Des cas d'utilisation à une spécification B

Hung Ledang

DES CAS D'UTILISATION À UNE SPÉCIFICATION B

Hung LEDANG

LORIA - Université Nancy 2 - UMR 7503
Campus scientifique - BP 239
54506 Vandœuvre-lès-Nancy Cedex - France
Email : ledang@loria.fr

Résumé

Cet article présente une approche pour développer une spécification B à partir d'un modèle des cas d'utilisation d'un système à construire. En exploitant la structuration du modèle des cas d'utilisation ainsi que sa complémentarité avec le modèle des classes du domaine d'application du même système, on propose de construire des machines abstraites B dont les opérations modélisent les cas d'utilisation; la spécification B pour un modèle des cas d'utilisation se compose des machines abstraites et l'implémentation correspondante pour les cas d'utilisation ainsi que les machines abstraites pour les classes et leurs associations.

Notre proposition a un double avantage : d'une part, elle fournit un cadre pour analyser formellement la cohérence du modèle des besoins produit dans un processus de développement par objets; d'autre part, elle permet d'intégrér la méthode B dès la phase d'élicitation des besoins dans un processus de développement de logiciels.

Mots clés

Cas d'utilisation, spécification B, machine abstraite, opération B.

1 Introduction

Dans un processus de développement par objets, on commence souvent par la construction du modèle des cas d'utilisation et du modèle des classes du domaine d'application [JCJO93, Gli00, Rum94]. Le modèle des cas d'utilisation représente les besoins relatifs aux fonctionnalités du système, tandis que le modèle des classes du domaine décrit les aspects structurels du système qui ont un équivalent direct dans le domaine du problème considéré. Les deux modèles ci-dessus visent à constituer un modèle unique - le modèle des besoins.

Le fait de développer parallèlement le modèle des cas d'utilisation et le modèle des classes nécessite de s'assurer de la conformité de ces deux modèles entre eux. L'utilisation de techniques formelles permet d'exprimer cette conformité. La methode B [Abr96] a été choisie pour cette étude parce qu'elle couvre un processus de développement prouvé des logiciels à partir de la spécifi cation abstraite jusqu'à l'implémentation exécutable. Cela est de plus approprié avec la disponibilité des outils support puissants de B (AtelierB [STE98], B-Toolkit [B-C96]). Par ailleurs, l'intégration objet-B permet de donner lieu à une approche à la fois pratique (par objets) et prouvée (à l'aide de B) pour développer des logiciels.

Dans cet article on propose une approche pour intégrer à la fois le modèle des cas d'utilisation et le modèle des classes du domaine d'application dans une spécifi cation B unique. Pour cela, on utilise les schémas de dérivations objet-B de [Mey01] relatifs au modèle des classes pour traduire en B le modèle des classes du domaine d'application. En ce qui concerne les cas d'utilisation, on exploite le fait que

les cas d'utilisation sont décomposés en sous cas d'utilisation qui peuvent être des opérations de base des classes dans le modèle de classes du domaine d'application du système. Chaque cas d'utilisation donne lieu à au moins une opération B. L'opération B d'un cas d'utilisation "père" est implantée par les opérations B de ses cas d'utilisation "fils". La transcription commence avec les cas d'utilisation au niveau du système, c'est-à-dire les cas d'utilisation au niveau de "user goal" dans la proposition de Cockburn [Coc97, Coc00]. Elle se termine lorsque les machines abstraites des classes et de leurs associations sont intégrées dans la spécification B.

L'article est structuré comme suit : la section 2 présente le modèle des cas d'utilisation et le modèle des classes du domaine d'application du système de contrôle d'accès à un ensemble de bâtiments [LPJ00] qui sert comme étude de cas tout au long de la présentation; la section 3 décrit intuitivement nos idées pour modéliser un modèle des cas d'utilisation. Une discussion relative à l'approche est présentée dans la section 4.

2 Étude de cas - le système de contrôle d'accès

Dans cette section, à titre d'exemple pour les sections suivantes, on présente le modèle des cas d'utilisation et le modèle des classes du domaine d'application du système de contrôle d'accès [LPJ00]. On propose de traiter l'exemple dans le cas où le système fonctionne en mode normal et non avec le cas où l'incendie peut se produire à n'importe quel moment.

2.1 Modèle des cas d'utilisation

Un *cas d'utilisation* est une manière spécifique d'exprimer l'utilisation d'un système. C'est l'image d'une fonctionnalité du système, déclenchée en réponse à la stimulation d'un utilisateur ou d'un système externe (*acteur*). L'ensemble des cas d'utilisation spécifie la fonctionnalité complète du système. Dans le formalisme graphique d'UML [RJB98] les cas d'utilisation sont décrits à l'aide de *diagrammes des cas d'utilisation*; le comportement d'un cas d'utilisation est souvent décrit par des textes structurés. La forme textuelle est largement admise grâce à sa capacité d'expression et la facilité de la structuration des cas d'utilisation [Coc97, Coo99, Coc00].

Un cas d'utilisation "complexe" peut être décomposé en sous cas d'utilisation; cela permet de détailler les cas d'utilisation afin d'associer au comportement des cas d'utilisation les opérations de bases de classes reliées. Un cas d'utilisation "père" peut se connecter à ses "fils" par deux types de relations; la *relation d'utilisation* - stéréotypée par «`uses`» - signifie que le comportement du cas d'utilisation "père" comprend également le comportement décrit dans le cas d'utilisation "fils"; la *relation d'extension* - stéréotypée par «`extends`» - signifie que le cas d'utilisation "fils" étend le comportement du cas d'utilisation "père" sous certaines *conditions d'extension*. Il est à noter que notre approche de formalisation des cas d'utilisation en B ne peut pas traiter les cas où cette condition correspond à un événement qui peut se produire à n'importe quel moment durant l'exécution du cas d'utilisation "père" [1].

Dans l'exemple du système de contrôle d'accès, il y a un seul acteur **Utilisateur**; il s'agit de la personne qui souhaite entrer ou sortir d'un bâtiment. Il y a aussi un cas d'utilisation unique **gérer Passage (Gp)** correspondant au protocole de gestion de passage. La description de ce cas d'utilisation est une ré-écriture du protocole de passage décrit dans le cahier des charges.

Nom : *gérer Passage*
Paramètre : `Un utilisateur se présente à un lecteur, muni d'une carte d'accès.`

1. Cela vient de limites de B vis à vis de la modélisation de l'exception et de l'interruption.

Pré-condition :

Le lecteur est libre; ça signifie que le voyant rouge et le voyant vert du lecteur sont éteints; la porte correspondante est dans l'état bloqué.

Le scénario de base :

1. Le système vérifie si l'accès est autorisé.
2. Si l'accès est autorisé, alors
   3. L'utilisateur retire sa carte du lecteur.
   4. Le système allume le voyant vert.
   5. Le système débloque la porte.
   6. Si la porte est franchie (l'utilisateur passe par la porte), alors
      7. Le système met à jour le journal de passage.
      8. Le système bloque la porte.
      9. Le système éteind le voyant vert.

Post-condition

Le lecteur en question redevient libre.

Le scénario de remplacement : *l'accès est refusé*

Recopier l'étape 1 du scénario de base.
Condition d'occurrence : l'accès est refusé
Action :

1. Le système allume le voyant rouge pendant 2 secondes.
2. L'utilisateur retire sa carte.

Le scénario de remplacement : *la porte n'est pas franchie*

Recopier les étapes 1-5 du scénario de base.
Condition d'occurrence : timeout
Action :

1. Le système bloque la porte.
2. Le système éteind le voyant vert.

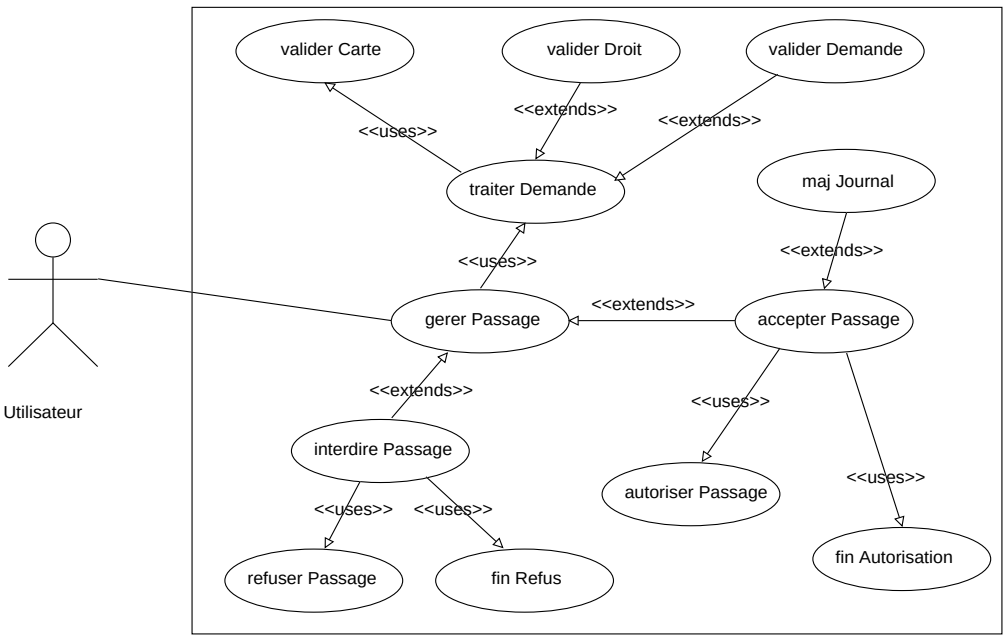

FIG. 1 – ***Structuration du cas d'utilisation "gérer Passage"***

La figure 1 montre la structuration du cas d'utilisation **gérer Passage**; on le décompose en trois sous cas d'utilisation; il s'agit de :

traiter Demande (Td), qui correspond à l'étape 1 dans le scénario de base;

accepter Passage (Ap), qui correspond aux étapes 3 à 9 dans le scénario de base et au scénario de remplacement **la porte n'est pas franchie** ;

interdire Passage (Ip), qui correspond au scénario de remplacement **l'accès est refusé**.

Td s'occupe de l'activité de vérification de la demande de passage. **Ap** réalise les travaux du système dans le cas où l'accès est accepté et **Ip** dans le cas où l'accès est réfusé; de plus, **Td** doit faire partie de **Gp** et selon le résultat de **Td**, **Ap** ou **Ip** sera incorporé dans **Gp**. On dit que **Td** est inclus dans **Gp**, tandis que **Ap** et **Ip** étendent **Gp**.

La décomposition est aussi faite pour les trois cas d'utilisation **Td**, **Ap** et **Ip**. **Td** est décomposé en trois sous cas d'utilisation : **valider Carte**, **valider Droit** et **valider Demande**; **Ap** est décomposé en trois sous cas d'utilisation : **autoriser Passage**, **fin Autorisation** et **maj Journal** (mettre à jour journal); **Ip** est décomposé en **refuser Passage** et **fin Refus**. En l'état on juge que tous les sous cas d'utilisation les plus fins sont attribués aux opérations de base des classes du domaine d'application du système (voir figure 2). La décomposition est terminée.

2.2 Modèle des classes

La figure 2 montre le squelette des classes du domaine d'application du système de contrôle d'accès; il est constitué des classes **Carte**, **Groupe**, **Batiment**, **Enregistrement**, **Lecteur**, **Porte** et **Sas**. Une **Carte** a une identification, un état selon lequel on peut constater que la carte est validée ou non. Un **Groupe** ou un **Bâtiment** possède seulement un nom servant d'identification. Un **Enregistrement** comporte des informations concernant un passage : l'identification de carte, l'identification de bâtiment, la direction du passage et l'heure où le passage a eu lieu. Ici, on ignore l'entité personne qui possède la carte; la justification vient du fait qu'une personne dispose d'une carte; donc on traite la demande de passage avec la carte et non avec la personne. Dans ce cas une carte sera affiliée à un groupe. Entre **Groupe** et **Bâtiment** il y a une relation qui décrit quel groupe a droit d'accès à quel bâtiment; cette relation se traduit par une classe d'association **GroupeBatiment** entre **Groupe** et **Batiment**. Un **Lecteur** est muni de deux voyants : un vert et un rouge qui peuvent être allumés ou éteints. L'état d'une **Porte** est bloquée ou débloquée. Une porte bloqué doit être fermée, une porte débloqué peut être fermée ou non. Un **Sas** est un point d'entrée ou de sortie et se compose d'un lecteur, d'une porte et du sens de passage (entrée ou sortie).

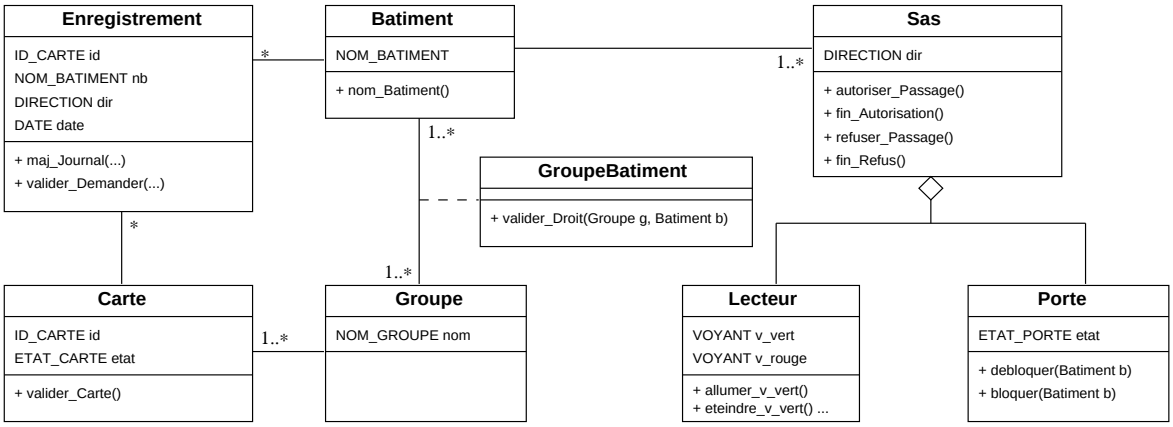

FIG. 2 – ***Squelette des classes du système de contrôle d'accès***

3 Transcrire les cas d'utilisation vers B

3.1 Modéliser un cas d'utilisation en B

Chaque cas d'utilisation est modélisé par une opération B; les paramètres (s'il y en a) des cas d'utilisation sont modélisés comme paramètres des opérations B; le corps de l'opération B modélise les pré-conditions, les post-conditions et l'effet du cas d'utilisation. L'opération B d'un cas d'utilisation est regroupée avec les données dérivées des classes associées au cas d'utilisation. La figure 3 montre un exemple de l'opération B *gerer_Passage* pour le cas d'utilisation **gérer Passage**. Dans la partie déclaration des données de la machine B associée on peut voir les ensembles B, les variables B qui sont dérivées des classes dans la figure 2. La pré-condition del'opération *gerer_Passage* modélise bien la pré-condition informelle dans la description textuelle de **gérer Passage**. La substitution **choice ... or ... end** dans le corps de *gerer_Passage* modélise l'effet du cas d'utilisation; il s'agit de créer un nouvel enregistrement dans le journal de passage dans le cas où le passage a été effectué; dans le cas contraire, aucun changement par rapport à l'état précédent n'a été effectué.

MACHINE *UseCase1*
SETS
...
CARTE;ENREGISTREMENT;BATIMENT;GROUPE;SAS;
LECTEUR;PORTE;ID_CARTE;NOM_BATIMENT;
NOM_GROUPE;ETAT_CARTE;ETAT_PORTE;DIRECTION;
VOYANT={allume,eteint}
VARIABLES
...
carte,enregistrement,batiment,groupe,sas,lecteur,porte,
carte_id,carte_etat,enregistrement_id,enregistrement_nb,
enregistrement_dir,enregistrement_date,batiment_nom,
groupe_nom,sas_dir,lecteur_v_vert,lecteur_v_rouge,
porte_etat
INVARIANT
...
$carte \subseteq CARTE \wedge \ldots \wedge$
$lecteur\_v\_vert \in lecteur \longrightarrow VOYANT \wedge$
...

INITIALISATION
...

OPERATIONS
gerer_Passage(*ll*,*cc*) =
pre
$ll \in lecteur \wedge cc \in carte \wedge$
$lecteur\_v\_vert(ll) = eteint \wedge$
$lecteur\_v\_rouge(ll) = eteint$
then
choice
skip
or
/* *créer un enregistrement* */
end
end

END

FIG. 3 – ***Opération B du cas d'utilisation "gérer Passage"***

3.2 Développer une spécification B pour un modèle des cas d'utilisation

3.2.1 Démarche générale

À partir du modèle des cas d'utilisation au niveau du système on crée une machine abstraite : *UseCase1*; les données dans *UseCase1* sont dérivées du modèle de classes en appliquant les règles proposées par [Mey01]; les opérations dans *UseCase* modélisent les cas d'utilisation au niveau du système; les sous cas d'utilisation des cas d'utilisation au niveau du système donnent lieu à une autre machine abstraite *UseCase2*. Afin d'utiliser *UseCase2* pour implanter *UseCase1* il est souvent nécessaire d'ajouter des opérations utilitaires servant d'articulation aux opérations des cas d'utilisation. Il s'agit des opérations de base des classes dans le modèle des classes du domaine d'application qui sont reliées au modèle des cas d'utilisation en question.

Dans la figure 4, la machine *UseCase2* contient les opérations *traiter_Demande*, *accepter_Passage* et *interdire_Passage* pour les trois sous cas d'utilisation **traiter Demande**, **accepter Passage** et **interdire**

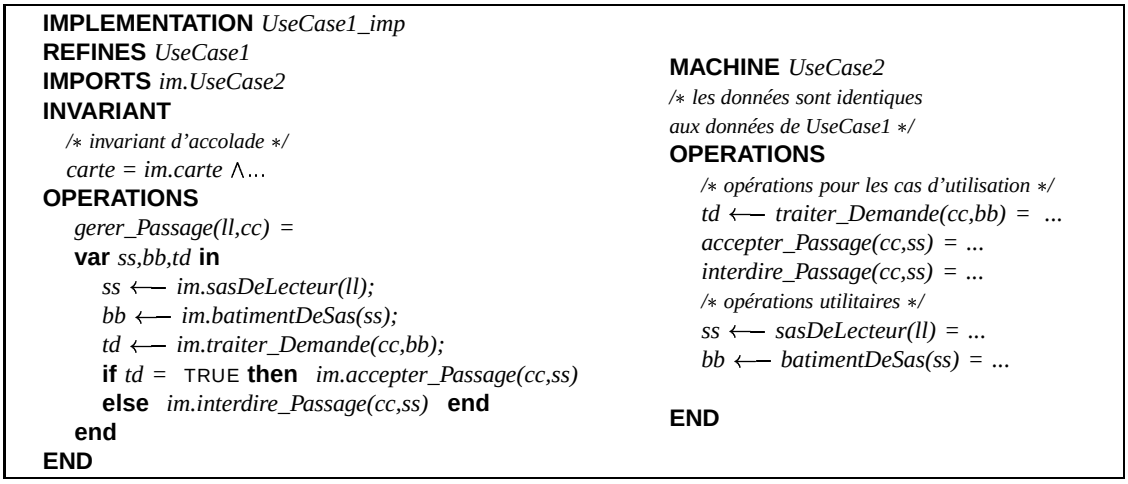

FIG. 4 – ***Implémentation de l'opération gerer_Passage***

Passage du cas d'utilisation **gérer Passage**; l'opération *gerer_Passage* est implantée par *traiter_Demande*, *accepter_Passage* et *interdire_Passage* ainsi que par deux opérations utilitaires *sasDeLecteur* et *batimentDeSas*. On note également que les données dans les deux machines *UseCase1* et *UseCase2* sont identiques car elles sont dérivées des mêmes classes qui sont associées aux cas d'utilisation. Afi n de distinguer les deux ensembles de données dans la clause INVARIANT de l'implémentation *UseCase1_imp*, on renomme *UseCase2* dans la clause IMPORTS.

Dans la suite, on implante la machine *UseCase2* de la même manière que la machine *UseCase1*. Mais cette fois-ci chaque opération utilitaire dans *UseCase2* est implantée par une opération identique dans la machine importée (voir fi gure 5). On répète ce processus jusqu'à ce que la machine obtenue ne contienne que les opérations de base des classes.

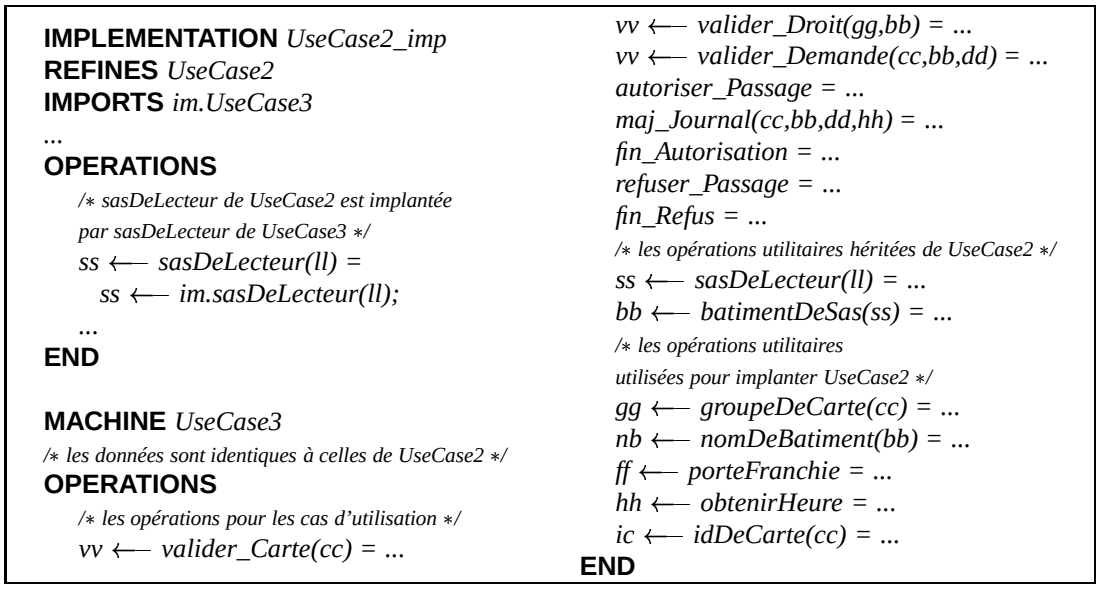

FIG. 5 – ***Implanter l'opération utilitaire sasDeLecteur***

Dans l'exemple, pour implanter la machine *UseCase2*, on crée la machine *UseCase3* pour les cas d'utilisation **valider Carte**, **valider Droit**, **valider Demande**, **autoriser Passage**, **maj Journal**, **fin**

Autorisation, **refuser Passage**, **fin Refus**. Des opérations utilitaires sont également identifiées comme le montre la figure 5. L'opération utilitaire *sasDeLecteur* dans *UseCase2* est implantée par une opération identique *sasDeLecteur* dans *UseCase3*; cela est permis parce que *UseCase3* est renommée dans *UseCase2_imp*.

La dernière machine modélise seulement les opérations de base des classes; elle est décomposée en machines de classes et d'associations non-fixées[2] qui sont créées par les schémas de dérivation objet-B proposés par [Mey01]. Dans l'exemple, la machine *UseCase3* peut être décomposée en machines des classes et des associations telles que *Carte*, *Sas*, *Enregistrement*... comme le montre la figure 6. Noter les liens "USES" entre machines dans cette figure; la machine *Enregistrement* "USES" la machine *Carte* à cause d'une association `*:1` entre les deux classes **Enregistrement** et **Carte**; il en est de même pour les liens entre *Carte* et *Groupe*, *Enregistrement* et *Batiment*, *Sas* et *Batiment*. La machine *GroupeBatiment* correspond à l'association non-fixée entre **Groupe** et **Batiment**; par définition *GroupeBatiment* "USES" *Groupe* et *Batiment*. De plus, l'agrégation entre **Sas** et **Lecteur** se traduit par un lien d'inclusion entre *Sas* et *Lecteur*; il en est de même pour le lien d'inclusion entre *Sas* et *Porte*.

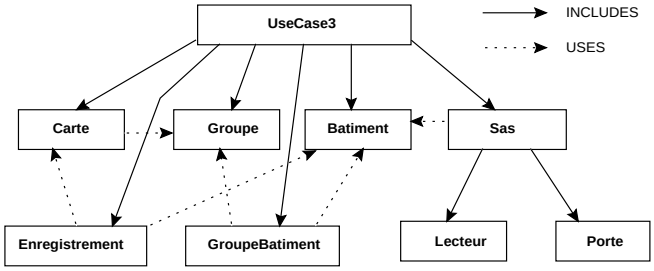

FIG. 6 – ***Décomposer UseCase3 en machines des classes et des associations***

3.2.2 Duplication d'opération

Dans le processus de transcription des cas d'utilisation en B il peut y avoir des situations où un cas d'utilisation donne lieu à plusieurs opérations B identiques. À titre d'exemple, considérons la situation suivante : soit **UC1** et **UC2** deux cas d'utilisation au niveau du système. Supposons que le comportement de **UC2** soit inclus dans le comportement de **UC1**[3]; dans cette situation, **UC2** est modélisé à la fois dans deux machines *UseCase1* (car **UC2** est au niveau du système) et *UseCase2* (car l'opération B de **UC2** est utilisée pour implanter l'opération de **UC1**). Le principe pour implanter une opération utilitaire marche bien pour les opérations de **UC2** comme montré dans la figure 7 (afin de simplifier la figure, nous supposons que *UC1* et *UC2* n'ont pas d'argument).

3.2.3 Modularité

On peut créer une machine abstraite *Types* pour modéliser tous les types d'attributs communs à différentes classes. Dans l'exemple, c'est le cas pour les types : **DIRECTION**, qui est commun pour les deux classes **Enregistrement** et **Sas**; **ID_CARTE**, qui est commun pour les deux classes **Carte** et **Enregsitrement**; et **NOM_BATIMENT** qui est commun pour les deux classes **Batiment** et **Enregistrement**. La machine *Types* dans ce cas sera seulement vue par les machines (et leur implémentation s'il y en a) qui modélisent les variables de types communs. La figure 8 représente l'architecture de la spécification B finale pour le modèle des cas d'utilisation du système de contrôle d'accès. Noter que la machine

2. L'association dont l'instance est créée et supprimée indépendamment des instances des classes reliées.

3. Dans l'exemple, si on veut que le système fonctionne pour faire face à un incendie, alors **gérer Passage** joue le rôle de **UC1** et **gérer Incendie** joue le rôle de **UC2**.

```
MACHINE UseCase1
   ...
OPERATIONS
   UC1 = ...
   UC2 = ...
   ...
END

IMPLEMENTATION UseCase1_imp
REFINES UseCase1
IMPORTS im.UseCase2
   ...
OPERATIONS
   /* on appelle im.UC2 dans UC1*/
   UC1 = ...
   /* UC2 appelle "identiquement" im.UC2 */
   UC2 = im.UC2;
   ...

END

MACHINE UseCase2
   ...
OPERATIONS
   /* UC2 dans UseCase2 est identique
   à UC2 dans UseCase1 */
   UC2 = ...
   ...
END
```

FIG. 7 – ***Un cas d'utilisation peut avoir plusieurs opérations B identiques***

UseCase3 n'a pas de lien "SEES" avec *Types* car toutes ses données sont déclarées dans ses machines incluses. Dans *UseCase1* et *UseCase2* on déclare explicitement les variables de toutes les classes, elles ont donc des liens "SEES" avec *Types*.

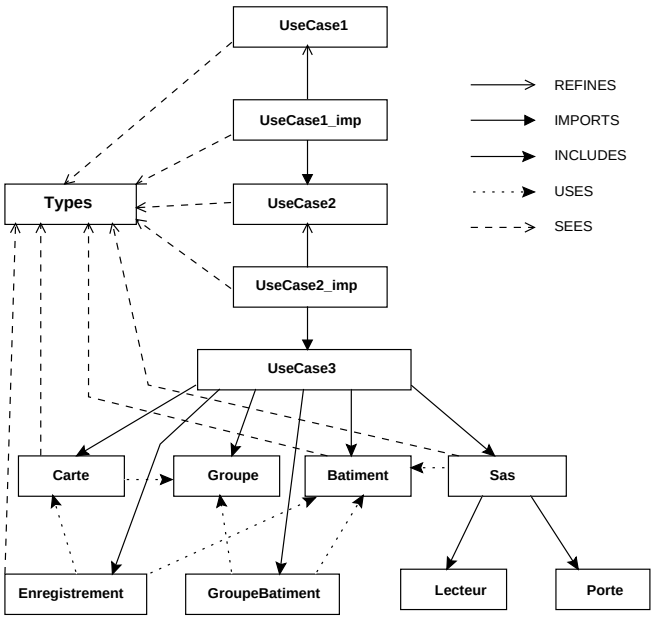

FIG. 8 – ***Architecture de la spécification B du modèle des cas d'utilisation du système de contrôle d'accès***

4 Conclusion

4.1 Notre contribution à propos de la formalisation des cas d'utilisation

L'idée de formalisation des cas d'utilisation n'est pas neuve. Dans [Hur97], Hurlbut a présenté une inspection auprès de trente et une propositions pour modéliser des cas d'utilisation; dans [PBP99], Petre et al. ont proposé une approche pour modéliser les cas d'utilisation par calcul de raffi nement; ce travail

donne la perspective de vérifer formellement des propriétés des cas d'utilisation tel que "achievability". Le manque d'outils support puissants est l'inconvénient majeur des approches existantes. Notre approche est différente des approches existantes par le fait qu'elle a remédié à cet inconvénient en permettant d'utiliser les outils de preuve de B (AtelierB [STE98], B-Toolkit [B-C96]) pour analyser formellement le modèle des besoins orienté objets qui se compose du modèle des cas d'utilisation et du modèle des classes du domaine d'application du même système. Cela permet de vérifer la conformité entre le modèle des cas d'utilisation et le modèle des classes du domaine d'application.

Notre approche respecte bien les principes proposés par Jacobson à propos de la formalisation des cas d'utilisation [Jac95], à savoir :

1. la communication entre les événements qui se produisent à l'intérieur du système n'est pas concernée;
2. les conflits entre les cas d'utilisation sont ignorés;
3. la concurrence entre les cas d'utilisation est ignorée.

Notre approche s'inspire de l'idée de structuration des cas d'utilisation de Cockburn [Coc97, Coc00]. Les cas d'utilisation dont on a parlé dans notre article correspondent aux cas d'utilisation de type "user-goal" et "sub-function" dans la proposition de Cockburn. Il est à noter que la structuration des cas d'utilisation ne décrit pas la conception du système.

Dans notre approche, avant de transcrire les cas d'utilisation en B il est conseillé de les décrire sous forme textuelle; cela vise à deux buts : d'une part, les analystes et les utilisateurs potentiels peuvent coopérer aisément pour définir et détailler les fonctionnaliés attendues du système; d'autre part, la description textuelle des cas d'utilisation sert à compléter les corps des opérations B pour les cas d'utilisation. Notre approche fonctionne bien pour le stéréotype «`uses`». En ce qui concerne le stéréotype «`extends`» si la condition d'extension est maîtrisée par le système ça marche, sinon il est difficile de modéliser la dépendance entre le cas d'utilisation "père" et le cas d'utilisation "fils" (voir section 2.1).

4.2 Notre contribution à propos des chémas de dérivation objet-B

De nombreux travaux autour de l'intégration Objet-B ont été réalisés [Lan96, Ngu98, Mey01] mais leurs résultats se concentrent principalement sur les aspects statiques de la spécification orientée objets. Pour les aspects dynamiques tels que ceux décrits dans les diagrammes des cas d'utilisation et les diagrammes de collaborations, il n'y a aucune proposition appropriée. Dans notre article, nous apportons des guides pour dériver les machines B pour les cas d'utilisation. Nos machines sont au dessus des machines B dérivées des classes et des associations. L'architecture de la spécification B obtenue (voir un exemple dans la figure 8) reflète la structuration du modèle des cas d'utilisation mais pas l'architecture du système.

4.3 Travaux à envisager

Dans [LS01] nous proposons une approche pour formaliser en B les diagrammes de collaborations qui décrivent la réalisation des cas d'utilisation en termes d'objets. Le modèle de collaboration peut être considéré comme un raffinement du modèle des cas d'utilisation. Cette idée donne lieu à une étude à propos de la liaison de raffinement entre la spécification B dérivée de diagrammes de collaboration et la spécification B dérivée de modèles des cas d'utilisation.

Nos travaux et les travaux de [Mey01] donnent lieu à une approche complète pour dériver une spécification B à partir d'une spécification orientée objet qui contient à la fois les aspects statiques et dynamiques d'un système. Il en résulte que le développement de la spécification B devient "moins difficile" à l'aide de la spécification orientée objets. De plus, il est tout à fait possible d'analyser formellement une

spécifi cation orientée objet en analysant la spécifcation B correspondante. C'est aussi un sujet d'étude à aborder prochainement.

Remerciements: l'auteur adresse avec beaucoup de plaisir ses remerciements à Madame Jeanine Souquières pour sa direction tout au long du développement du papier. Merci également aux rapporteurs pour leurs remarques utiles et constructives.

Références

[Abr96] J.R. Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge University Press, 1996. ISBN 0-521-49619-5.

[B-C96] B-Core(UK) Ltd, Oxford (UK). *B-Toolkit User's Manual*, 1996. Release 3.2.

[Coc97] A. Cockburn. Structuring Use Cases with Goals. http://members.aol.com/acockburn/papers/usecases.htm, 1997.

[Coc00] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000. ISBN 0201702258.

[Coo99] Cooperative Requirements Engineering With Scenarios. Reports series 1996, 1997, 1998, 1999. http://SunSITE.Informatik.RWTH-Aachen.DE/CREWS, 1999.

[Gli00] M. Glinz. A Lightweight Approach to Consistency of Scenarios and Class Models. In *Proceedings of the Fourth International Conference on Requirements Engineering*, Illinois, June 10-23, 2000. Available at http://www.ifi.unizh.ch:80/groups/req/staff/glinz/activities.html.

[Hur97] R. Hurlbut. A Survey of Approaches For Describing and Formalizing Use Cases. http://www.iit.edu/∼rhurlbut/xpt-tr-97-03.html, 1997.

[Jac95] I. Jacobson. Modeling with Use Cases : Formalizing Use-Case modeling. *Journal of Object-Oriented Programming*, 8(3), June 1995.

[JCJO93] I. Jacobson, M. Christerson, P. Jonson, and G. Overgaard. *Le génie logiciel orienté objet*. Addison Wesley France, 1993.

[Lan96] K. Lano. *The B Language and Method : A Guide to Practical Formal Development*. FACIT. Springer-Verlag, 1996. ISBN 3-540-76033-4.

[LPJ00] Y. Ledru, G. Padiou, and J. Jaray. Étude de cas: Système de contrôle d'accès. http://www-lsr.imag.fr/afadl2000/EtudeDeCas/, 2000.

[LS01] H. Ledang and J. Souquières. Modeling class operations in B : a case study on the pump component. Technical Report A01-R-011, Laboratoire Lorrain de Recherche en Informatique et ses Applications, March 2001. Available at http://www.loria.fr/∼ledang/publications/UML01.ps.Z.

[Mey01] E. Meyer. *Développements formels par objets: utilisation conjointe de B et d'UML*. PhD thesis, LORIA - Université Nancy 2, Nancy (F), mars 2001.

[Ngu98] H.P. Nguyen. *Dérivation de spécifications formelles B à partir de spécifications semi-formelles*. PhD thesis, Conservatoire National des Arts et Métiers - CEDRIC, Paris (F), décembre 1998.

[PBP99] L. Petre, R.J. Back, and I.P. Paltor. Formalising UML Use Cases in the Refinement Calculus. Technical Report 279, Turku Centre for Computer Science, 1999. Available at http://www.tucs.abo.fi/publications/techreports/TR279.html.

[RJB98] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998. ISBN 0-201-30998-X.

[Rum94] J. Rumbaugh. Using use cases to capture requirements. *Journal of object-oriented programming*, 7(5), September 1994.

[STE98] STERIA - Technologies de l'Information, Aix-en-Provence (F). *Atelier B, Manuel Utilisateur*, 1998. Version 3.5.