` formalisent en **B** la relation abstraite `embarquement` fournie par la Figure 4. La machine `Context` (voir Figure 7) factorise des ensembles abstraits AVION, PERSONNE et VOL utiles pour les deux niveaux : spécification et raffinement.

Quant à la machine `B_Avion` (voir Figure 7), elle regroupe les caractéristiques `reserver` et `passagers` venant du modèle UML de la Figure 4. La sémantique pré/post en OCL de la méthode `reserver` appartenant à la classe `Avion` est préservée par l'opération de même nom appartenant à la machine `B_Avion`.

| | | |
|---|---|---|
| **MACHINE** Context | **SETS** AVION;PERSONNE;VOL | **END** |
| **MACHINE** B_Avion<br>**SEES** Context<br>**VARIABLES** passagers | **INVARIANT** passagers ∈ AVION ⇸ PERSONNE<br>**INITIALISATION** passagers := ∅<br>**OPERATIONS**<br>reserver(av1,pr1) = **PRE** | av1 ∈ AVION ∧ pr1 ∈ PERSONNE ∧ pr1 ∉ ran(passagers) **THEN** passagers(av1) := pr1 **END**<br>**END** |

Figure 7: Modélisation en **B** de l'état de spécification abstraite

Les deux machines `B_Vol` et `B_Avion_r` (voir Figure 8) formalisent en **B** le raffinement de la relation abstraite `embarquement` fourni par la Figure 5. La machine `B_Vol` formalise en **B** la classe `Vol` du modèle UML présenté par la même figure. Elle introduit la variable `vol_personne` (relation) permettant de modéliser l'association entre `Vol` et `Personne`. L'opération `reserver` de la machine `B_Vol` est spécifiée en tenant compte de la sémantique pré/-post en OCL de la méthode `reserver` de la classe `Vol` de la Figure 5. La machine `B_Avion_r` raffine `B_Avion` en incluant une instance de la machine `B_Vol` (voir la clause INCLUDES). Elle introduit la variable `avion_vol` (fonction partielle) permettant de modéliser l'association entre `Avion` et `Vol`. L'invariant de la machine `B_Avion_r` est issu de l'invariant de collage du pattern `Class_Helper` (Inv_Collage_Class_Helper) (1). Les trois variables `association`, `association1` et `association2` sont remplacées respectivement par `passagers`, `avion_vol` et `ff.avion_personne`. Les machines B fournies dans les deux Figures 7 et 8 ont été prouvées grâce au prouveur interactif de l'atelierB.

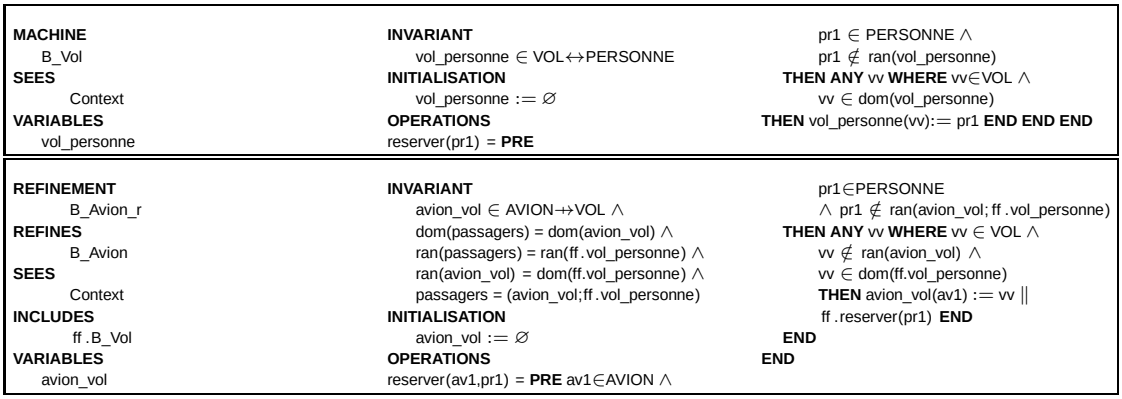



Figure 8: Modélisation en **B** de l'état de spécification concrète

# 3 Conclusion

La technique de raffinement permet l'obtention de logiciels corrects par construction. Mais l'application de cette technique pose des problèmes aussi bien dans un cadre formel que semi-formel. Nous avons apporté une contribution sous la forme d'un catalogue de 7 patterns de raffinement permettant aux spécifieurs UML de tirer profit de la technique de raffinement. Ces patterns apportent des solutions aux problèmes récurrents lors de la construction d'un diagramme de classes UML. Dans ce papier, nous avons décrit d'une façon partielle uniquement le pattern d'insertion d'une classe intermédiaire : `Class_Helper`.

# Références

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

[2] B. Ben Ammar. *Contribution à l'ingénierie des systèmes - Raffinement et Refactoring de spécifications UML.* PhD thesis, Faculté des Sciences Economiques et de Gestion de Sfax, Mai 2012.

[3] B. Ben Ammar, M. T. Bhiri, and J. Souquières. Quelques patrons de raffinement pour le développement de diagrammes de classes UML. In *6ème atelier sur les Objets, Composants et Modèles dans l'ingénierie des Systèmes d'Information, OCM-SI, couplé avec le 15ème congrès INFORSID*, Perros-Guirec France, 2007.

[4] B. Ben Ammar, M. T. Bhiri, and J. Souquières. Incremental development of uml specifications using operation refinements. *Innovations in Systems and Software Engineering*, 4:259–266, 2008.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: a system of patterns*, volume 1. John Wiley and Sons, 1996.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns.* Addison-Wesley, Boston, MA, January 1995.

[7] H. Habrias and C. Stoquer. Une sémantique formelle pour le raffinage en UML. In *XII Colloque National de la Recherche en IUT, CNRIUT'06*, Brest, France, June 2006.

[8] T. S. Hoang, A. Furst, and J. R. Abrial. Event-b patterns and their tool support. *Software Engineering and Formal Methods, International Conference on*, 0:210–219, 2009.

[9] A. Requet. Bart: A tool for automatic refinement. In *ABZ*, page 345, 2008.

[10] A. van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications.* Wiley, 2009.