Le raffinement vu comme primitive de spécification - une comparaison de VDM, B et Specware

Yves Ledru, Catherine Oriat, Marie-Laure Potet
Institut IMAG - Laboratoire Logiciels Systèmes Réseaux
681 rue de la Passerelle, BP 72
38402 Saint Martin d'Hères Cedex,
e-mail : {Yves.Ledru, Catherine.Oriat, Marie-Laure.Potet}@imag.fr

Résumé

La structuration de spécification est surtout perçue au travers de la notion de module. Le raffinement propose un autre type de liaison entre éléments de spécification qui peut également être adapté pour les structurer. Cet article étudie la combinaison de la modularité et des raffinements. Il s'intéresse à trois approches formelles : VDM, B et Specware. VDM propose les raffinements de données et d'opérations; B et Specware proposent un raffinement de modules. Une étude de cas sert de base à cette comparaison.

1 Introduction

La construction de spécifications complexes implique l'existence de primitives de structuration. Parmi les langages de spécification, nous noterons le calcul des schémas dans Z [15], les extensions modulaires de VDM en VVSL [14] et celles proposées par IFAD [5, 7], les modules de RAISE [6], les machines abstraites de B [1].

Cependant la modularité, qui découpe le texte de la spécification en morceaux de taille plus facile à appréhender, n'est pas la seule façon de structurer une spécification. Le raffinement est une primitive complémentaire qui permet de raisonner à plusieurs niveaux d'abstraction et de construire une spécification en intégrant progressivement des précisions pertinentes. Ainsi, dans des domaines d'application où les spécifications sont de préférence déterministes (applications critiques, logiciels de finance,...) le raffinement donne un moyen d'intégrer progressivement des détails qui limiteront la liberté d'implantation tout en éliminant le non-déterminisme.

Ce travail compare trois méthodes qui proposent une notion de raffinement. Les méthodes choisies sont VDM, B et Specware.

- VDM est avec Z et B l'une des méthodes orientées modèles les plus utilisées. La notion de raffinement est intégrée dans la définition originale de la méthode [8] mais n'est pas reprise dans la norme ISO [2]. VDM inclut deux notions de raffinement : sur les structures de données et sur les opérations.
- La méthode B et ses outils proposent un support automatisé de la notion de raffinement. Les obligations de preuve sont engendrées automatiquement et des outils de preuve assistent la démonstration de ces théorèmes. La syntaxe de B permet d'exprimer les raffinements et de combiner modularité et raffinement.

– Specware [1] [16, 17] combine également modularité et raffinements. Basé sur les spécifications algébriques et la théorie des catégories, Specware comporte une représentation graphique qui permet de raisonner sur la structure des spécifications en faisant abstraction du contenu précis des modules.

Dans cette comparaison on s'intéressera plus particulièrement au raffinement et aux point suivants : Quelles notions de raffinement la méthode propose-t-elle? Comment le raffinement est-il exprimé dans la méthode? Comment le raffinement permet-il de structurer un développement?

L'étude de cas est présentée dans la section 2, les sections 3, 4 et 5 spécifient cet exemple dans les trois formalismes et évaluent certains éléments de comparaison. Enfin la section 6 tire les conclusions de cette étude.

2 Etude de cas et points de comparaison

L'étude de cas du transfert bancaire est inspirée du "Barclay's Code of Business Banking" [3] et de connaissances de base du fonctionnement bancaire. A l'origine, elle illustrait le prototypage de spécification dans l'environnement KIDS/VDM [9].

2.1 Spécification la plus abstraite

Il s'agit de spécifier un transfert entre deux comptes bancaires par émission d'un chèque. Un modèle abstrait comprend les comptes bancaires et l'opération de transfert. Ce premier modèle (que nous appellerons $BANK$) est organisé comme une machine abstraite dont l'état met en correspondance des numéros de compte ($Acno$, un type de base [2]), leur solde (bal, un entier) et le découvert admis (od, un naturel). Un invariant impose que $bal \geq -od$. La seule opération disponible à ce niveau exprime le transfert d'un compte vers un autre d'un montant donné ($Transf$, spécifié en VDM à la Sect. 3.1). La précondition de cette opération impose que les comptes émetteur et crédité existent, qu'ils soient différents et que le solde du compte émetteur et le découvert associé permettent de retirer le montant. La postcondition exprime que les soldes des comptes émetteur et crédité sont modifiés du montant du transfert. La Fig. 1 illustre le transfert de 2000 unités entre les comptes 1 et 2; cette figure ne représente pas les découverts.

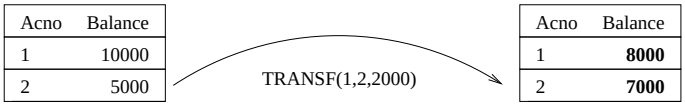

FIG. 1 – *Le niveau le plus abstrait ($BANK$)*

2.2 Raffinement d'opération

Le client de la banque ne connait que la moitié de cette opération : son compte est crédité si il reçoit le chèque, et débité si il l'émet. On peut dès lors raffiner cette opéra-

1. Specware est une marque déposée de Kestrel Development Corporation.
2. Dans les figures qui illustrent cet article, les numéros de compte sont représentés par des naturels.

tion par une séquence de $Credit$ et $Debit$ (Fig. 2). Ces opérations prennent les mêmes paramètres que l'opération de transfert (émetteur, crédité, montant). La précondition de $Debit$ est la même que celle de $Transf$, celle de $Credit$ ne met aucune condition sur le solde du compte émetteur. La postcondition de $Credit$ ajoute le montant au solde du compte crédité tandis que la postcondition de $Debit$ enlève ce montant du solde du compte émetteur. "$Credit; Debit$" et "$Debit; Credit$" sont deux raffinements valides de $Transf$; dans cet exemple, on exécutera $Credit$ avant $Debit$.

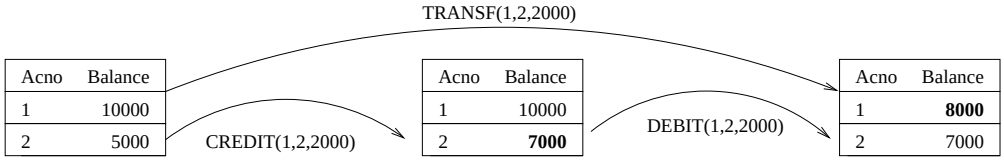

FIG. 2 – *Introduction de $Credit$ et $Debit$ ($BANK$-OD)*

2.3 Raffinement de données

En fait, le transfert du chèque prend du temps. De sorte que même si le solde apparait comme crédité le jour où le client dépose son chèque, il faut quelque temps pour que le chèque soit envoyé à la banque émettrice et que l'émetteur soit effectivement débité. C'est pourquoi la banque distingue le solde du compte et un "solde effectif" ($ebal$, un entier) qui peut être différent du solde. Cette notion suppose un raffinement de la structure de données pour mettre en correspondance chaque compte avec son solde effectif. Les spécifications de $Credit$ et $Debit$ sont modifiées, ainsi, à la Fig. 3, il apparait que l'opération $Credit$ ne modifie aucun solde effectif, mais que $Debit$ modifie les deux soldes. Ici plusieurs combinaisons sont possibles, comme celle où $Credit$ débite le solde effectif et $Debit$ le crédite.

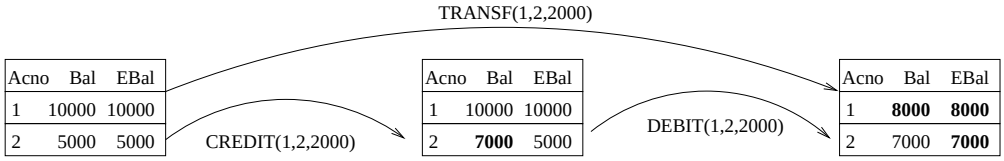

FIG. 3 – *Introduction du solde effectif ($BANK$-OD-DR)*

L'exercice est donc de tracer le développement de cette spécification en trois étapes : BANK, BANK_OD pour la décomposition de l'opération *Transf* et BANK_OD_DR pour l'introduction du solde effectif.

3 VDM

VDM est une méthode orientée modèle où la spécification est structurée en variables d'état et opérations. La spécification des opérations est exprimée en termes de pré- et postconditions.

3.1 Spécification la plus abstraite

La spécification la plus abstraite est inspirée des “teaching notes” de [8]. Dans cette spécification, le solde et le découvert sont regroupés dans un type “composite” $acdata$, l’équivalent d’un “record” en Pascal. La spécification ne comprend qu’une variable d’état (am) qui est une fonction discrète entre les numéros de compte et leur $acdata$. Les pré- et postconditions de $TRANSF$ sont la traduction en VDM de celles exprimées dans la section 2. En VDM, des μ fonctions sont utilisées pour avoir accès aux divers champs du composite.

$$
\begin{array}{ll}
1.0 & TRANSF\ (fr\text{-}ac : acno, to\text{-}ac : acno, a : \mathbf{N}_1) \\
.1 & \textbf{ext wr}\ am : acno \xrightarrow{m} acdata \\
.2 & \textbf{pre}\ fr\text{-}ac \in \textbf{dom}\ am \wedge to\text{-}ac \in \textbf{dom}\ am \wedge fr\text{-}ac \neq to\text{-}ac \wedge \\
.3 & \quad am\ (fr\text{-}ac).bal - a \geq -\ am\ (fr\text{-}ac).od \\
.4 & \textbf{post}\ am = \overleftarrow{am} \dagger \{fr\text{-}ac \mapsto \mu\ (\overleftarrow{am}\ (fr\text{-}ac), bal \mapsto \overleftarrow{am}\ (fr\text{-}ac).bal - a), \\
.5 & \quad to\text{-}ac \mapsto \mu\ (\overleftarrow{am}\ (to\text{-}ac), bal \mapsto \overleftarrow{am}\ (to\text{-}ac).bal + a)\}\ ;
\end{array}
$$

3.2 Raffinements

Deux raffinements sont proposés par VDM [8] : le raffinement de données (“data reification”) et le raffinement d’opérations (“operation decomposition”). Dans la description initiale de la méthode, les obligations de preuve correspondant à ces deux types de raffinement sont décrites précisément. Il faut cependant noter que la norme ISO ne décrit que le langage de spécification et ne propose ni la notion de raffinement ni la modularité [3]. Dans [8], les deux notions de raffinement ne s’expriment pas au même niveau : la décomposition d’opérations est interne à un module alors que le raffinement de données est une relation entre deux modules.

3.3 Etude de cas

La structuration de notre développement est :

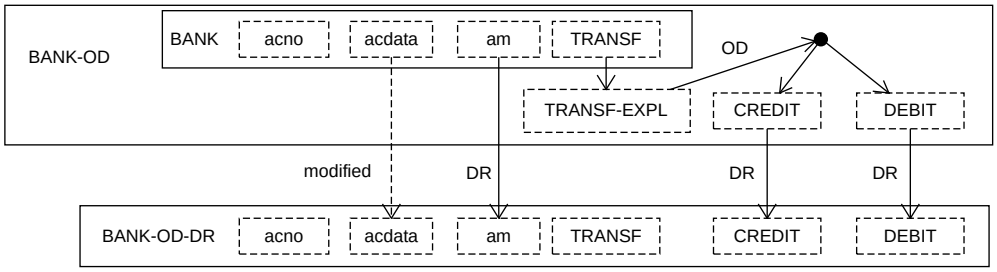

FIG. 4 – *Structure des raffinements VDM*

Premier raffinement : La décomposition d’opération correspond à une extension de la spécification. Dans notre étude de cas, $Credit$ et $Debit$ sont exprimés dans le même

3. Dans ce travail, nous avons utilisé les constructions modulaires offertes par l’environnement IFAD/VDM pour encapsuler les machines abstraites et distinguer les raffinements.

module que $Transf$ (Fig. 4). En fait, toutes ces opérations travaillent sur les mêmes variables d'état. Le raffinement est exprimé au travers d'une version "explicite" de $Transf$ dont la spécification contient du code impératif:

```
2.0  TRANSF-EXPL : acno × acno × N1 →o ()
.1   TRANSF-EXPL (fr-ac, to-ac, a) △
.2     (CREDIT(fr-ac, to-ac, a) ;
.3      DEBIT(fr-ac, to-ac, a) )
```

Pour la décomposition d'opérations, VDM reprend des obligations de preuve similaires à la logique de Hoare. Ici, il s'agit de montrer que le code de $Transf\text{-}expl$ est une implantation de $Transf$.

Second raffinement : Le raffinement de données implique non seulement la spécification de nouvelles variables d'état, mais aussi des nouvelles versions de chaque opération. On peut l'exprimer en définissant un nouveau module dont l'état et les opérations ont les mêmes noms que dans le module abstrait. VDM impose de spécifier le raffinement de structures de données par une fonction d'abstraction ("retrieve function"), totale sur les états concrets, qui associe un état abstrait à tout état concret :

```
3.0  RETR : BANK-OD-DR`Bank → BANK-OD`Bank
.1   RETR (mk-BANK-OD-DR`Bank (am)) △
.2     mk-BANK-OD`Bank ({n ↦ (mk-BANK-OD`acdata (am (n).bal, am (n).od)) |
.3                            n ∈ dom am})
```

Ici, la fonction d'abstraction est une définition en compréhension qui renvoie un état concret où le champ $ebal$ est oublié. Pour le raffinement des données, VDM impose des obligations de preuve sur la fonction d'abstraction et sur les opérations.

- Deux obligations de preuve sont définies sur la fonction d'abstraction. D'une part, celle-ci est totale sur les valeurs possibles de l'état concret. Cela traduit le fait que toute valeur concrète doit avoir une interprétation abstraite. D'autre part, il faut prouver que tout état abstrait a une représentation concrète ("adequacy"). Dans notre étude de cas, cette preuve est simple, il suffit d'associer à chaque état abstrait l'état concret de même solde, de même découvert et dont le solde effectif est, par exemple, une copie du solde.
- Deux obligations de preuve doivent être satisfaites par chaque opération abstraite vis à vis de sa correspondante concrète. D'une part, toutes les valeurs concrètes qui satisfont la précondition abstraite (via la fonction d'abstraction) doivent satisfaire la précondition concrète. En d'autres mots, le domaine d'application des opérations concrètes est "inclus" dans celui des opérations abstraites. D'autre part, les valeurs concrètes qui satisfont la postcondition concrète ont une image, par la fonction d'abstraction, qui satisfait la postcondition abstraite. Une variante de cette seconde obligation de preuve s'applique à la spécification de l'état initial.

4 B

4.1 Raffinements

Trois types de composants existent en B : les machines abstraites (mot clé MACHINE), les raffinements (mot clé REFINEMENT) et les implantations (mot clé IMPLEMENTATION). Les machines abstraites correspondent au premier niveau de spécification. Les raffinements sont des spécifications qui contiennent une référence à la spécification raffinée. Les implantations sont des raffinements particuliers respectant des contraintes qui permettent leur traduction directe dans un langage de programmation. D'autre part les implémentations ne sont pas raffinables, elles ont donc pour effet de figer les algorithmes et les représentations des structures de données. Le raffinement introduit par la méthode B est relatif à une machine abstraite en son entier : son état interne et son jeu d'opérations. La différence avec VDM est que le raffinement des données est *observationnel*, c'est à dire qu'il dépend du jeu d'opérations offert par la spécification. En particulier il n'est pas imposé de représenter toutes les valeurs abstraites. Par exemple la spécification initiale de BANK doit être complétée par un observateur *Consult_bal* qui permet de consulter le solde d'un compte. En l'absence de cet observateur cette spécification pourrait être raffinée par une machine sans état, avec une opération *Transf* ne faisant rien. L'introduction de l'opération *Consult_bal* oblige à représenter les comptes. De la même manière l'opération *Consult_ebal* devra être ajoutée lors de l'introduction du solde effectif.

Les obligations de preuve peuvent être assimilées à celles de VDM si ce n'est qu'il n'est pas imposé de représenter toutes les valeurs abstraites (critère d'observabilité) et que des valeurs abstraites non identiques peuvent être représentée par une même valeur concrète. Il n'y a donc pas d'obligation de preuve propre au choix de la représentation de l'état. Les obligations de preuve associées à l'état initial et aux opérations expriment les mêmes contraintes qu'en VDM, en remplaçant la fonction d'abstraction par l'invariant de liaison.

4.2 Composition

La méthode B offre des primitives permettant de composer des spécifications. On ne s'intéresse ici qu'aux primitives *imports* et *sees* qui introduisent des structurations dans le développement. Les contraintes sur ces primitives sont les suivantes :

- On ne peut bâtir de nouvelles spécifications (machine abstraite, raffinement ou implantation) qu'à partir de machine abstraite. Ceci est lié au fait que les raffinements et les implantations ne sont pas des spécifications autonomes.
- Les définitions partagées entre plusieurs spécifications doivent être localisées dans une machine abstraite. Cette contrainte permet de garantir que les définitions partagées sont implantées d'une manière compatible, puisqu'il n'il y a qu'une seule implantation.
- Les preuves d'invariance et de raffinement doivent être préservées, dans la mesure du possible.

La primitive de composition *imports* permet, dans une implantation, d'utiliser une machine abstraite qui sera implantée indépendamment. Ceci est rendu possible par le

fait que l'état de la machine importée n'est pas visible directement, on ne peut utiliser que les opérations (vue close d'un composant [13]). L'utilisation de la primitive *imports* crée une copie locale de la spécification. C'est la primitive *sees* qui permet de partager des définitions. Lorsque le partage concerne des états cette primitive introduit des contraintes fortes d'utilisation, en particulier pour pouvoir composer les raffinements [10], mais ceci ne nous concernera pas dans l'exemple traité.

4.3 Étude de cas

Le raffinement en B impose la préservation de l'interface du composant : il n'est pas possible d'ajouter des opérations et il n'y a pas de notion d'opérations locales. Il s'ensuit que le second niveau de la spécification (BANK_OD) nécessite une nouvelle machine abstraite afin d'introduire les deux opérations *Credit* et *Debit*. Ceci sera aussi vrai du troisième niveau de la spécification (BANK_OD_DR) où on ajoute une opération permettant d'observer le solde effectif. La spécification sera donc composée des trois machines abstraites suivantes :

- BANK définit les opérations *Transf* et *Consult_bal* sur le solde.
- BANK_2 définit les opérations *Credit*, *Debit* et *Consult_bal* sur le solde.
- BANK_3 définit les opérations *Credit*, *Debit*, *Consult_bal* et *Consult_ebal* sur le solde et le solde effectif.

Ces trois machines abstraites partagent une définition commune, l'ensemble *Acno*. Comme B impose de localiser dans une machine abstraite les définitions partagées, nous introduisons la machine ACCOUNT qui sera vue par toutes les autres machines. Il ne reste plus qu'à introduire les différents niveaux de la spécification. Pour cela nous allons établir les relations de raffinement. Nous avons ici choisi d'utiliser des implantations qui correspondent à la dernière étape de raffinement. Ceci a pour effet de figer, au niveau de la spécification, le corps des opérations (par exemple nous imposons que l'opération de transfert soit implantée sous la forme d'une opération de crédit suivie d'une opération de débit). La structure générale de la spécification est donc [4] :

Premier raffinement : BANK_OD décrit le raffinement des opérations de BANK à l'aide des opérations de BANK_2. Dans ce cas les variables de BANK_2 et les variables de BANK ont les mêmes noms et sont assimilées lors du raffinement. L'invariant implicite est $BANK.bal = BANK\_2.bal \wedge BANK.od = BANK\_2.bal$. L'opération *Transf* est décrite à l'aide de *Credit* et *Debit*. L'opération *Consult_bal* de BANK est assimilée directement à l'opération de même nom de BANK_2 (clause PROMOTES).

IMPLEMENTATION *BANK_OD*
REFINES BANK
SEES ACCOUNT
IMPORTS BANK_2
PROMOTES CONSULT_BAL
OPERATIONS
TRANSF(*fr_ac*, *to_ac*, *val*) =
BEGIN CREDIT(*fr_ac*, *to_ac*, *val*); DEBIT(*fr_ac*, *to_ac*, *val*) END
END

4. Cet exemple a été développé avec l'Atelier B [11].

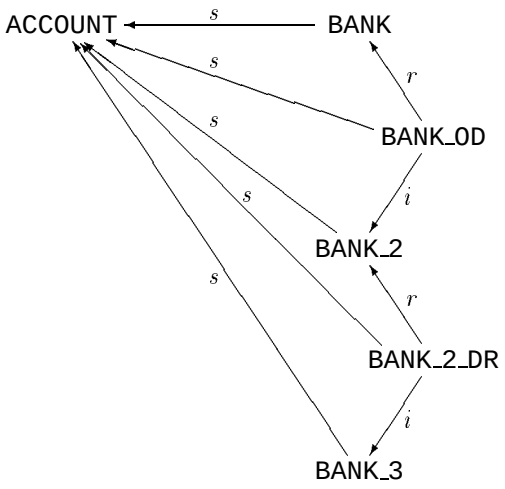

FIG. 5 – *Structure des raffinements B*

Second raffinement : BANK_2_DR décrit le raffinement des opérations *Credit*, *Debit* et *Consult_bal* sur le nouvel état qui contient maintenant le solde effectif. L'invariant de liaison est $BANK\_3.bal = BANK\_2.bal \wedge BANK\_3.od = BANK\_2.bal$, qui a pour effet de ne pas contraindre le solde effectif. Les opérations de BANK_2 sont directement implantées par les opérations de même nom de BANK_3.

IMPLEMENTATION *BANK_2_DR*
REFINES BANK_2
SEES ACCOUNT
IMPORTS BANK_3
PROMOTES CREDIT, DEBIT, CONSULT_BAL
END

La propriété de monotonie du raffinement fait qu'il n'est pas nécessaire de redéfinir l'opération *Transf* en terme des nouvelles définitions des opérations *Credit* et *Debit*. L'implantation BANK_OD_DR peut être produite systématiquement sous la forme :

IMPLEMENTATION *BANK_OD_DR*
REFINES BANK
SEES ACCOUNT
IMPORTS BANK_3
PROMOTES CONSULT_BAL
OPERATIONS
TRANSF(*fr_ac*, *to_ac*, *val*) =
BEGIN CREDIT(*fr_ac*, *to_ac*, *val*); DEBIT(*fr_ac*, *to_ac*, *val*) END
END

5 Specware

Specware [16, 17] est un système basé sur les spécifications algébriques [4, 12, 18] et la théorie des catégories, qui permet d'écrire des spécifications, de les combiner et de les raffiner.

5.1 Spécification abstraite

La spécification DATA regroupe toutes les informations nécessaires aux comptes bancaires, c'est-à-dire les numéros de comptes (*Acno*), les données d'un compte (*Acdata*), et les fonctions associant les comptes aux données $Account\text{-}Map = Acno \rightarrow Acdata$. On accède aux données d'un compte à l'aide de deux opérations $bal{:}Acdata \rightarrow Integer$ et $od : Acdata \rightarrow Nat$. Cette spécification permet de factoriser des données utilisées à plusieurs endroits.

La spécification BANK importe DATA et introduit l'opération de transfert entre deux comptes $transf : Account\text{-}Map, Acno, Acno, Pos \rightarrow Account\text{-}Map$. Dans le cadre algébrique, les spécifications sont purement fonctionnelles. L'opération *transf* doit donc comporter un paramètre supplémentaire de type *Account-Map* en entrée et en sortie pour modéliser le changement d'état. L'opération $transf$ est définie par quatre axiomes (pour simplifier, on ne se préoccupe pas des préconditions) :

$$bal(transf(am, fr\text{-}ac, to\text{-}ac, a)(fr\text{-}ac)) = bal(am(fr\text{-}ac)) - a$$
$$bal(transf(am, fr\text{-}ac, to\text{-}ac, a)(to\text{-}ac)) = bal(am(to\text{-}ac)) + a$$
$$ac \neq fr\text{-}ac \wedge ac \neq to\text{-}ac \Rightarrow bal(transf(am, fr\text{-}ac, to\text{-}ac, a)(ac)) = bal(am(ac))$$
$$od(transf(am, fr\text{-}ac, to\text{-}ac, a)(ac)) = od(am(ac))$$

On spécifie ici $transf$ de façon "observationnelle", par rapport aux sélecteurs bal et od. Nous avons fait ce choix afin de pouvoir raffiner plus facilement les deux spécifications DATA et BANK indépendamment.

5.2 Raffinements

En Specware, le raffinement est basé sur des morphismes de spécifications. Un morphisme de spécifications $m : A \rightarrow B$ de A vers B associe à chaque sorte de A une sorte de B, et à chaque opérateur de A un opérateur de B dont le profil est compatible. D'autre part, les axiomes de A (traduits dans B) doivent être des théorèmes de B.

Le raffinement d'une spécification A (abstraite) vers une spécification C (concrète) est un *lien* entre A et C formé d'une spécification M, appelée *médiateur* et de deux morphismes de spécifications $m_1 : A \rightarrow M$ et $m_2 : C \rightarrow M$. Le morphisme m_2 doit être une extension définitionnelle, ce qui signifie que les classes de modèles de M et C doivent être isomorphes. Intuitivement, le médiateur définit les éléments abstraits en fonction des éléments concrets, et les morphismes m_1 et m_2 la liaison entre les noms abstraits et concrets.

Pour montrer que le médiateur, accompagné des deux morphismes est bien un raffinement, il faut montrer que m_1 et m_2 sont bien des morphismes *de spécifications*, et que m_2 est une extension définitionnelle.

Un exemple simple de raffinement est constitué par un morphisme de spécifications entre A et C. Cela définit bien un raffinement de A vers C, car C peut jouer le rôle du médiateur : on pose $M = C$ et $m_2 = id_C$.

5.3 Étude de cas

On peut raffiner indépendamment les spécifications BANK (raffinement de l'opération $transf$ en $credit$ et $debit$) et DATA (introduction du solde effectif).

Premier raffinement : Pour le raffinement d'opérations, on définit une spécification BANK_2, qui importe DATA et introduit les deux opérations *credit* et *debit*. Le médiateur BANK_OD importe BANK_2 et déclare une opération $transf$ définie à l'aide de *credit* et *debit* par l'axiome :

$$transf(am, fr\text{-}ac, to\text{-}ac, a) = \\ debit(credit(am, fr\text{-}ac, to\text{-}ac, a), fr\text{-}ac, to\text{-}ac, a).$$

Pour que BANK_2 soit un raffinement de BANK, il faut essentiellement montrer que l'inclusion de BANK dans BANK_OD est un morphisme de spécifications, c'est-à-dire que les axiomes de BANK sont des théorèmes de BANK_OD. Cela consiste à montrer, comme en B, que la définition de *transf* en fonction de *credit* et *debit* satisfait bien la spécification initiale de *transf*.

Second raffinement : La spécification DATA_2 importe DATA et introduit l'opérateur $ebal : Acdata \to Integer$. On introduit également un constructeur $make\text{-}acdata : Integer, Natural, Integer \to Acdata$ pour implanter le type *Acdata*, ainsi que les équations correspondantes sur les sélecteurs bal, od et $ebal$. On a un morphisme de spécifications de DATA vers DATA_2 (en l'occurrence une importation), qui constitue un raffinement de DATA vers DATA_2. À ce morphisme correspond un foncteur d'oubli des modèles de DATA_2 vers les modèles de DATA, qui consiste à oublier les opérateurs *ebal* et *make-acdata*. L'oubli de *ebal* correspond à la fonction d'abstraction en VDM.

On peut ici combiner les deux raffinements, en assemblant DATA_2 et BANK_2 (DATA étant partagée) par une somme amalgamée [5]. La spécification obtenue, BANK_2', contient BANK_2 et et les éléments introduits dans DATA_2. La seule contrainte imposée sur *ebal* à ce niveau est son action sur le constructeur $make\text{-}acdata$.

Cette spécification peut être raffinée en BANK_3 : BANK_3 importe DATA_2, et redéfinit *credit* et *debit* à l'aide des équations (on n'a pas indiqué le comportement sur les comptes autres que $fr\text{-}ac$ et $to\text{-}ac$) :

$$credit(am, fr\text{-}ac, to\text{-}ac, a)(fr\text{-}ac) \\ = \quad make\text{-}acdata(bal(am(fr\text{-}ac)), od(am(fr\text{-}ac)), ebal(am(fr\text{-}ac))) \\ credit(am, fr\text{-}ac, to\text{-}ac, a)(to\text{-}ac) \\ = \quad make\text{-}acdata(bal(am(to\text{-}ac)) + a, od(am(to\text{-}ac)), ebal(am(to\text{-}ac))) \\ debit(am, fr\text{-}ac, to\text{-}ac, a)(fr\text{-}ac) \\ = \quad make\text{-}acdata(bal(am(fr\text{-}ac)) - a, od(am(fr\text{-}ac)), ebal(am(fr\text{-}ac)) - a) \\ debit(am, fr\text{-}ac, to\text{-}ac, a)(to\text{-}ac) \\ = \quad make\text{-}acdata(bal(am(to\text{-}ac)), od(am(to\text{-}ac)), ebal(am(to\text{-}ac)) + a)$$

Pour montrer qu'on a bien un raffinement de BANK_2' vers BANK_3, on doit montrer qu'on a bien un morphisme de spécifications de BANK_2' vers BANK_3, autrement dit que les définitions concrètes de *credit* et *debit* dans BANK_3 sont compatibles avec les définitions abstraites de BANK_2'.

La structure générale du développement est résumée Fig. 6. Les flèches simples sont des morphismes de spécifications, les importations sont étiquetées par i et les extensions définitionnelles par d. Les flèches doubles $\Downarrow$ sont les liens de raffinement.

5. *pushout* en anglais

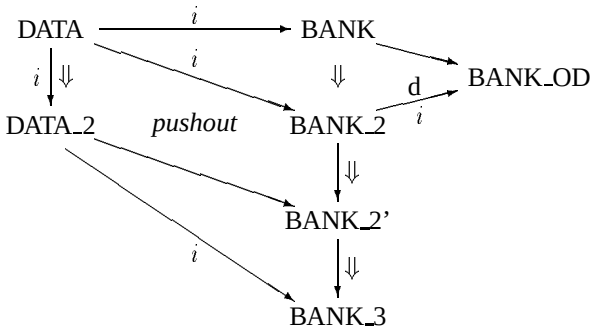

FIG. 6 – *Structure des raffinements en Specware*

Par composition des morphismes de spécifications, on a un raffinement de BANK_2 vers BANK_3, qui correspond au raffinement BANK_2_DR en B.

Une spécification correspondant au raffinement BANK_OD_DR en B peut être construite explicitement en Specware par une somme amalgamée de BANK_OD et BANK_3 en partageant BANK_2 :

```
spec BANK_OD_DR is colimit of diagram
   nodes BANK_2, BANK_OD, BANK_3
   arcs BANK_2 -> BANK_OD : import-morphism
        BANK_2 -> BANK_3 : {}
end-diagram.
```

6 Conclusions

6.1 Notion de raffinement

Bien que VDM n'offre pas de notion de module, les deux formes de raffinement (décomposition des opérations et représentation des données) permettent de raffiner un état interne et les opérations associées, offrant ainsi une forme de raffinement proche de celle de B (au critère d'observabilité près). Distinguer différentes formes de raffinement en Specware n'a pas de sens : on ne fait qu'exprimer des contraintes sur les fonctions. Par contre Specware permet de raffiner avec un mécanisme proche de ce qui est fait en B (redéfinition des opérations sur les nouvelles représentations). La différence essentielle est que, en B, le raffinement porte sur un état interne (principe d'encapsulation), alors que Specware permet de raffiner des types. Il s'ensuit que l'évaluation d'un terme, utilisant les définitions d'un raffinement donné, doit faire intervenir les différents liens existant entre les spécifications afin de convertir les valeurs abstraites en valeurs concrètes et réciproquement.

Par contre, dans le raffinement offert par VDM ou B, les interfaces ne sont pas modifiées puisque les types ne sont pas raffinables. Il suffit donc d'exécuter la même interface en utilisant directement le module raffiné. Les liens de raffinement ne sont plus utiles lors du calcul.

6.2 Représentation du raffinement

Dans le cadre du raffinement de modules, une relation de raffinement doit nommer les spécifications mises en jeu et décrire les opérations de la spécification abstraite en termes des opérations de la spécification concrète. B et Specware offrent des similitudes sur la manière dont les raffinements sont représentés. Rappelons que, dans le cas général, Specware établit un lien entre une spécification abstraite et une spécification concrète à l'aide d'un médiateur et de deux morphismes. De la même manière on considère en B des raffinements mettant en jeu une spécification abstraite, une spécification concrète et un composant de raffinement R qui importe (ou inclut) la spécification concrète et raffine la spécification abstraite.

- Dans les deux cas, la définition des opérations se trouve indifféremment dans la spécification concrète ou dans le composant établissant le lien (le médiateur et les morphismes pour Specware et le composant R pour B). En cas de renommage, le raffinement en B est plus contraignant. Les noms abstraits et concrets doivent être identiques. Dans le cas des opérations, il est néanmoins possible de simuler un renommage en introduisant un appel explicite de l'opération concrète.
- Dans Specware, le lien entre les spécifications est introduit par les flèches qui correspondent à des morphismes de spécifications. Dans la méthode B le lien entre les spécifications est introduit de manière syntaxique, dans le composant de raffinement, en nommant la spécification raffinée et la spécification importée.

Néanmoins deux différences peuvent être pointées. La première différence est que le composant de raffinement n'est pas une spécification au sens de B. Un raffinement est un delta par rapport à la spécification de niveau supérieur. Ceci permet de ne pas tout redéfinir : les préconditions peuvent être omises, les opérations dont la définition est toujours valide n'ont pas besoin d'être redéfinies et les invariants découlant des invariants abstraits sont aussi superflus. Il s'ensuit une simplification des preuves à faire.

La seconde différence, plus importante, est que dans la méthode B les raffinements ne sont pas manipulables puisqu'ils ne correspondent pas à une entité indépendante, ni les relations entre composants.

Par exemple, si nous désirons prouver des propriétés sur l'opération de transfert, exprimée en termes des opérations de crédit et débit sur le solde effectif, (par exemple la propriété d'identité entre le solde et le solde effectif), sans modifier ce qui à déjà été fait, il faut écrire à la main un composant correspondant à BANK_OD_DR. Par contre, en Specware, nous disposons d'une description de la construction de cette spécification (spécification BANK_OD_DR, donnée précédemment).

Ceci constitue une différence importante entre les deux méthodes. Dans B les liens entre composants (importation, raffinement) sont représentés syntaxiquement mais ne sont pas manipulables à l'intérieur de la méthode. Dans Specware, les flèches (avec leurs propriétés) font partie intégrante du langage : on dispose de construction syntaxiques pour les décrire, les nommer ou utiliser leurs propriétés.

Par exemple si DATA contient une équation (comme l'invariant attendu entre le solde et le découvert autorisé) alors, lors de la preuve du raffinement de BANK, il faut montrer que cette équation est valide dans le médiateur BANK_OD. Or cette même

équation a été recopiée dans BANK_OD. Un tel raisonnement peut être décrit : en termes catégoriels ceci revient à montrer la commutation d'un diagramme, propriété décidable dans ce cas.

6.3 En conclusion ...

L'utilisation des méthodes formelles nécessite de pouvoir les appliquer à grande échelle et donc de maîtriser la complexité de l'écriture, des preuves et des développements. Pour cela les langages et méthodes doivent offrir des constructions permettant cette maîtrise. Ceci est un problème ouvert et ce travail s'inscrit dans cette perspective.

Remerciements Nous remercions Y. Srinivas qui nous a aidés à développer une première version de la spécification Specware. Nous tenons également à remercier les rapporteurs anonymes de la conférence AFADL pour leurs commentaires sur la version initiale de cet article. L'énoncé original de l'étude de cas nous a été fourni par A. Finkelstein et M. Feather.

Références

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

[2] D.J. Andrews, H. Bruun, B.S. Hansen, P.G. Larsen, N. Plat, et al. *Information Technology — Programming Languages, their environments and system software interfaces — Vienna Development Method-Specification Language Part 1: Base language*. ISO, 1995.

[3] B̃arclays Bank PLC. The Barclays code of business banking. Barclays Bank, Commercial Banking Division, 1991.

[4] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, 1985.

[5] R. Elmstrom, P. G. Larsen, and P. B. Lassen. The IFAD VDM-SL toolbox: a practical approach to formal specifications. *ACM SIGPLAN Notices*, 29(9):77–80, 1994.

[6] RAISE Language Group. *The RAISE Specification Language*. Prentice Hall - BCS Practioner series, 1992.

[7] The VDM-SL Tool Group. *Users Manual for the IFAD VDM-SL Toolbox.* IFAD, Forskerparken 10, 5230 Odense M, Denmark, February 1994. IFAD-VDM-4.

[8] C. B. Jones. *Systematic Software Development Using VDM (Second Edition)*. Prentice-Hall, London, 1990.

[9] Y. Ledru. Specification and animation of a bank transfer using KIDS/VDM. *Automated Software Engineering*, 4:33–51, 1997.

[10] Y. Rouzaud M.-L. Potet. Composition and refinement in the B-method. In *Proceedings of the Second International B Conference*, volume 1393 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[11] Steria Méditerranée. *Le Langage B. Manuel de référence version 1.5*. S.A.V. Steria, BP 16000, 13791 Aix-en-Provence cedex 3, France.

[12] J. Meseguer and J. A. Goguen. Initiality, induction, and computability. In M. Nivat and J. C. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 14, pages 459–541. Cambridge University Press, 1985.

[13] B. Meyer. *Object-Oriented Construction*. Prentice-Hall, 1988.

[14] C.A. Middelburg. *Logic and Specification: Extending VDM-SL for advanced formal specification*. Chapman and Hall, 1993.

[15] J.M. Spivey. *The Z notation - A Reference Manual (Second Edition)*. Prentice Hall, 1992.

[16] Y.V. Srinivas and R. Jüllig. Specware(TM): Formal support for composing software. Technical Report KES.U.94.5, Kestrel Institute, december 1994.

[17] Y.V. Srinivas and R. Jüllig. *Specware Language Manual*. Kestrel Institute, November 1995.

[18] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 13, pages 677–788. Elsevier Science Publishers B.V., 1990.