Conservatoire **N**ational des **A**rts et **M**étiers

THÈSE

Présentée pour l'obtention du tire de

DOCTEUR Du CNAM Spécialité **INFORMATIQUE**

par

Hong Phuong NGUYEN

DÉRIVATION DE SPÉCIFICATIONS FORMELLES B À PARTIR DE SPÉCIFICATIONS SEMI-FORMELLES

Soutenue le 3 Décembre 1998 devant le jury composé de

Présidente	Mme Véronique DONZEAU-GOUGE (CNAM Paris)
Directeurs	M. Philippe FACON (IIE-CNAM Paris)
	Mme Régine LALEAU (IIE-CNAM Paris)
Rapporteurs	M. Henri HABRIAS (IRIN, Université de Nantes)
	M. Yves LEDRU (LSR-IMAG, Université Grenoble I)
Examinateurs	M. Jacky AKOKA (CNAM Paris)
	M. Xavier CASTELLANI (IIE-CNAM Paris)
	M. Clément ROQUES (STERIA)

Centre d'**É**tudes et **D**e **R**echerche en **I**nformatique du **C**NAM (CEDRIC)

Remerciements

Je souhaite exprimer ma gratitude à Madame Véronique Donzeau-Gouge, Professeur au CNAM, pour m'avoir fait l'honneur de présider mon jury de thèse.

Monsieur Henri Habrias, Professeur à l'Université de Nantes, et Monsieur Yves Ledru, Professeur à l'Université Joseph Fourier, m'ont fait l'immense plaisir d'accepter d'être rapporteurs de cette thèse. Leurs conseils, leurs remarques ont véritablement amélioré la qualité de ce mémoire. Je tiens à les remercier très sincèrement.

Je remercie très vivement Monsieur Xavier Castellani, Professeur à l'IIE (CNAM Paris) pour avoir accepté de prendre part à mon jury de soutenance et pour ses précieuses observations.

J'adresse également mes sincères remerciements à Monsieur Jacky Akoka, Professeur au CNAM et Monsieur Clément Roques pour l'honneur qu'ils m'ont fait en participant au jury.

Ce travail de recherche m'a été proposé par Monsieur Philippe Facon, Professeur à l'IIE (CNAM Paris) et responsable de l'équipe de recherche "Spécification et Analyse du Logiciel" du CEDRIC. Je tiens à lui exprimer ma profonde gratitude pour m'avoir accueillie au sein de son équipe, pour avoir accompagné mes premiers pas dans la recherche, pour avoir dirigé ce travail, pour sa confiance et pour l'aide qu'il m'a apportée tout au long de ces années.

Madame Régine Laleau, Maître de Conférences à l'IIE m'a fait bénéficier, tout au long de ce travail, de ses précieux conseils et de discussions enrichissantes. Elle a contribué efficacement, par sa disponibilité, ses relectures et ses remarques pertinentes, à l'aboutissement de ce travail. Je la remercie profondément. Qu'elle trouve ici l'expression de ma reconnaissance.

Un grand MERCI

à Éric Soutif pour son amitié, la précieuse aide qu'il m'a apportée pendant la rédaction de ce manuscrit, et pour son inlassable soutien surtout durant les derniers mois.

à Houda Benhadid, Bertrand Decocq, Rouquia Djabali, Ilham Nadira Lammari pour leur gentillesse et les bons moments passés ensemble pendant ces trois années.

à Luong Tiet pour m'avoir fait profiter de ses compétences sur l'Atelier B, pour son soutien, ses encouragements dans les moments difficiles.

à Mireille Jouve pour son soutien moral lors de nos trajets en train.

à Jean-luc Kors, Hubert Delafosse, Christophe Billot pour l'aide technique qu'ils m'ont apportée.

au personnel enseignant et administratif de l'IIE qui m'a accueillie pour préparer cette thèse.

à Georges Psareff et Auguste Konate du service reprographie, qui m'ont aidée à la mise en forme de ce document.

à mes amis qui, de près ou de loin, ont contribué à l'accomplissement de ce travail.

Je remercie Hung qui, malgré l'éloignement, a été pour moi d'un énorme soutien. Je le remercie aussi pour sa confiance.

Enfin, mes plus grands remerciements s'adressent à ma famille sans qui je ne serais pas où j'en suis aujourd'hui. Qu'ils sachent à travers ces quelques mots combien je leur suis reconnaissante et combien je mesure tout ce que je leur dois.

TABLE DES MATIÈRES

Chapitre 1 - Introduction **1**

I. ..... Problématique
II. ..... Solutions existantes
III. ..... Objectif du travail de recherche
IV. ..... Résumé de la contribution
V. ..... Organisation de la thèse

Chapitre 2 - État de l'Art **9**

I. ..... Méthodes semi-formelles
II. ..... Méthodes formelles
II.1. ..... *Présentation de Z*
II.1.1. ..... La notion d'ensemble
II.1.2. ..... La notion de schéma
II.1.3. ..... Object-Z
II.2. ..... *Présentation de B*
III. ..... Les différentes démarches pour l'intégration des méthodes semi-formelles et formelles 17
III.1. ..... *Visualisation graphique des langages de spécification formelle existants*
III.2. ..... *Proposition d'une nouvelle méthode*
III.3. ..... *Dérivation, à partir d'une spécification semi-formelle, d'une spécification formelle*
III.4. ..... *Choix d'une démarche*
IV. ..... Travaux existants dans la dérivation d'une spécification formelle à partir d'une spécification semi-formelle 21
IV. 1. ..... *Travaux concernant la dérivation d'une spécification formelle Z et Object-Z à partir d'une spécification semi-formelle*
IV.1.1. ..... De OMT vers Object-Z dans [DUP 97] et [DUP 98]
IV.1.2. ..... De Fusion vers Z dans [FRA 97]
IV.1.3. ..... De UML vers Z dans [SHR 97]
IV.2. ..... *Travaux concernant la dérivation d'une spécification formelle B à partir d'une spécification semi-formelle* *37*
IV.2.1. ..... Du modèle E/R étendu vers B dans [NAG 94]
IV.2.2. ..... De OMT vers B dans [LAN 94b]
IV.2.3. ..... De NIAM vers B dans [HAD 96]
IV.2.4. ..... De OMT vers B dans [MAL 98]

IV.2.5. Des diagrammes d'états de Harel vers B dans [SEK 98]
V. Synthèse générale
V.1. *Comparaison selon des critères quantitatifs*
V.1.1. L'aspect statique
V.1.2. L'aspect dynamique
V.1.3. La modularisation
V.2. *Comparaison selon des critères qualitatifs*
V.2.1. L'aspect statique
V.2.2. L'aspect dynamique
V.2.3. La modularisation
V.2.4. Les preuves
V.3. *Conclusion*

Chapitre 3 - Étude des Méthodes Semi-Formelles 55

I. Introduction
II. Le modèle d'objets
II.1. *Objet et concepts liés*
II.1.1. Objet
II.1.2. Entité
II.1.3. Attribut
II.1.4. Identité d'objet, Clé
II.2. *Association et concepts liés*
II.2.1. Lien, Association
II.2.2. Rôle
II.2.3. Association fixe
II.2.4. Qualification
II.2.5. Entité associative
II.2.6. Agrégation
II.2.7. Contraintes d'intégrité

II
II.3. *Héritage et concepts liés*
II.3.1. Lien d'héritage
II.3.2. Contrainte de couverture entre sous-entités
II.3.3. Contrainte de disjonction entre sous-entités
II.4. *Opération*
II.4.1. Opération sur une entité
II.4.2. Opération sur une association
III. Le modèle Dynamique
III.1. *Concepts de base*
III.1.1. Événement, État, Transition, Diagramme états/transitions, Condition
III.1.2. Opération
III.1.3. Liaison entre le modèle d'objets et le modèle dynamique
III.2. *Concepts évolués*
III.2.1. Transition automatique
III.2.2. Concurrence d'objets
III.2.3. Échange d'événements
III.2.4. Diagramme associé à l'administrateur

III.2.5. Scénario et Diagramme de suivi d'événements
IV. Conclusion

Chapitre 4 - Présentation du langage de spécification de la méthode B 97

I. Présentation générale
I.1. La méthode B
I.2. Le langage B
I.3. La notion de "machine abstraite"
I.4. Les outils
II. Syntaxe des données d'une "machine abstraite"
II.1. La clause "MACHINE"
II.2. La clause "SETS"
II.3. La clause "DEFINITIONS"
II.4. La clause "VARIABLES"
II.5. La clause "INVARIANT"
II.6. La clause "INITIALISATION"
II.7. Expressions de la théorie des ensembles de B

III
III. Syntaxe des opérations d'une "machine abstraite"
III.1. En-tête d'une opération
III.2. Corps d'une opération
III.3. Langage des substitutions généralisées
III.3.1. Substitution "Skip"
III.3.2. Substitution "Devient égal"
III.3.3. Substitution "Devient tel que"
III.3.4. Substitution "Devient un élément de"
III.3.5. Substitution "Pré-condition"
III.3.6. Substitution "If"
III.3.7. Substitution "Any"
III.3.8. Substitution "Appel d'opération"
III.3.9. Substitution "Simultanée"
IV. Modularité dans les spécifications
IV.1. La clause "INCLUDES"
IV.1.1. Visibilité
IV.1.2. Transitivité
IV.1.3. Renommage
IV.2. La clause "USES"
IV.2.1. Visibilité
IV.2.2. Transitivité
IV.3. La clause "PROMOTES"
V. Raffinement
V.1. Raffinement des données 112

V.2 ... *Raffinement des opérations*
VI. ... Preuves
VI.1 ... *Substitution "Skip"*
VI.2. ... *Substitution "Devient égal"*
VI.3. ... *Substitution "Devient tel que"*
VI.4. ... *Substitution "Devient un élément de"*
VI.5. ... *Substitution "Pre-condition"*
VI.6. ... *Substitution "If"*
VI.7. ... *Substitution "Any"*
VI.8. ... *Substitution "Appel d'opération"*
VI.9. ... *Substitution "Simultanée"*
VII. ... Exemple
VII.1 ... *La machine abstraite*
VII.2 ... *Obligations de preuve liées à la machine*
VIII ... Conclusion

IV

Chapitre 5 - La dérivation, à partir d'une spécification semi-formelle, d'une spécification formelle B ... 121

I. ... Introduction
II. ... Traduction du modèle d'objets
II.1 ... *Formalisation des concepts du modèle d'objets*
II.1.1. ... Objet et concepts liés
II.1.2. ... Association et concepts liés
II.1.3. ... Héritage et concepts liés
II.2 ... *Modularisation de la spécification*
II.2.1. ... Entité
II.2.2. ... Association
II.2.3. ... Héritage
II.3 ... *Génération des opérations de base*
II.3.1. ... Principes généraux
II.3.2. ... Opérations sur les entités
II.3.3. ... Opérations sur les associations
II.3.4. ... Opérations sur les attributs des entités et des associations
III. ... Traduction du modèle dynamique
III.1. Utilisation des notations B dans les diagrammes états/transitions... 164
III.2. Formalisation d'un état ... 165
III.3. ... *Génération de squelettes d'opérations B et d'obligations de preuve à partir des diagrammes E/T et scénarios*
III.3.1 ... Scénario mono-entité correspondant à une seule transition avec une seule action
III.3.2 ... Scénario mono-entité correspondant à une seule transition avec plusieurs actions
III.3.3 ... Scénario mono-entité correspondant à plusieurs transitions partant d'un même état avec

conditions de déclenchement différentes
III.3.4. Scénario mono-entité correspondant à plusieurs transitions partant d'états différents
III.3.5. Scénario multi-entités
III.3.6. Scénario dont l'événement d'entrée arrive à l'administrateur d'une entité
III.4. Répartition des opérations dans les machines
III.5. Autre technique possible : utilisation du raffinement

V
IV. Réalisation d'un prototype
IV.1. Les fonctionnalités du prototype
IV.1.1. Module de saisie des diagrammes
IV.1.2. Module de traduction du modèle d'objets en B
IV.1.3. Module de traduction du modèle dynamique en B.
IV.2. La réalisation du prototype
IV.2.1. Le module de saisie des diagrammes
IV.2.2. Le module de traduction des diagrammes
V. Conclusion

Chapitre 6 - Conclusion 197

I. Bilan de la contribution
II. Positionnement de la contribution
II.1. Application des critères quantitatifs
II.1.1. L'aspect statique
II.1.2. L'aspect dynamique
II.1.3. La modularisation
II.2. Application des critères qualitatifs
III. Perspectives

Bibliographie 203

Index 211

Annexes 215

Chapitre 1

INTRODUCTION

I. Problématique

Les méthodes les plus utilisées en conception de systèmes d'information (OMT, UML, ...), dites semi-formelles, sont basées principalement sur l'utilisation de divers diagrammes, d'explications en langue naturelle et de guides méthodologiques. Ces méthodes présentent des avantages indéniables. Elles représentent le système d'une manière à la fois intuitive et synthétique. De ce fait, elles sont bien adaptées à la plupart des utilisateurs. Ces avantages ont contribué à répandre largement leur utilisation dans l'industrie. Mais leur principal défaut est l'absence d'une sémantique précise des diverses notations utilisées, ce qui entraîne souvent des ambiguïtés. En outre, il est impossible de prouver la cohérence du système, ce qui limite la fiabilité des logiciels produits, sauf à augmenter notablement la phase de test. Le caractère incomplet de ces méthodes semi-formelles est résumé par la phrase de [HAB 95] : "sans doute du fait que les *méthodes* sont devenues en informatique des produits de commerce, les références formelles ont été abandonnées. Si on ne dispose pas de sémantique bien définie, si on ne dispose pas de capacités télépathiques, il faut faire appel à l'expert.".

D'un autre côté, les méthodes formelles du génie logiciel fournissent des notations issues des mathématiques, en particulier de la logique, permettant de décrire très précisément les propriétés des systèmes à construire. De ce fait, ces méthodes disposent de techniques de preuve permettant de vérifier complètement le raffinement des spécifications en code exécutable. Ces avantages apportent des solutions aux problèmes posés par l'utilisation des méthodes semi-formelles. Malgré cela, elles ont été encore assez peu utilisées pour différentes raisons, en particulier une certaine faiblesse méthodologique (ce sont encore souvent plus des notations que des méthodes proprement dites) et un formalisme rebutant pour beaucoup de praticiens.

Est-ce à dire qu'il faut abandonner les méthodes semi-formelles au profit des méthodes formelles ? Non, car les deux approches sont complémentaires. Il est donc intéressant de les intégrer afin de combiner leurs avantages respectifs. Se pose alors la question du choix de la phase de développement qui est la plus appropriée à l'intégration. Dans toutes les méthodes semi-formelles classiques, on retrouve les phases suivantes : analyse des besoins, spécification, conception et codage. La phase de développement où l'intégration devrait être la plus bénéfique est celle de la spécification où la clarté, la précision, l'absence d'ambiguïté sont particulièrement importantes. Un certain nombre d'équipes de recherche ont ainsi commencé à se pencher sur le problème de l'intégration de ces deux types de méthodes : cf. le workshop organisé en Mars 1996 " Methods Integration Workshop " [MET 96].

Le problème qui se pose maintenant est le suivant : *comment intégrer les méthodes semi-formelles et formelles dans la phase de spécification ?*

II. Solutions existantes

Concernant l'intégration des méthodes semi-formelles et formelles, nous trouvons dans la littérature trois types de démarche.

La première de ces démarches consiste à visualiser graphiquement les langages de spécification formelle existants. Nous pouvons citer deux travaux [DIC 91] et [MAM 98] qui s'inscrivent dans cette perspective. Le premier présente un outil de traduction (réversible) sous forme de graphes d'une spécification décrite en VDM [JON 90]. Le second définit une méthode de représentation d'expressions relationnelles par des graphes et propose un algorithme qui calcule la formule relationnelle sans quantificateurs interprétant le graphe.

La deuxième démarche vise à proposer une nouvelle méthode définie sur des bases formelles solides. La méthode Fusion [COL 94] tente de former une nouvelle méthode cohérente regroupant et étendant les concepts des méthodes orientées-objet existantes. Les opérations sont décrites notamment en utilisant la structure "pré, post-condition" héritée de certaines méthodes formelles. Le projet DAIDA [SCH 91] peut également se situer dans cette démarche car il propose un langage de spécification pour chaque étape du cycle de vie.

La dernière démarche est l'obtention, à partir d'une spécification semi-formelle, d'une spécification formelle. Cette démarche est appelée dérivation. Elle peut être faite selon deux approches [FAC 95] : l'approche "interprétée" et l'approche "compilée". Le principe de base de l'approche "interprétée" est de conserver dans la spécification formelle des concepts du méta-modèle du système d'information. On écrit donc une fois pour toute une formalisation du méta-modèle. Puis on effectue la traduction de la partie propre à chaque application en utilisant la formalisation du méta-modèle. Le travail de [MON 97] consiste à construire un "interprète formel" des concepts du méta-modèle de la méthode OMT en B et décrit les fonctions formelles génériques pour les opérations de base. [MIS 97] propose un méta-modèle pour les concepts objets de [MEY 88] puis décrit ce méta-modèle dans les notations formelles Z [SPI 92]. Contrairement à l'approche "interprétée", l'approche "compilée" consiste à donner des règles permettant de traduire chaque concept du modèle semi-formel en un ensemble de notations formelles. On obtient ainsi directement pour chaque spécification semi-formelle, la spécification formelle correspondante. Beaucoup de travaux s'inscrivent dans cette démarche parmi lesquels nous pouvons citer [NAG 94], [LAN 94b] et [HAD 96] (le langage formel étant celui de B [ABR 96]), [FRA 97], [DUP 97, 98], [SHR 97] (le langage formel étant Z).

Signalons que cette classification des démarches envisageable pour l'intégration des méthodes semi-formelles et formelles avait déjà été adoptée dans [LED 96].

III. Objectif du travail de recherche

La démarche qui consiste à dériver une spécification formelle à partir d'une spécification semi-formelle a été retenue pour les raisons suivantes. Si la visualisation graphique des langages de spécification formelle convient aux spécifieurs qui apprécient une vue graphique, en revanche elle ne fournit pas plus d'information et présente l'inconvénient d'introduire un formalisme supplémentaire propre à chaque méthode formelle. L'approche consistant à proposer une nouvelle méthode pose des problèmes financiers et humains : elle nécessite une reconstruction des applications existantes et implique une formation spécifique aux utilisateurs dont les habitudes de travail peuvent changer radicalement, sans oublier le coût de développement des outils associés. La dérivation présente un double avantage : les acquis des méthodes semi-formelles déjà très répandus sont conservés et renforcés d'un point de vue formel sans que cela nécessite une reconstruction du système. Nous avons donc adopté cette dernière démarche et plus précisément l'approche "compilée". En effet, la dérivation suivant

l'approche "interprétée" demande une parfaite connaissance du méta-modèle, est plus lourde à utiliser et ne produit pas des spécifications modulaires.

Dans cette optique, il est naturel d'arrêter au préalable le choix d'une méthode semi-formelle de départ et d'une méthode formelle d'arrivée.

Devant la diversité des méthodes semi-formelles existantes, il n'est pas facile de faire un choix. De plus, chacune de ces méthodes apporte un vocabulaire différent pour des concepts souvent identiques et leur cadre d'utilisation n'est pas clairement établi. De ce fait, un premier travail a été d'étudier les diverses définitions des concepts dans les différentes méthodes semi-formelles (en particulier OMT [RUM 91] et UML [UML 97] qui sont les plus utilisées), d'en extraire les concepts les plus répandus, d'en préciser la sémantique et parfois d'ajouter de nouveaux concepts.

Concernant le choix d'un langage formel, la première étape a été de définir quel type de langage formel utiliser. Pour cela nous avons d'abord établi les principales caractéristiques des systèmes d'information que nous voulons modéliser, à savoir des systèmes d'information orientés principalement sur la gestion des données par opposition aux systèmes d'information temps réel ou de production. Ces caractéristiques sont les suivantes :

(i) gérer des ensembles d'objets existants plutôt que des ensembles d'objets possibles (types) ou des objets isolés,
(ii) définir des identités d'objet indépendamment de leur valeur et donc du concept de clé,
(iii) considérer le concept d'association à un haut niveau d'abstraction sans le réduire à un pointeur d'une entité vers une autre ou à un produit cartésien.

Deux autres facteurs sont également à considérer :

(iv) les méthodes formelles doivent offrir un mécanisme de modularité bien adapté aux systèmes d'information,
(v) elles doivent être des méthodes de développement complètes avec des bases mathématiques jusqu'à une axiomatisation, et offrir des outils facilitant leur utilisation.

Ces cinq critères nous ont permis d'éliminer un certain nombre de méthodes. La première catégorie concerne les méthodes orientées propriétés (ou algébriques) comme Larch [GUT 85] ou OBJ [GOG 83] car elles ne permettent pas de construire un modèle explicite du système. Dans la catégorie des méthodes orientées modèles, nous avons éliminé les méthodes formelles orientées objets comme VDM++, Object-Z, … [LAN 94a]. Tout d'abord, elles sont encore en cours de développement. Ensuite, et surtout, elles ne permettent pas de prendre en compte les points (i), (iii) et (v); en fait, elles sont mieux adaptées à la spécification de programmes écrits en langages à objets (type C++), or nous nous adressons plus à la spécification d'applications pour bases de données. Finalement, il reste les méthodes orientées modèles "classiques" (ensemblistes) telles que VDM [JON 90], Z [ABR 80] et B [ABR 96]. Nous avons éliminé Z car, comme le dit leur auteur commun [BUG 95] : "Z est plus une notation pour écrire des spécifications qu'une méthode de développement complète". Enfin, le choix de B est motivé par son utilisation de plus en plus répandue, son caractère récent et le fait qu'elle est particulièrement bien documentée et outillée par rapport aux autres méthodes.

Cette thèse consiste donc à proposer et décrire des règles de dérivation d'une spécification formelle en B à partir d'une spécification semi-formelle (type OMT, UML). Elle développe les propositions faites dans [FAC 96a], [FAC 96b], [FAC 96c] et [FAC 98].

IV. Résumé de la contribution

La première étape a consisté à étudier les concepts des méthodes semi-formelles de manière approfondie. Cette étude préliminaire a eu pour but de retenir les concepts les plus répandus des méthodes semi-formelles et d'examiner précisément leur sémantique. Cet examen nous a souvent amené à faire un choix entre plusieurs interprétations possibles des notations semi-formelles et même à les compléter lorsqu'elles ne sont pas suffisantes pour décrire des situations réelles. Concernant le modèle d'objets qui décrit la structure des données et leurs relations, nous nous sommes fixée comme objectif d'étudier un ensemble de concepts assez riches pour permettre une modélisation fidèle (objet, entité, association, héritage, ...). En ce qui concerne le modèle dynamique qui décrit les traitements ayant lieu sur les données, nous n'avons pas suivi la démarche appliquée au modèle d'objets à cause de la diversité des modèles dynamiques et de la grande imprécision sémantique des concepts eux-mêmes. Aussi avons nous choisi d'appuyer notre étude sur le modèle dynamique de OMT en raison de sa richesse

en concepts et de sa large diffusion. Nous avons présenté les concepts de base proposés par OMT dans les diagrammes états/transitions et les diagrammes de suivi d'événements. Nous avons éclairci certains points imprécis et fixé leur cadre d'utilisation. Les concepts évolués de OMT ont été également étudiés. Afin de spécifier les opérations d'entité, nous avons introduit la notion de diagramme associé à l'administrateur d'une entité. L'introduction des types de scénarios nous a permis de classifier les constructions possibles des diagrammes états/transitions ce qui est essentiel pour la dérivation des opérations.

Ces points étant clarifiés, l'étude de la dérivation proprement dite a été rendue possible. Notre objectif a été de proposer des règles de dérivation permettant d'obtenir une spécification formelle à la fois fidèle à la spécification semi-formelle de départ, modulaire pour simplifier les preuves et rendre possible la réutilisation des modules, et compréhensible.

La spécification semi-formelle initiale doit contenir un diagramme d'objets, un diagramme de suivi d'événements composés des scénarios, un diagramme états/transitions pour chaque entité du diagramme d'objets. Sa dérivation se fait en plusieurs étapes :

- *Traduction du diagramme d'objets en spécification B.* Les règles utilisées donnent, pour chaque concept ou contrainte du modèle, les notations B correspondantes. Puis la spécification obtenue est modularisée tout en restant fidèle au diagramme d'objets initial.

- *Génération des opérations de base à partir du diagramme d'objets*. Nous avons étudié la possibilité de générer automatiquement les opérations de base, associées aux entités et associations en fonction des contraintes de cardinalité exprimées dans le diagramme d'objets. Ces opérations permettent l'ajout, la suppression d'instances d'entité ou d'association et la modification de valeur d'attribut.

- *Utilisation des notations B dans les diagrammes états/transitions*. En effet, au niveau de la spécification semi-formelle, les diagrammes états/transitions sont annotés en langue naturelle. Si on veut obtenir une spécification B complète, il faut remplacer ces notations par des notations B. Cette étape est faite par le concepteur à l'aide des résultats des deux premières étapes.

- *Génération de squelettes d'opération B et d'obligations de preuve à partir des scénarios et des diagrammes états/transitions* . Chaque scénario donne lieu à une opération B qui décrit l'effet du scénario sur les données du système. Chaque type de scénario nous amène à une constitution particulière du corps de l'opération associée ainsi qu'à une obligation de preuve.

- *Répartition des opérations générées dans les modules* obtenus à la première étape.

- *Achèvement des spécifications obtenues.* Les modules doivent être complétés par le corps des opérations et les invariants correspondant aux contraintes d'intégrité non exprimables graphiquement. Cette étape est faite par le concepteur.

Afin de prouver la validité de la méthode de dérivation, des études de cas ont été réalisées en appliquant nos règles de dérivation et ont été validées par l'Atelier B [DIG 96]. Elles sont présentées dans les annexes.

V. Organisation de la thèse

Cette thèse est organisée en 6 chapitres.

Le chapitre 2 dresse un état de l'art dans le domaine de l'intégration des méthodes semi-formelles et formelles. Les deux premières parties de ce chapitre présentent rapidement les méthodes semi-formelles et formelles. La troisième partie est consacrée à une discussion sur les trois démarches possibles pour cette intégration. Cette discussion est suivie d'une présentation détaillée et de commentaires sur les travaux existants dans la démarche choisie pour cette recherche. Ce chapitre se termine par une synthèse générale mettant en évidence les acquis et les limites des travaux étudiés et ce qui motive par là-même ce travail de recherche.

Le chapitre 3 est consacré à une étude approfondie des concepts des méthodes semi-formelles dans le but d'en retenir les concepts les plus répandus et de définir précisément leur sémantique. Nous examinons, dans la première partie, les concepts associés au modèle d'objets décrivant la structure des données et leur relation dans le système. La deuxième partie présente l'étude des concepts relatifs au modèle dynamique qui décrit les traitements sur les données.

L'ensemble des concepts présentés dans ce chapitre constitue les éléments de base du travail de dérivation présenté dans le chapitre 5.

Le chapitre 4 présente les notions et les notations de base de la méthode B nécessaires à la compréhension de cette thèse.

Le chapitre 5 constitue le noyau de cette thèse, à savoir la dérivation, à partir d'une spécification semi-formelle, d'une spécification formelle B. Ce chapitre se compose de trois parties. La première concerne la traduction du modèle d'objets en B et la deuxième celle du modèle dynamique. Nous donnons des règles de traduction pour chacun des concepts présentés dans le chapitre 3, aussi bien pour la formalisation proprement dite que pour la structuration de la spécification formelle. Dans la troisième partie, nous décrivons l'architecture d'un outil implémentant les règles proposées au chapitre précédent, outil en cours de développement.

Enfin, nous terminons par une synthèse de notre contribution à l'intégration des méthodes semi-formelles et formelles et esquissons les suites possibles à ce travail.

Chapitre 2

ÉTAT DE L'ART

Le but de ce chapitre est de présenter un état de l'art des travaux publiés concernant l'intégration des méthodes semi-formelles et formelles. Les parties I et II sont consacrées à un rappel rapide sur les méthodes semi-formelles et formelles. Dans la partie III, nous discutons les différentes démarches possibles pour cette intégration. Cette discussion est suivie d'une présentation et de commentaires sur les travaux existants dans la démarche que nous avons choisie (partie IV). Nous terminons en montrant les acquis et les limites de ces travaux.

I. Méthodes semi-formelles

Les méthodes semi-formelles ont été conçues pour modéliser les systèmes d'information dans le but d'en obtenir une description moins ambiguë et plus synthétique (par l'utilisation de schémas) qu'en langue naturelle. Nous pouvons citer les méthodes suivantes : OMT [RUM 91], E/R [CHE 76], NIAM [HAB 93], UML [UML 97]. Deux modèles au minimum sont toujours proposés par ces méthodes. L'un décrit l'aspect statique, c'est-à-dire la structure des données et leurs relations. L'autre décrit les traitements qui ont lieu sur les données : il s'agit de l'aspect dynamique. Certaines méthodes comportent d'autres modèles. Par exemple, on trouve dans la méthode OMT le modèle fonctionnel qui décrit la transformation des valeurs des données dans le système.

Cette partie ne vise pas à présenter les méthodes semi-formelles dans leur totalité. Elle décrit simplement les concepts principaux qui se retrouvent dans les articles étudiés dans l'état de l'art pour en faciliter l'abord. Une description complète des modèles statique et dynamique de OMT sera donnée au chapitre 3.

Concernant l'aspect statique, il est important de noter que, malgré une terminologie parfois propre à chaque méthode, la sémantique des concepts des différentes méthodes reste très proche. Par ailleurs, l'aspect statique est traité dans tous les articles que nous étudierons. C'est pourquoi nous présentons maintenant, une fois pour toutes, les concepts principaux présents dans ces articles. En revanche, les points particuliers, ainsi que la description graphique de chaque concept, sont présentés lors de l'étude des différents articles.

- *Entité :*
  À l'origine, ce concept a été proposé par [CHE 76] dans le modèle E/A, « An entity is a thing which can be distinctly identified ». Voici sa traduction en francais donnée dans le Dictionnaire encyclopédique du Génie Logiciel [HAB 97] : « Une "entité" est une chose qui peut être distinctement identifié ». Une personne donnée, une compagnie sont des exemples d'entité. Un ensemble d'entités regroupe les entités ayant les mêmes structure et comportement. Par exemple, l'ensemble PERSONNE regroupe toutes les personnes. Des "attributs" décrivent la structure de l'entité. Chaque entité a sa propre identité définie par la valeur d'un sous-ensemble d'attributs appelé identifiant. Un attribut est défini par son nom et son type qui est obligatoirement un type de base.
  Dans d'autres méthodes proposées ultérieurement un ensemble d'entités porte le nom de classe ([SMI 77], OMT, UML, ...) ou d'entité (MERISE [TAR 83]) et une entité de [CHE 76] porte le nom d'objet.
  Exemple : Personne est un ensemble d'entités avec les attributs : Nom, Adresse, ...

- *Association :*
  Pour faire le lien entre des entités, CHEN propose également le concept d'association, « A relationship is an association among entities », ce qui est traduit par « Une "association" est un lien entre entités » [HAB 97]. Par exemple, l'association "père-fils" est définie entre deux entités personnes. CHEN a défini un ensemble d'associations comme étant une relation mathématique entre plusieurs entités. Des contraintes de cardinalité peuvent accompagner la description d'un ensemble

d'associations. Ces contraintes expriment le nombre minimum-maximum de liens pour chaque entité.
Dans la méthode NIAM, un ensemble d'associations est appelée relation. Dans OMT et UML une association désigne ensemble d'associations de CHEN.
Une association peut avoir des attributs.

- *Agrégation :*
  J. M. Smith et D. C. P. Smith [SMI 77] ont défini le concept d'agrégation. Initialement, l'agrégation consiste à regrouper différentes entités en une nouvelle entité de niveau supérieur.
  Exemple : Adresse est une agrégation de Code_Postal, Rue et Numéro.

  Une agrégation, d'après OMT et UML est une association particulière dans laquelle les objets d'une classe sont les composants d'objets d'une autre classe. Ce concept exprime la relation "composé-composant" ou "partie-de". Cette définition n'est en rien contradictoire avec la définition de [SMI 77]. Par exemple, Code_Postal est un composant d'Adresse (c'est-à-dire une partie d'Adresse) et une Adresse est composée entre autres d'un Code_Postal.

- *Généralisation/Spécialisation :*
  Le concept de généralisation/spécialisation a été également introduit par [SMI 77].
  Le concept de généralisation consiste à définir une entité (appelée super-entité) comme l'union de plusieurs autres entités (appelées sous-entités).
  Par exemple : Lecteur est une généralisation de Lecteur Adulte et Lecteur Enfant.
  À l'opposé, la spécialisation consiste à raffiner une entité en une ou plusieurs autres sous-entités.
  Par exemple : Employé est une spécialisation de l'entité Personne.

  Ce concept se retrouve également dans la littérature sous le nom de lien d'héritage. Par ce lien, une sous-entité hérite des propriétés (attributs et opérations) de sa super-entité. Une sous-entité peut posséder des propriétés supplémentaires qui lui sont propres. Tout objet d'une sous-entité est objet de sa super-entité.
  Une super-entité peut ne pas avoir d'instances propres, elle est dite abstraite. Elle sert à regrouper les propriétés communes de ses sous-entités.
  Une entité peut aussi hériter de plusieurs autres super-entités. Il s'agit de l'héritage multiple.

Pour ce qui concerne l'aspect dynamique, on observe une très grande variété des modèles mêmes et, à l'intérieur de chaque modèle, les concepts utilisés diffèrent d'un point de vue sémantique. Nous choisissons donc de décrire l'aspect dynamique dans la présentation de chaque article.

II. Méthodes formelles

Actuellement, il existe deux types de langage de spécification formelle :

- Les langages de spécification algébrique (orientés propriétés) :
  Ces langages sont basés sur la notion de type abstrait algébrique. La présentation d'un type abstrait algébrique est constituée d'une partie syntaxique (signature des opérations du type) et d'une partie sémantique (équations entre termes construits à partir de ces opérations).
  Exemple : LARCH [GUT 85], OBJ [GOG 83].

- Les langages de spécification ensembliste (orientés modèles) :
  Par opposition aux méthodes "orientées propriétés", ils utilisent des types abstraits de données prédéfinis (en particulier ensemblistes) pour modéliser l'état du système à construire, et on spécifie indépendamment chaque opération en décrivant son effet sur l'état du système.
  Exemple : VDM [JON 90], Z [SPI 92], Object-Z [DUK 91], B [ABR 96].

La question est de savoir laquelle des deux orientations est la mieux adaptée à la spécification des systèmes d'information. Les critères présentées dans le chapitre 1 nous ont fait préférer les approches orientées modèles.

Nous présentons rapidement ci-après les langages de spécification Z, Object-Z puis celui utilisé dans la méthode B : ils sont utilisés dans les articles que nous étudierons de manière approfondie dans la partie IV. Une description plus complète de Z peut être trouvée dans [SPI 92], de B au chapitre 4. L'objectif ici est seulement d'introduire les concepts de base nécessaires pour comprendre la partie IV de ce chapitre.

II.1. Présentation de Z

II.1.1. La notion d'ensemble

Les spécifications Z sont basées sur la notion d'ensemble. Un ensemble est une collection d'entités distinctes qui ont toutes le même type. Z ne propose pas de nombreux types prédéfinis. Il fournit seulement l'ensemble des entiers et des naturels. Pour disposer de plus de types, il faut introduire des ensembles de base. Introduire un ensemble de base assure juste qu'il existe un tel type. Par exemple, la déclaration suivante introduit les ensembles de base DATE et NOM : *[DATE, NOM]*.

Une autre manière de définir un ensemble est de décrire ses éléments. Par exemple :

Booléen ::= Vrai | Faux

On peut, à partir de ces ensembles, définir des sous-ensembles :

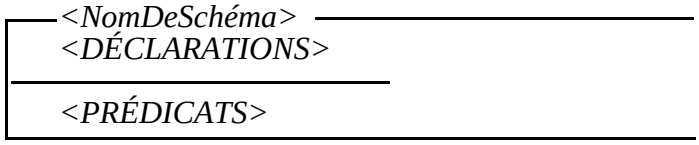
Si S est un ensemble.

$\mathbb{P}$ S est l'ensemble de tous les sous-ensembles de S.

$\mathbb{F}$ S est l'ensemble de tous les sous-ensembles finis de S.

II.1.2. La notion de schéma

Un schéma est une structure de spécification de données ou de traitements. Il est caractérisé par un nom. Un schéma est composé de deux parties : les déclarations et les prédicats. Les déclarations constituent le lexique des objets locaux aux schémas. Les prédicats précisent les contraintes portant sur les variables ou la relation entre les états initiaux et finaux d'une opération.



(i) Schéma d'état

Un schéma d'état consiste en la déclaration de variables et de prédicats qui contraignent ces variables. Les schémas peuvent inclure d'autres schémas. Cela signifie que toutes les variables et les prédicats du schéma inclus deviennent des parties du schéma incluant, c'est-à-dire que les déclarations sont fusionnées et que les prédicats sont joints.

(ii) Schéma d'opération

Un schéma d'opération représente la relation entre l'état avant l'opération et l'état après. Les opérations ont des variables d'entrée (suffixées par ?) et des variables de sortie (suffixées par !). Elles sont spécifiées par la relation entre ces deux types de variables et les deux états : état initial (avant l'opération) et l'état final (où les variables après l'opération sont primées).

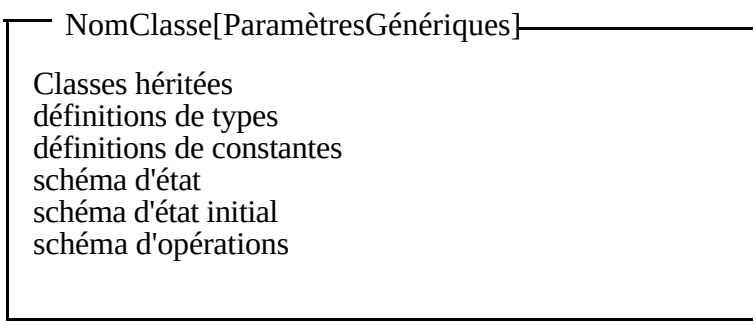
Les schémas et les définitions d'ensembles nécessitent d'écrire des expressions sur les nombres, les ensembles, les relations et les prédicats. Le langage Z est donc constitué de notations logiques et ensemblistes, structurées par le mécanisme des schémas.

NB : À l'intérieur d'un schéma, les parties entre crochets ([...]) correspondent à des commentaires.

II.1.3. Object-Z

Object-Z est une extension orientée-objet de Z. Il introduit la notion de classe qui encapsule un schéma d'état avec ses opérations. Des classes complexes peuvent être spécifiées à partir de classes plus simples en utilisant en particulier l'héritage.

Syntaxiquement, une classe est constituée de :



Un schéma de classe en Object-Z n'est pas seulement une extension syntaxique de la notion de schéma en Z. Au niveau de la sémantique, un schéma de classe définit un type dont les instances sont des références à des objets, c'est-à-dire des identités d'objet.

L'héritage permet de réutiliser des spécifications. Comme montré dans le schéma précédent, il est représenté par l'inclusion du nom de la sur-classe dans la sous-classe. ↓C représente l'ensemble des objets possibles de C et de ses sous-classes.

II.2. Présentation de B

La méthode B est une méthode de développement complète dont on trouvera une description détaillée dans le chapitre 4. Nous ne présenterons ici que quelques aspects nécessaires à partir d'un exemple inspiré de [ABR 96] :

```
MACHINE     MA_Personne
SETS        PERSONNES,
            SEXE = {M,F}
VARIABLES   Personnes, Sexe, Nom
INVARIANT   Personnes ⊆ PERSONNES ∧
            Sexe ∈ Personnes → SEXE ∧
            Nom ∈ Personnes → STRING
OPERATIONS
            Ajout_Personne(p,sexe, nom) =
            PRE     p ∈ PERSONNES − Personnes ∧
                    sexe ∈ SEXE ∧
                    nom ∈ STRING
            THEN    Personnes := Personnes ∪ {p} ||
                    Sexe(p) := sexe ||
                    Nom(p) := nom
            END ;
            Supp_Personne (p) =
            PRE     p ∈ Personnes
            THEN    Personnes := Personnes − {p} ||
                    Sexe := {p} ⩤ Sexe ||
                    Nom := {p}⩤ Nom
            END
END
```

En B, les spécifications sont organisées en "machines abstraites". Ces machines peuvent déclarer des ensembles abstraits sans autre précision (ici PERSONNES qui est l'ensemble de toutes les personnes possibles) ou décrits en extension (ici SEXE). On y trouve des variables d'état (ici Personnes pour l'ensemble des personnes existantes, Sexe et Nom pour les informations sur ces personnes) contraintes par un invariant : ici on a indiqué que l'ensemble Personnes est inclus dans PERSONNES, que Sexe est une fonction totale de Personnes dans SEXE et Nom une fonction totale de Personnes dans STRING. Le langage utilisé dans l'invariant, donc pour décrire l'état, est une version simplifiée du langage de la théorie classique des ensembles.

Les opérations procurent l'interface entre les variables d'état et l'extérieur de la machine abstraite. Les opérations sont décrites par le langage des substitutions généralisées, qui est une sorte de pseudo-code ensembliste, avec des constructions non exécutables du type "soit un ...tel que ... alors ..." (la sémantique de toute substitution est définie en termes de transformations de prédicats). Pour comprendre l'exemple, signalons simplement que $E \rightarrow F$ est l'ensemble des fonctions totales de E vers F, que l'opérateur || dénote l'"exécution" en parallèle de deux substitutions et que $S \lhd F$ vaut la restriction de F à : domaine(F) $-$ S.

Une machine A peut inclure une machine B en déclarant : *includes* B. Les opérations de B peuvent alors être appelées à partir de celles de A. Les variables de B ne sont, par contre, visibles qu'en "lecture" dans A. On ne peut appeler qu'une opération de la machine incluse dans une opération de la machine appelante. Ces contraintes permettent de vérifier indépendamment pour chaque machine que son invariant est bien respecté. Une machine A peut utiliser une machine B en déclarant : *uses* B. Comme pour le *includes*, les variables de B sont visibles (en lecture) dans A. En revanche, les opérations de B ne sont pas utilisables dans A.

Le but de ces mécanismes est de permettre le développement incrémental de spécifications, tout en assurant la séparation des preuves à effectuer pour chaque machine : les nouvelles machines incluant ou utilisant une machine M ne doivent pas ajouter aux obligations de preuve de cette dernière ; si on a prouvé en particulier que les opérations de M respectaient bien l'invariant de M, aucune nouvelle construction n'imposera de revérifier cet invariant.

III. Les différentes démarches pour l'intégration des méthodes semi-formelles et formelles

Les méthodes semi-formelles et formelles présentent leurs avantages, leurs inconvénients propres et, comme nous l'avons vu, les défauts des unes sont les qualités des autres. Aussi leur intégration est-elle primordiale pour une vision complète du système. Dans cette perspective, trois démarches rendent possible une telle intégration comme nous l'avons mentionné dans l'introduction :

- Visualiser graphiquement les langages de spécification formelle existants,
- Proposer une nouvelle méthode définie sur des bases formelles solides,
- Compléter et transformer les spécifications semi-formelles en spécifications formelles.

III.1. Visualisation graphique des langages de spécification formelle existants

Nous décrivons ci-dessous deux travaux de ce domaine :

- [DIC 91] présente un outil pour traduire sous forme de graphes une spécification décrite en VDM [JON 90], de manière réversible (VtP : VDM through Pictures). Plusieurs vues du système sont définies au moyen de graphes : vue pour les données, vue pour les opérations et vue pour les fonctions. Concernant les données, un graphe est défini pour décrire leur structure : TDS (Type Structure Diagram). Pour ce qui est des opérations, le graphe correspondant (OSD : Operation State Diagram) représente les états du système et comment les opérations modifient ces états. Un dernier graphe, IFD (Implicit Function Diagram), représente les fonctions implicites. Les règles de traduction des spécifications VDM vers les graphes proposés (et vice-versa) sont décrites.

- Un autre travail a été effectué au sein de notre équipe de recherche [MAM 98]. Ce travail a consisté, dans un premier temps, à définir une méthode de représentation d'expressions relationnelles par des graphes. Ensuite, partant d'un graphe étiqueté par des noms de relations, et de deux sommets privilégiés, un algorithme calcule la formule relationnelle sans quantificateurs interprétant le graphe. L'intérêt d'une telle interprétation est qu'elle est

très proche des langages de spécification formelle B et Z basés sur les langages ensemblistes et relationnels.

III.2. Proposition d'une nouvelle méthode

Cette approche consiste à définir une nouvelle méthode avec des bases formelles solides dans le but de concilier au sein d'une même méthode les avantages de l'aspect graphique des spécifications semi-formelles et la précision, la rigueur des spécifications formelles.

- La méthode Fusion [COL 94] est une tentative dans cette direction. Elle regroupe et étend les concepts des méthodes orientées-objet existantes dans l'objectif d'en retenir les meilleurs idées pour former une méthode nouvelle et cohérente. Les opérations, dans la phase d'analyse, sont décrites notamment en utilisant la structure "pré, post-condition" héritée de certaines méthodes formelles. Néanmoins, une partie importante de la description du système reste encore non formelle.

- Un autre travail intéressant que nous pouvons situer entre cette deuxième approche et la troisième est le projet DAIDA [SCH 91]. Il couvre complètement le cycle de vie d'une application. En fait, les auteurs ont proposé un langage pour chaque étape du cycle de vie. Dans la phase d'analyse des besoins, le langage TELOS est utilisé pour représenter les connaissances. La langage formel TDL est ensuite employé pour décrire le modèle de données et les transactions dans la phase de conception. Enfin, concernant l'étape d'implantation, le langage de programmation pour les bases de données est le langage DBPL. Malgré le côté formel du langage TDL, ce langage ne permet pas de faire de preuves. De ce fait, il a été proposé ensuite de traduire une spécification TDL en machine abstraite B. Cependant, en TDL, il n'y a pas d'association et, en outre, la traduction produit une seule machine alors que, parmi les critères abordés dans le chapitre 1, nous visons particulièrement la modularité de la spécification obtenue. En fait, ce travail concerne plus le raffinement.

III.3. Dérivation, à partir d'une spécification semi-formelle, d'une spécification formelle

Cette dérivation peut être faite suivant deux approches [FAC 95] : l'approche "interprétée" et l'approche "compilée".

- *Approche "interprétée" :*

  Le principe de base de cette approche est de conserver dans la spécification formelle les concepts du méta-modèle du système d'information. On écrit donc, une fois pour toutes, une formalisation du méta-modèle du modèle semi-formel. Puis on effectue la traduction de la partie propre à chaque application en utilisant la formalisation du méta-modèle.

  Dans ce contexte, nous pouvons citer le travail de [MON 97] qui développe les propositions de [FAC 95]. Ce travail consiste à construire un "interprète formel" des concepts du méta-modèle de la méthode OMT en B, et les fonctions formelles génériques pour les opérations de base (ajout, suppression, ...). Ensuite, l'auteur montre comment utiliser le méta-modèle pour chaque application concrète.

  [MIS 97] propose un méta-modèle pour les concepts objets de [MEY 88] puis décrit ce méta-modèle dans les notations formelles Z. Le but est de combiner les différents avantages des méthodes formelles et de la méta-modélisation.

  Le travail proposé dans [EVA 97] et [CLA 97] consiste à donner à certains concepts de la méthode UML une sémantique précise grâce à l'utilisation d'une notation formelle, en l'occurrence Z, dans le but d'éliminer les problèmes d'incohérence, d'incomplétude et d'ambiguïté de la sémantique d'UML.

  À la frontière de l'approche "interprétée", nous pouvons mentionner le travail de [FRE 96]. Ce travail a pour but de comparer des méthodes de conception orientées-objet. Il propose une méta-modélisation de ces méthodes comme un moyen de comparaison. Tous les schémas d'un méta-modèle sont constitués de trois parties : concepts, spécifications et remarques. La présentation de la partie spécification s'appuie sur le formalisme Z. Le but est d'introduire plus de rigueur en vue de faciliter les comparaisons et les transformations entre modèles. Cependant ce travail reste surtout à un niveau syntaxique.

- *Approche "compilée" :*

  Contrairement à l'approche "interprétée", l'approche "compilée" consiste à donner des règles permettant de traduire un modèle semi-formel directement dans un langage formel. Beaucoup de travaux ont été réalisés dans ce domaine. Dans la dérivation de spécifications formelles B, nous pouvons citer les travaux de [NAG 94], [LAN 94b], [HAD 96],

[MAL 98]. En spécification formelle Z, ceux de [FRA 97], [DUP 97], [DUP 98], [SHR 97]. En VDM : [LAL 95], [LAN 96], [LAR 94]. Un état de l'art plus complet des travaux concernant cette approche est détaillé dans la partie IV de ce chapitre.

III.4. Choix d'une démarche

Afin de motiver notre choix d'une démarche, nous présentons ici les avantages et les limites de chacune des trois démarches envisageables.

- La visualisation graphique des langages de spécification formelle convient aux spécifieurs qui préfèrent une vue graphique à une description textuelle de la même spécification. Cependant la vision graphique ne fournit pas plus d'information. De plus, cette démarche introduit un formalisme supplémentaire propre à chaque méthode formelle mais ne dispense pas du besoin d'avoir une bonne maîtrise de la méthode formelle correspondante.

- La deuxième approche offre, à l'intérieur d'une même méthode, les avantages réunis des deux méthodes de spécification semi-formelle et formelle. Mais l'introduction d'une nouvelle méthode pose des problèmes financiers et humains : elle nécessite une reconstruction des applications existantes, elle demande aux utilisateurs une formation spécifique dont les habitudes de travail peuvent changer radicalement.

- La dérivation de spécifications formelles part des spécifications semi-formelles des méthodes déjà existantes, répandues et, surtout, largement pratiquées dans le milieu industriel. Cette démarche présente l'intérêt de conserver les acquis des méthodes semi-formelles tout en les renforçant d'un point de vue formel. Elle ne nécessite pas de reconstruire les systèmes mais les complète utilement. Cependant, lors de la dérivation de spécifications formelles, un complément d'information est nécessaire car elle ne consiste pas en une simple traduction. Mais ce complément n'est-il pas en réalité un gain d'information ?

Nous pouvons conclure de ces éléments que la troisième démarche est la plus intéressante. Dans cette démarche, la dérivation utilisant l'approche "compilée" permet une utilisation directe du langage formel, tandis que celle utilisant l'approche "interprétée" a pour avantage de formaliser les notations du méta-modèle. L'approche "interprétée" oblige à définir plus explicitement la sémantique des notations semi-formelles alors que, dans l'approche "compilée", cette sémantique est cachée dans les règles de dérivation. Les spécifications

obtenues d'une application sont plus courtes, moins répétitives et ont une structure plus proche de celle du modèle semi-formel initial. Cependant, une telle dérivation est moins intuitive et ne produit pas des spécifications modulaires.

Dans l'objectif de proposer une aide à la spécification formelle à partir d'une spécification semi-formelle des systèmes d'information, nous avons choisi l'approche compilée. Aussi suivons nous cette démarche : il s'agit de définir des règles permettant de traduire une spécification semi-formelle en (squelette de) spécification formelle.

IV. Travaux existants dans la dérivation d'une spécification formelle à partir d'une spécification semi-formelle dans les systèmes d'information

Le travail présenté dans cette thèse est une contribution à la dérivation, à partir d'une spécification semi-formelle, d'une spécification formelle.

Afin de situer ce travail de recherche par rapport à ceux qui ont été déjà effectués, nous présentons, dans cette partie, un état de l'art concernant les travaux existants dans la dérivation d'une spécification formelle Z et B. Les articles choisis constituent un échantillon représentatif de l'ensemble des travaux publiés dans le domaine. Chaque article est présenté comme suit : une description suivi de nos commentaires (points forts, points faibles). Pour faciliter la comparaison des travaux, nous nous sommes efforcée de généraliser les règles de formalisation des modèles semi-formels proposées par les auteurs (règles qui sont souvent données soit sous forme d'exemples, soit de manière textuelle) afin d'obtenir une description homogène, selon la forme utilisée dans notre travail.

IV. 1. Travaux concernant la dérivation d'une spécification formelle Z et Object-Z à partir d'une spécification semi-formelle

Quatre articles nous apparaissent représentatifs de l'ensemble des travaux publiés dans le domaine :

- [DUP 97] et [DUP 98] dérivent des spécifications Object-Z à partir de diagrammes d'objets et diagrammes états/transitions de la méthode OMT.

- [FRA 97] utilise la méthode Fusion.
- [SHR 97] génère des spécifications Z à partir de classes UML.

IV.1.1. De OMT vers Object-Z dans [DUP 97] et [DUP 98]

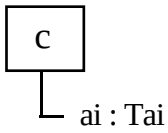
Ces deux articles présentent l'étude de la traduction du modèle d'objets et modèle dynamique de la méthode OMT en Object-Z.

IV.1.1.1. Description des règles de formalisation du modèle d'objets

Les concepts du modèle d'objets examinés par ce travail sont les concepts de classe, d'association, d'héritage et d'agrégation. Nous présentons ci-dessous les règles de traduction de ces concepts en Object-Z.

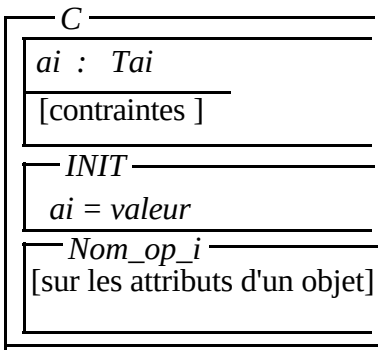
(i) Formalisation d'une classe

Chaque classe donne lieu à deux schémas Z : un schéma qui décrit la définition d'une classe en intention, c'est-à-dire la description des propriétés communes aux objets possibles, et un autre pour la description en extension (l'ensemble des objets existants)

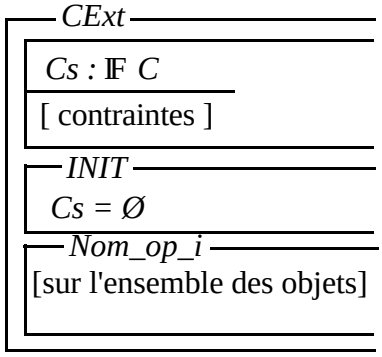


La formalisation en intention d'une classe :

[Tai] *si Tai* $\neq \mathbb{N}$

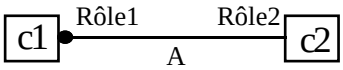


La formalisation en extension d'une classe :

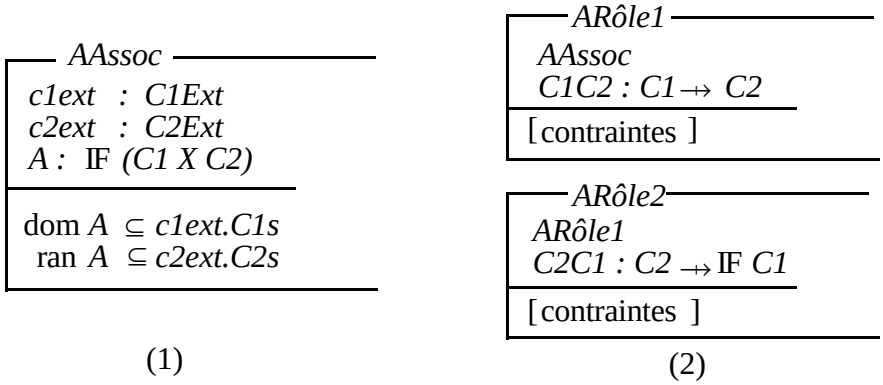


(ii) Formalisation d'une association

Les associations étudiées dans ce travail peuvent être binaires ou n-aires. Deux façons de formaliser une association ont été proposées :



- Formalisation d'une association comme une relation entre plusieurs classes



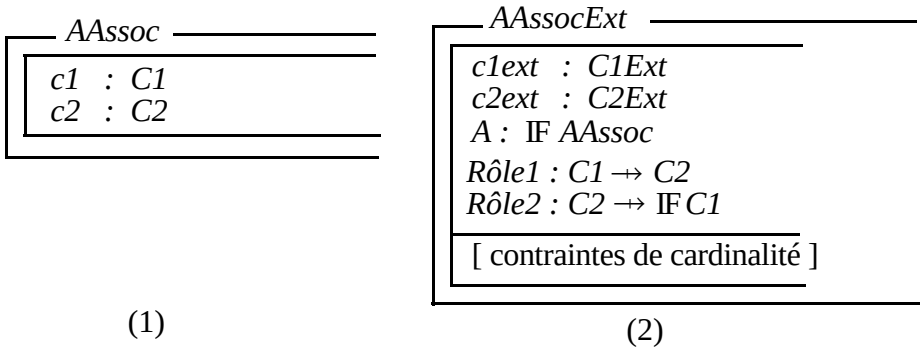
(1) Le schéma AAssoc déclare l'association A entre les classes c1, c2

(2) Les schémas ARôle1 et ARôle2 définissent les rôles et les contraintes de cardinalité (sous forme de fonction) entre les classes c1, c2 et l'association A.

- Formalisation d'une association comme un groupe de liens

La formalisation en intention d'une association :

La formalisation en extension d'une association :

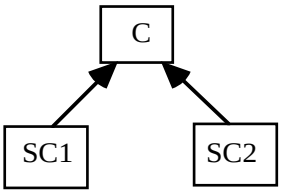


(1) Le schéma AAssoc déclare le type de chaque classe participant à l'association

(2) Le schéma AAssocExt contient :

- l'ensemble d'objets de l'association
- la présentation de chaque rôle sous forme de fonction
- les contraintes de cardinalité

(iii) Formalisation du lien d'héritage



La formalisation en intention d'une sous-classe :

SC1

C

[Déclaration des attributs et des contraintes, des opérations spécifiques à SC1]

(1)

La formalisation en extension de la super-classe :

CExt

sc1ext : SC1Ext
sc2ext : SC2Ext
Cs : ℙ↓ C

[Les contraintes possibles de type :
. identifiant
. inclusion des sous-classes dans la super-classe
. la super-classe est non-instanciable
. contrainte de disjonction entre les sous-classes]

(2)

Rappelons que ↓C désigne l'ensemble des objets possibles de C et de ses sous-classes.

(1) Le schéma SC1 contient la déclaration de la sous-classe. Le schéma SC1Ext définissant l'ensemble des objets de SC1 et le schéma C déclarant les attributs, les opérations de C sont spécifiées par la règle de formalisation d'une classe.

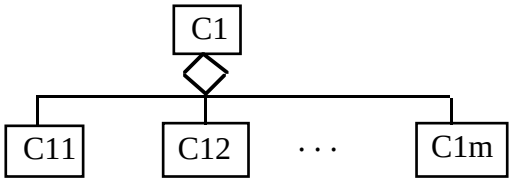
(2) Le schéma CExt exprime les contraintes entre les ensembles d'objets.

(iv) Formalisation du lien d'agrégation

Deux solutions sont proposées :

- La première solution consiste à voir l'agrégation comme une association ayant un sens particulier. Dans ce cas, une agrégation est spécifiée comme les autres associations.

- La deuxième solution représente l'agrégation par l'inclusion des objets composants dans l'objet composé, et si nécessaire la description de la dépendance existentielle des objets composants vers l'objet composé.



La décomposition d'une classe `c1` en classe `c11`, `c12`, ..., `c1m` est représentée par l'inclusion dans la classe composée `c1` des variables `c11`, `c12`, ..., `c1m` de type des classes composantes (`c11` : `C11` ...) et par la description des fonctions constituant la relation de composition.
La dépendance existentielle est spécifiée par un prédicat dans la classe composante C11Ext :

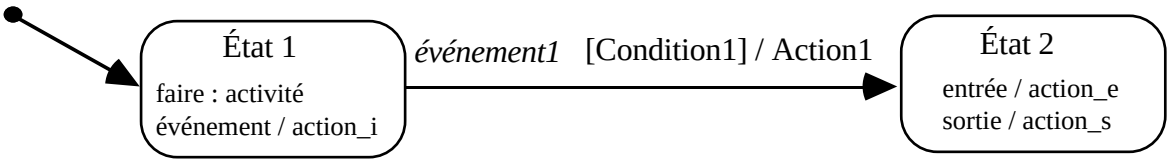
$$\forall\ cs \in C11s.\ (\exists\ c1 \in C1.\ (c1.c11 = cs))$$

où C11s est l'ensemble des objets de `C11` et `C1` est la classe composée,
les deux premiers "." signifie "tel que",
le troisième "." correspond à l'opérateur d'accès aux composants.

IV.1.1.2. Description des règles de formalisation du diagramme d'états

Le modèle dynamique utilisé pour la traduction est le diagramme d'états. Il représente le comportement local de chaque classe d'objets. Les diagrammes d'états sont basés sur la notion d'états et de transitions entre les états.

Ci-dessous sont représentés les concepts utilisés pour décrire les diagrammes d'états :



Un état d'un objet est une abstraction des valeurs de cet objet à un moment particulier. Dans un état, un objet peut subir certaines opérations qui ne changent pas son état. Ces opérations peuvent être une activité (qui nécessite un certain temps d'exécution), une action qui s'exécute à l'apparition d'un événement, une action qui s'exécute à l'entrée ou à la sortie d'un état.

Une transition représente le changement d'état d'un objet vers un nouvel état. Elle est associée à un événement déclencheur, une condition et une action. Deux types d'événements sont considérés : événement de type appel d'opération et événement externe. Dans la figure précédente, l'État 1 est un état initial.

Les diagrammes d'états peuvent aussi être structurés par les liens de généralisation ou d'agrégation. Avec le lien de généralisation, un état peut avoir un sous-diagramme. Quant à l'agrégation, elle exprime la relation entre les états des composants et l'état de l'entité composée.

(i) Traduction d'un état

Pour décrire les états du diagramme d'états associé à une classe, une variable est ajoutée. Son type est un type énuméré qui se compose de tous les états possibles des objets de la classe. En outre, les contraintes exprimant la relation entre chaque état et les attributs de la classe doivent être également ajoutées.

État_Classe ::= État1 | État2 | ... Étatn

État : État_Classe

État = État1 <=> (attribut_i = valeur_i $\wedge$ ...$\wedge$ attribut_j = valeur_j)

(ii) Traduction d'une transition

- La représentation d'un événement dépend de son type :
  - Chaque événement de type appel d'opération donne lieu à une opération dans la classe Object-Z correspondant.
  - Chaque événement externe est décrit par une classe qui est liée à la classe ÉVÉNEMENT par le lien de généralisation. La classe ÉVÉNEMENT possède un attribut représentant le moment où un événement apparaît. Nous obtenons donc :

[MOMENT, OBJET CONCERNÉ]

ÉVÉNEMENT

moment : MOMENT

Événement 1

ÉVÉNEMENT

objet concerné : OBJET CONCERNÉ

- Une opération est définie pour chaque transition afin de modifier l'état en l'état suivant.

Modif État État1 État2

ev ? : $\downarrow$ *ÉVÉNEMENT*

ev ? $\in$ *Événement1* $\wedge$ *État = État1* $\wedge$ *État'= État2*

- Une transition peut être accompagnée par une action (action1 du précédent diagramme d'états). Cette action doit être définie comme une opération dans le modèle d'objets.

- Enfin, une transition est traduite par une conjonction :
  - de l'opération représentant le changement d'états,
  - de l'action sur la transition,
  - de la pré-condition correspondant à la condition sur la transition.

$$\text{TransitionÉtat1\_État2} \cong (\ \text{ModifÉtatÉtat1État2}\ \wedge\ [\text{Condition1}] \wedge\ \text{Action1}\ )$$

(iii) Formalisation du diagramme d'états

Toutes les transitions sont rassemblées dans une opération ConsommerÉvénement qui va décider l'exécution d'une opération correspondant à l'événement reçu. L'opérateur de choix non-déterministe est utilisé ([]) : il va choisir une opération à exécuter si les pré-conditions de plusieurs opérations sont satisfaites.

$$\begin{aligned} \text{ConsommerÉvénement} \cong\ & \text{TransitionÉtat1\_État2}\ [] \\ & \ldots\ \text{TransitionÉtati\_Étati+1}\ [] \\ & \ldots\ \text{TransitionÉtatn-1\_Étatn} \end{aligned}$$

(iv) La visibilité des opérations

Toutes les opérations intermédiaires (ModifÉtatÉtatiÉtatj, TransitionÉtati_Étatj, ...) sont cachées. Seule l'opération ConsommerÉvénement et les opérations de classe sont visibles. Cette visibilité est exprimée par le symbole ↑ :

$$\uparrow(\ \text{ConsommerÉvénement, Opération1, Opérationk}\ )$$

(v) Formalisation de l'agrégation dans les diagrammes d'états

Dans la notation OMT, la sémantique des liens de dépendance entre l'existence des composants et la classe d'agrégat n'est pas explicite. Ce travail propose des règles de traduction concernant le cas où les classes composantes existent indépendamment de leur classe d'agrégat. Donc, chaque composant a son propre diagramme d'états qui ne fait pas référence aux diagrammes d'états d'autres composants. Cependant, l'évolution de la classe d'agrégat dépend du comportement de ses composants. Et la classe d'agrégat peut aussi envoyer les événements à ses composants. Nous avons le diagramme suivant :

Diagramme d'états de la classe d'agrégat "classe 1":

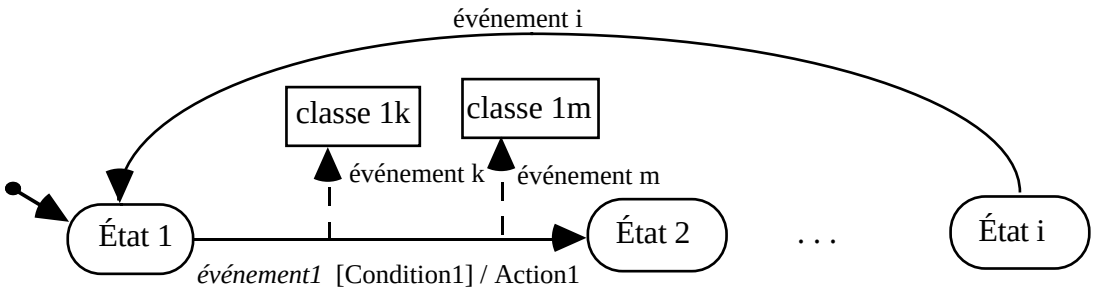
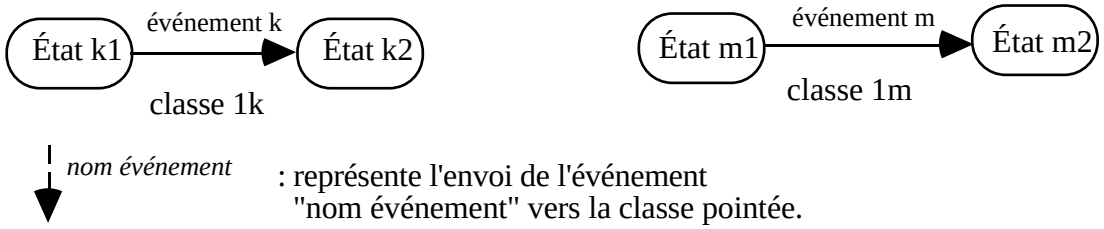


Diagramme d'états des classes composants "classe 1k", "classe 1m" :



Le diagramme d'états de chaque classe composante est traduit en Object-Z selon les règles mentionnées ci-dessus.

Le diagramme d'états de la classe d'agrégat est traduit de la manière suivante :

- Les états :

  La contrainte de la variable État est maintenant définie selon l'état de ses composants :

  État = État_1 <=> classe11.État = État_1i ∧

  . . .

  classe1m.État = État_mj

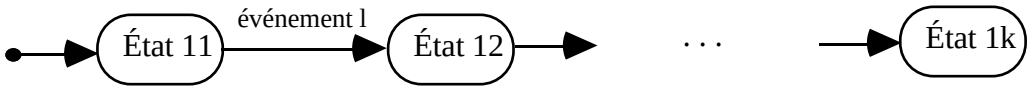
- Les transitions :

  L'envoi d'événements vers les objets composants doit être exprimé. Supposons que événement k de la figure précédente soit un événement externe et que événement m soit de type appel d'opération. La transition de l'État1 à l'État2 est traduite en Object-Z comme suit :

  TransitionÉtat1_État2 ≙ ModifÉtatÉtat1État2
  ∧ [Condition1]
  ∧ ([ev ? : ↓ ÉVÉNEMENT | ev ? ∈ événement 1] ∧ classe1k.ConsommerÉvénement)
  ∧ classe1m.événementm
  ∧ Action1

(v) Formalisation de la généralisation dans les diagrammes d'états

Nous avons ce cas lorsqu'un état possède un sous-diagramme d'états. Un objet dans un état d'un diagramme de haut niveau doit être précisément dans un état du sous-diagramme. Par exemple, le diagramme suivant est un sous-diagramme d'états possible de l'État1 :



Notons qu'il y a aucun lien entre la généralisation dans les diagrammes d'états et celle des objets.

La traduction d'un sous-diagramme d'états est faite de la manière suivante :

- La variable Sous-État est ajoutée dans le schéma de la classe 1. Son type est un type énuméré qui se compose de tous les états du diagramme et, en plus, de l'élément Vide qui est utilisé pour exprimer le lien entre les sous-états et leur super-état.

  Sous-État ::= État11 | État12 | ... État1p | Vide
  Sous-ÉtatÉtat1 : Sous-État
  État = État1 <=> (Sous-ÉtatÉtat1 ∈ {État11, État12, . . ., État1p})
  ∧ ¬(État = État1) <=> Sous-ÉtatÉtat1 = Vide

- Rappelons que chaque transition de l'Étati vers l'État1 donne lieu à l'opération ModifÉtatÉtatiÉtat1. Cette opération doit alors spécifier le fait que la transition arrive aussi à l'État11. Nous avons donc :

Modif État Étati État1

ev ? : ↓ *ÉVÉNEMENT*

ev ? ∈ *Événement i* ∧ *État = État i* ∧ *Sous-État1' = État11* ∧ *État'= État*

- Chaque transition du sous-diagramme d'états est traduite en Object-Z, dans le schéma de la classe 1, en utilisant les mêmes règles que pour un diagramme d'états.

IV.1.1.3. Commentaires

Des règles de dérivation, à partir d'une spécification semi-formelle OMT, d'une spécification formelle en Object-Z ont été proposées pour tous les concepts principaux du modèle d'objets.

Une classe est modélisée en prenant en compte deux aspects : le premier représente l'intention (la description des propriétés communes aux instances possibles) de la classe et le deuxième l'extension (ensemble de toutes les instances existantes). Cela permet de définir les opérations qui portent sur l'ensemble des objets existants (comme la création, la suppression d'instance de la classe), opérations fréquentes dans les applications de base de données.

Une association est formalisée de deux manières. La première consiste à voir une association comme une relation entre plusieurs classes. La deuxième considère une association comme un groupe de liens ayant une structure et une sémantique commune. Ces deux propositions permettent de ne pas créer inutilement un ensemble d'instances possibles pour l'association puisqu'une instance d'association est identifiée par les objets connectés. Elles permettent également de modéliser les associations n-aires.

Toutefois la formalisation en Object-Z présente quelques inconvénients. Les classes en Object-Z ne présentent pas les objets existants de la classe. Il faut donc créer un nouveau schéma pour définir l'extension d'une classe OMT. Les spécifications obtenues peuvent devenir volumineuses.

Concernant la transformation du modèle dynamique de OMT en Object-Z, l'ensemble des concepts du modèle dynamique a été étudié de manière rigoureuse, en particulier les concepts d'agrégation et de généralisation dans les diagrammes d'états. Les règles de traduction de ces deux concepts ont également été proposées. Ces règles n'avaient été abordées dans aucun des travaux précédents. Néanmoins, les solutions proposées présentent des limites :

(i) Le fait de traduire les états par une variable supplémentaires de type énuméré, alors que dans certains cas les états sont exprimables à partir des attributs de la classe, peut donner lieu à des variables redondantes. De ce fait, il est obligatoire d'associer à chaque transition au moins une opération de changement d'états (op_m) et éventuellement une autre opération composée de op_m et de l'action à exécuter sur la transition. Cela produit un nombre d'opérations important qui peut rendre la description d'une classe volumineuse.

(ii) Chaque action sur une transition est supposée être générée à partir du modèle d'objets. Cependant, à partir du modèle d'objets, les règles de transformation en Object-Z

fournissent seulement un squelette de l'opération. Ceci est dû au manque de la description détaillée d'une telle opération dans OMT.

(iii) Les règles de dérivation sont illustrées à travers un exemple où il n'y a pas de notion d'ensemble d'instances de la classe.

IV.1.2. De Fusion vers Z dans [FRA 97]

L'objectif de ce travail est de proposer des règles de traduction des concepts de la modélisation semi-formelle orientée-objet vers les notations formelles. La méthode orientée-objet utilisée est la méthode Fusion [COL 94]. Elle regroupe et étend les concepts des méthodes existantes: OMT [RUM 91], OOAD [BOO 94], Designing Object-Oriented Sofware [WIR 90] et Object-Oriented Software Engineering [JAC 92]. La méthode Fusion a pour objectif de prendre les meilleures idées de ces méthodes afin de former une nouvelle méthode cohérente qui couvre les phases d'analyse, de conception et d'implémentation.

Ce travail étudie la formalisation du modèle d'objets et des modèles d'opération de la phase d'analyse, ainsi que la formalisation du OIG (Object Interface Graph) de la phase de conception.

IV.1.2.1. Description des règles de formalisation du modèle d'objets

Les concepts du modèle d'objets abordés dans ce travail sont les concepts de classe, association, agrégation, généralisation/spécialisation.

(i) Formalisation d'une classe de base
Une classe de base est une classe qui n'est ni une classe d'agrégat, ni une sous-classe d'une spécialisation.

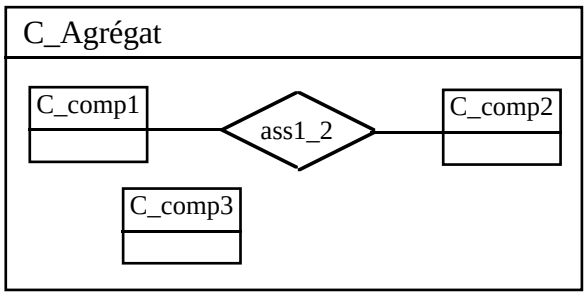
Chaque classe de base donne lieu à un schéma Z. Ce schéma se compose de la description des attributs et des contraintes. Il ressemble beaucoup au schéma Object-Z associé à l'intention d'une classe proposé par [DUP 97]. La différence est qu'ici, le schéma-Z contient en plus une variable représentant l'identifiant de chaque objet.

(ii) Formalisation d'une classe d'agrégat et d'une sous classe

- Classe d'agrégat

Dans la méthode Fusion, une classe qui contient une structure de classes est appelée classe d'agrégat (C_Agrégat). Les classes dans la structure de classes sont ses composants (C_comp_i). Les classes composantes peuvent être liées par le lien d'association ou le lien de généralisation/spécialisation.

Voici le schéma général d'une classe d'agrégat :



La formalisation de la structure interne :

C_AgrégatInterne

c_comp1 : C_comp1
c_comp2 : C-comp2
c_comp3 : C_comp3
ass1_2 : C_comp1 $\leftrightarrow$ *C_comp2*

[contrainte sur ass1_2]

La formalisation de la classe d'agrégat :

[CAID]

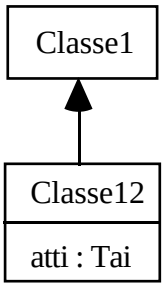
C_Agrégat

C_AgrégatInterne
ident : CAID
attr1 : ATTR1_TYPE ; . . .

[contraintes sur les attributs]

- Sous-classe

  La formalisation d'une sous-classe est faite en incluant le schéma de la super-classe dans le schéma de ses sous-classes.



La formalisation en intention d'une classe :

[Tai] si Tai $\neq \mathbb{N}$

Classe12

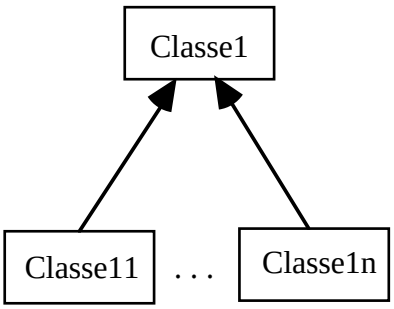
Classe1
atti : Tai

[contraintes sur atti]

(iii) Formalisation des contraintes sur la hiérarchie de spécialisations

Dans la hiérarchie de spécialisations, les sous-classes peuvent être disjointes (c'est-à-dire une instance ne peut pas appartenir à plus d'une sous-classe). Les instances de la super-classe peuvent être contraintes d'être les instances de ses sous-classes (dans ce cas, la super-classe est appelée classe abstraite).

Les propriétés sur la hiérarchie de spécialisations sont spécifiées dans un schéma comme suit :



La formalisation des propriétés de la hiérarchie :

Spec1

cl1 : ℙ *Classe_1*
cl11 : ℙ *Classe_11 ; . . .*

[une ou plusieurs contraintes ci-dessous]

disjoint(cl11, ..., cl1n)
abstrait(cl1, cl11, ..., cl1n)
. . .

(iv) Formalisation d'une association et des contraintes entres les classes
Dans cette dernière étape, un schéma du modèle d'objets est défini. Il contient :

- l'inclusion des schémas de la spécification,
- la définition des ensembles d'objets des classes,
- la formalisation des associations,
- et des contraintes entre les classes.

Nom du modèle d'objets

Spec1 ; . . . [schémas de la spécialisation]
classeobjet_1 : ℙ *Classe_1 ; . . .* [ensembles d'objets des classes]
ass_ij : Classe_i ↔ *Classe_j ;* [associations]

[contraintes de cardinalité et autres contraintes]

IV.1.2.2. Formalisation des modèles d'opérations

La formalisation du modèle d'opérations ne peut pas être automatisée entièrement car la description des opérations est principalement écrite en langage naturel. Seule la partie déclaration du schéma d'opération Z est générée. La description formelle de la relation entre l'état avant et l'état après d'une opération doit être fournie par le concepteur. Cette relation est décrite dans la partie prédicat du schéma d'opération Z.

IV.1.2.3. Formalisation lors de la phase de conception

Au niveau de la conception, certaines classes peuvent être ajoutées en fonction des besoins d'implémentation. Seuls les O.I.G (Object Interaction Graph) sont formalisés. Chaque O.I.G est accompagné par une description en langage naturel. Donc la formalisation ne peut pas être totalement automatisée. La manière de formaliser les O.I.G. est illustrée à travers un exemple. Aucune règle générale n'est donnée.

IV.1.2.4. Commentaires

Tous les éléments qui ne sont pas déjà décrits de manière formelle dans la méthode Fusion sont ainsi formalisés en Z. Concernant le modèle d'objets, la plupart des concepts importants ont été présentés et les règles de formalisation sont proposées. Chaque classe est associée à un schéma-Z qui décrit la définition d'une classe en intention, c'est-à-dire la description des propriétés communes aux objets possibles. Les ensembles d'objets des classes nécessaires à l'expression des contraintes sont spécifiés ensuite dans le schéma-Z de tout le modèle d'objets. Ne pas créer un schéma-Z pour la description en extension de chaque classe facilite la lecture de la spécification. Cependant, les identifiants étant considérés comme des attributs, leurs valeurs peuvent être modifiées. De plus, ce travail est limité à des associations binaires et ne donne pas la formalisation des attributs d'association. Le fait de spécifier toutes les contraintes dans le même schéma pose un problème de visibilité. Les modèles d'opérations et d'O.I.G. ne peuvent être formalisés que par le développeur. La transformation de ces modèles en Z est donnée sous forme d'exemples. Les règles générales ne sont pas données.

IV.1.3. De UML vers Z dans [SHR 97]

Ce travail propose une description de la sémantique des classes UML [UML 97] en notation Z [SPI 92]. Les concepts étudiés sont ceux de classe, d'association, d'agrégation et de généralisation.

IV.1.3.1. Description

(i) Représentation du concept de classe

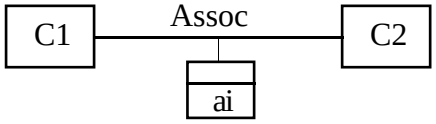
Une classe en UML se compose de deux aspects : l'aspect statique (contenant les attributs et l'identifiant des objets) et l'aspect dynamique (contenant les opérations qui s'appliquent sur les objets de la classe).

Seul l'aspect statique d'une classe a été formalisé. La façon dont une classe est formalisée en Z est la même que dans la partie IV.1.2.

(ii) Représentation du concept d'association

Dans cette étude, une association est définie entre deux classes et peut posséder des attributs.

Chaque association Assoc entre c1 et c2 donne lieu aux schémas Z ci-dessous :



[Ai]

Ai Attrib

ai : Ai

Assoc

$c1 : \mathbb{P}\ C1$

$c2 : \mathbb{P}\ C2$

$assoc : C1 \leftrightarrow C2$

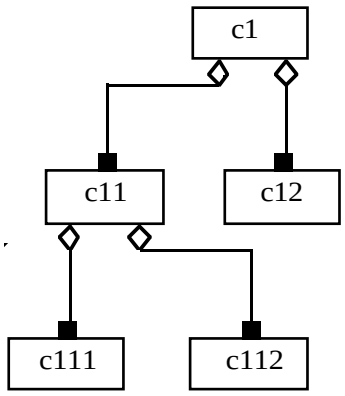
$assoc\_attrib : (C1 \times C2) \leftrightarrow Ai\ Attrib$

---

dom *assoc_Attrib = assoc*
contraintes de cardinalité et
autres contraintes

(iii) Représentation du concept d'agrégation

En UML, le concept d'agrégation est un cas particulier du concept d'association. Il est défini lorsque les cycles de vie des objets sont dépendants.



c1 : est une classe d'agrégation avec les composantes c11, c12.

c11 : est aussi une classe d'agrégation avec les composantes c111, c112.

Il y a deux types d'agrégations :

- agrégation "physique" : nous avons ce cas lorsqu'une instance d'une composante ne peut être liée qu'à une seule instance de sa classe d'agrégat.
- agrégation "catalogue" signifie qu'une instance d'une classe composante peut être partagée entre plusieurs instances de sa classe d'agrégat.

Dans ce travail, la formalisation de ce concept est basée sur l'interprétation suivante :

- c1 et c12 sont liées par une association appelée association d'agrégation.
- La dépendance du cycle de vie signifie que, lorsque les instances de la classe composante, liées à une instance de l'agrégat, sont supprimées :
    - dans le cas de l'agrégation physique, l'instance de l'agrégat et ses instances de la classe composante sont supprimées.
    - dans le cas de l'agrégation catalogue, l'instance de l'agrégat et les instances de la classe composante qui ne sont pas liées aux autres instances de l'agrégat sont supprimées.

Nous avons donc :

c11
ident : c11_ID *ass_c111 : c111* *ass_c112 : c112*

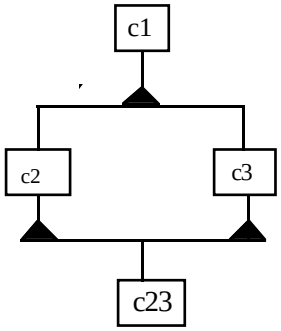
c1
ident : c1_ID *ass_c11 : c11* *ass_c12 : c12*

Les schémas ci-dessus ne formalisent que la structure de l'agrégation. Ils n'expriment pas les contraintes entre les instances de ces différentes classes, imposées par ce concept. Pour la description de ces contraintes, un nouveau schéma a été défini. Il contient la définition des ensembles d'instances existantes de toutes les classes et les prédicats exprimant les contraintes liées à l'agrégation.

Classe_Struct1
c1 : ℙ *C1* *c11* : ℙ *C11* *c12* : ℙ *C12* *c111* : ℙ *C111* *c112* : ℙ *C112*
∀ *e1, e2 : C1* | *(e1.ass_c11 = e2.ass_c11)* ∨ *(e1.ass_c12 = e2.ass_c12)* • *e1 = e2* [contrainte exprimée : deux éléments différents de C1 ne peuvent partager le même composant] . . .

(iv) Représentation du lien d'héritage

Dans ce travail, deux types de lien d'héritage ont été étudiés. Le premier est le cas de lien d'héritage simple et le deuxième , l'héritage multiple.



c2, c3 héritent de c1
(héritage simple)

c23 hérite de c2 et de c3
(héritage muliple)

Chaque super-classe est formalisée comme une classe normale.

Chaque sous-classe est définie dans un schéma Z en ajoutant une variable du type de la super-classe dans la partie déclaration et une variable pour chaque attribut qui n'est pas celui de sa super-classe.

Enfin, un schéma-Z est ajouté pour définir les contraintes associées à la hiérarchie d'héritages. La partie déclaration contient la définition des ensembles d'instances existantes de toutes les classes, la partie prédicat les contraintes associées aux liens d'héritage de la hiérarchie.

IV.1.3.2. Commentaires

Les concepts principaux et leur traduction en Z sont étudiés. Chaque association est définie dans un schéma Z différent. Cela rend la spécification plus lisible et mieux structurée. La formalisation des attributs d'association a été prise en compte. Concernant le concept d'agrégation, la sémantique a été donnée précisément. L'étude de la formalisation du lien d'héritage multiple est présentée. La formalisation de ce concept n'est pas proposée par d'autres méthodes.

Cependant, ce travail ne prend pas en compte les associations n-aires qui sont pourtant fréquentes dans les cas réels. En outre les opérations décrites dans le modèle d'objets n'ont pas été abordées et l'article ne traite pas l'aspect dynamique.

IV.2. Travaux concernant la dérivation d'une spécification formelle B à partir d'une spécification semi-formelle

Nous présentons ici cinq articles choisis pour les raisons suivantes :

- [NAG 94] part du modèle E/R étendu et d'un langage de règles ECA .
- [LAN 94b] utilise le modèle objet et le modèle dynamique de la méthode OMT.
- [HAD 96] génère des spécifications B à partir du modèle NIAM.
- [MAL 98] utilise le diagramme d'objets de la méthode OMT pour décrire un système qui est ensuite spécifié et implémenté en B.
- [SEK 98] part des diagrammes d'états de Harel [HAR 87].

IV.2.1. Du modèle E/R étendu vers B dans [NAG 94]

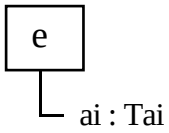
Ce travail propose un outil d'aide à la spécification formelle, qui utilise un modèle E/R [CHE 76, SMI 77] étendu et un langage de règles pour les contraintes d'intégrité et les règles de gestion [DAY 88] comme interface pour développer des spécifications formelles.

IV.2.1.1. Description des règles de formalisation du modèle E/R

Les concepts du modèle E/R étendu étudiés dans ce travail sont les concepts suivants : entité, association, agrégation et héritage.

(i) Formalisation d'une entité et de ses attributs

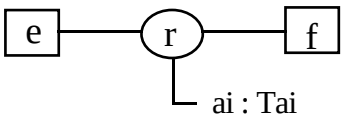
Chaque entité e donne lieu à une variable Ve et est spécifiée dans une machine séparée. Chaque attribut ai de type Tai est traduit par une variable Vai. Nous avons :



Machine	MA_e
Variables	Ve, Vai
Invariant	$Vai \in Ve \rightarrow Tai$

(ii) Formalisation d'une association et de ses attributs

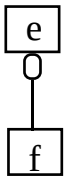
Chaque association r entre les entités e et f donne lieu à une variable Vr et est spécifiée dans une machine comme suit :



Machine	MA_r
Uses	MA_e, MA_f
Variables	Vr, Vai Ver, Vfr
Invariant	$Vai \in Vr \rightarrow Tai \wedge$ $Ver \in Vr \rightarrow Ve \wedge$ $Vfr \in Vr \rightarrow Vf$

(iii) Formalisation d'une entité d'agrégat

Chaque entité d'agrégat est représentée dans une machine qui inclut les machines des entités composantes. Le lien entre l'entité d'agrégat et chaque composant est traduit par une variable. Nous avons :



Machine	MA_e
Includes	MA_f
Variables	Ve, Vfe
Invariant	$Vfe \in Ve \leftrightarrow Vf$

(iv) Formalisation d'un lien d'héritage

Un lien d'héritage est traduit par le lien *Includes* de la machine représentant la super-entité dans la machine de la sous-entité et une contrainte d'inclusion entre les ensembles d'instances correspondant.

Machine	MA_f
Includes	MA_e
Variables	Vf
Invariant	$Vf \subseteq Ve$

IV.2.1.2. Description des règles de formalisation des contraintes d'intégrité et des règles de gestion

Le langage des règles est utilisé pour décrire les contraintes d'intégrité (C.I.) non exprimables graphiquement ainsi que les règles de gestion.

Les C.I. sont décrites par des formules du cacul des prédicats du premier ordre. Chaque C.I. va être traduite en B par un invariant de la machine concernée. Les règles de gestion, quant à elles, sont spécifiées par les règles ECA (Événement-Condition-Action). La syntaxe d'une règle est : ON Événement IF Condition THEN Action. Sa sémantique exprime : quand l'Événement apparaît, si la Condition est vérifiée, l'Action doit être exécutée. La partie Action est composée à partir des opérations de base de mise à jour telle que l'insertion, la suppression ou la modification. Ces opérations de base sont déduites automatiquement à partir de la

spécification B du diagramme E/R et elles doivent vérifier les C.I. Une règle de gestion est traduite ensuite par une opération B comme suit :

(i) L'événement (partie ON) est représenté par les variables logiques qui valent vrai quand l'événement est déclenché. La partie Événement est traduite ensuite en la pré-condition de l'opération avec, en outre, la vérification du typage de ses paramètres.

(ii) La partie Condition est traduite par une conditionelle (structure IF) dans le corps de l'opération.

(iii) La partie Action détermine les substitutions dans le corps de la conditionnelle.

VI.2.1.3. Commentaires

Concernant l'aspect statique, les règles de transformation du modèle E/R étendu ont été définies. La formalisation d'une association permet de prendre en compte les association n-aires. L'héritage est modélisé par l'inclusion des machines. Par cette manière de formaliser, la machine d'une sous-entité hérite également des opérations de la machine de la super-entité. Cependant, cette méthode soulève de nombreux problèmes :

(i) Il n'y a pas de distinction entre instances possibles et instances existantes d'une entité. En fait, le typage d'une entité semble avoir été oublié.

(ii) Les associations sont modélisées par des ensembles d'identités avec des fonctions de projection, mais les contraintes données ne sont pas suffisantes : avec la définition donnée, deux objets peuvent être reliés par deux liens de la même association et un lien peut changer d'objets reliés (tout en demeurant le même lien).

De plus, cette solution consiste à créer systématiquement une machine pour chaque association. Cela produit trop de machines, certaines inintéressantes. Par exemple, pour les associations qui ne possèdent pas d'opérations propres, il n'est pas utile de leur associer une machine.

(iii) L'héritage est modélisé par l'inclusion d'ensembles et l'inclusion de machines abstraites. Cette solution présente un inconvénient dans le cas fréquent où une entité e a plusieurs sous-entités e1, ... en et si nous voulons exprimer les C.I. entre les sous-entités. Par exemple, $Ve1 \cap ... \cap Ven = Ø$. Cette contrainte ne peut être exprimée que dans la

machine Interface qui inclut toutes les machines de la hiérarchie d'héritage. Or le lien *Includes* est transitif, donc la machine MA_e est incluse n fois (par MA_e1, ..., MA_en) dans la machine Interface. Ce qui entraîne une multiplication des ensembles d'instances existantes d'entités. Cette solution n'est donc pas satisfaisante.

Concernant l'aspect dynamique, une description de la dérivation, à partir d'un ensemble de C.I., non exprimables graphiquement, d'une spécification B est proposée. Cet aspect a été peu abordé par d'autres auteurs. Cependant, cette proposition présente des problèmes :

(i) Aucune règle n'est spécifiée pour la génération automatique des opérations de base.

(ii) Il est proposé de spécifier les événements par des variables logiques mais aucune précision n'est donnée.

(iii) L'auteur n'a pas pris en compte le cas où une opération de la machine Interface fait appel à plusieurs opérations d'une même machine entité ou association, cas interdit en B.

IV.2.2. De OMT vers B dans [LAN 94b]

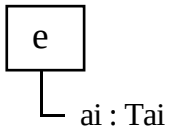
Cet article présente une approche pour la formalisation et le raffinement des modèles OMT [RUM 91] en B. Les modèles OMT utilisés sont le diagramme d'objets pour la description de la statique et les diagrammes états/transitions pour l'aspect dynamique.

IV.2.2.1. Description des règles de formalisation du modèle d'objets

Les concepts étudiés dans cet article sont : entité, association, et héritage.

(i) Formalisation d'une entité et de ses attributs

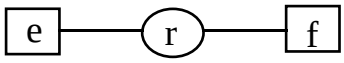
Chaque entité e et ses attributs ai (i=1..n) de type Tai sont spécifiés dans la machine MA_e comme suit :



Machine	MA_e
Sets	Se
Variables	Ve, Vai
Invariant	$Ve \subseteq Se \wedge$ $Vai \in Ve \rightarrow Tai$

(ii) Formalisation d'une association

Chaque association r entre les entités e et f est traduite par une variable Vr dans une des machines entités. Cette variable est une relation entre les entités connectées.



Machine	MA_e
Uses	MA_f
Sets	Se
Variables	Ve, Vr
Invariant	$Ve \subseteq Se \wedge$ $Vr \in Ve \leftrightarrow Vf$

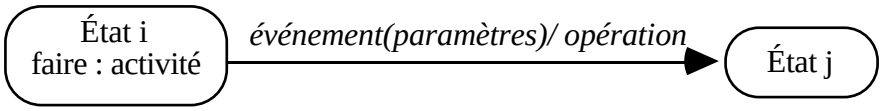
(iii) Formalisation d'un lien d'héritage

Le lien d'héritage est traduit par l'inclusion d'ensembles d'instances.

Machine	MA_e
Sets	Se
Variables	Ve, Vf
Invariant	$Ve \subseteq Se \wedge$ $Vf \subseteq Ve$

IV.2.2.2. Description des règles de formalisation du modèle dynamique

Les diagrammes utilisés dans ce travail pour décrire l'aspect dynamique sont sous forme de diagrammes états/transitions. Chaque diagramme possède un ensemble d'états et un ensemble de transitions entre les états. Chaque transition est accompagnée par une opération. Chaque état peut être accompagné par une activité. Elle sera exécutée immédiatement quand l'objet arrive dans cet état. Voici le schéma général :



(i) Une variable est ajoutée pour chaque état, notée Si. Elle représente l'ensemble d'objets dans cet état. Cette variable respecte l'invariant $Si \subseteq Ve$. Les ensembles des états sont deux à deux disjoints et l'union des ensembles de tous les états est égale à l'ensemble des objets de l'entité. Nous avons donc :

$$S1 \cup \ldots \cup Sn = Ve \quad \wedge$$
$$Si \cap Sj = \emptyset \quad \forall \quad i \neq j$$

(ii) Une transition entre deux états est traduite par une opération qui transforme l'objet d'un état dans un autre.

(iii) La pré-condition d'une transition devient la pré-condition de l'opération.

(iv) Une activité dans un état devient une opération qui ne change pas d'état.

IV.2.2.3. Commentaires

Concernant l'aspect statique, ce travail s'appuie sur la formalisation de trois concepts principaux : entité, association et héritage. Les solutions proposées ont quelques limites :

(i) La décomposition en machines n'est pas très satisfaisante : ni les associations, ni les sous-classes n'ont leurs propres machines, et donc une machine peut inclure trop d'informations.

(ii) Pour le lien d'héritage, le fait que toutes les variables d'une même hiérarchie se retrouvent dans la même machine fait perdre le bénéfice de la spécification incrémentale apporté par l'héritage.

(iii) Aucune règle n'est donnée pour la traduction des divers types de contraintes (par exemple, les contraintes de cardinalité et les contraintes sur le lien d'héritage).

(iv) Ni la formalisation des attributs d'association, ni la formalisation des associations n-aires n'ont été étudiées.

Concernant l'aspect dynamique, l'auteur a pris comme point de départ les diagrammes états/transitions. Ces diagrammes sont les plus répandus pour décrire l'aspect dynamique. Cependant, ce travail pose quelques problèmes :

(i) La façon de spécifier les états augmente systématiquement le nombre de variables car il est nécessaire d'ajouter une variable par état. Le nombre d'opérations est donc également augmenté car chaque transition est obligatoirement liée à une opération qui change explicitement l'état courant, même si cela n'est pas spécifié par le concepteur. La spécification obtenue n'est donc pas tout à fait fidèle au modèle semi-formel de départ (où un changement d'état peut être implicite).

(ii) Aucune règle n'est donnée pour la traduction des événements.

(iii) Le principe de la traduction de la pré-condition d'une transition est donné. Cependant, aucun moyen n'est donné pour spécifier une pré-condition dans les diagrammes états/transitions.

(iv) Les obligations de preuves qui permettent de préserver l'invariant et d'aboutir à l'état final de chaque opération ne sont pas abordées.

IV.2.3. De NIAM vers B dans [HAD 96]

Ce travail présente les techniques pour la génération des spécifications formelles B à partir du modèle NIAM [HAB 93]. La spécification obtenue contient non seulement la structure des données du modèle NIAM mais aussi certaines opérations de base respectant les contraintes graphiques.

IV.2.3.1. Description des règles de formalisation du modèle NIAM

Les concepts étudiés par ce travail sont les concepts de classe, de relation et les contraintes graphiques. Deux types de contraintes ont été abordés : les contraintes concernant une relation, par exemple, la fonctionnalité partielle ($\rightarrow\!\!\!\!\!\!+$), totale ($\rightarrow$) ..., et celles impliquant plus d'une relation comme les contraintes d'exclusion ($\otimes$), d'égalité(=), de totalité($\vee$) ...

(i) Chaque classe c donne lieu à un ensemble C et une variable c. L'ensemble C représente les objets possibles (présents ou qui vont être ajoutés dans la base de données). La variable c, sous-ensemble de C, représente les objets existants de la classe.

(ii) Chaque relation r entre c1 et c2 est traduite par une variable.

(iii) Les contraintes graphiques sont formalisées par des prédicats dans la partie *Invariant.*

(iv) Les variables sont initialisées par des ensembles vides.

IV.2.3.2. Description des opérations de base

Seuls les différents types d'opération d'ajout de lien ont été abordés. Ces opérations sont déduites automatiquement à partir de diagrammes d'objet correspondant à un sous-système. Ce type de diagramme se compose de deux relations ayant une classe en commun. La génération de ces opérations prend en compte les contraintes d'égalité et de sous-ensemble entre deux relations. La manière dont elles sont spécifiées est illustrée à travers des exemples.

Ensuite, un algorithme qui permet de définir des opérations d'un système global à partir des opérations de ses sous-systèmes, est également proposé.

IIV.2.3.3. Commentaires

Tous les cas de contrainte concernant une relation entre deux classes sont présentés avec leur formalisation en B. La génération des opérations de base a été étudiée. Notons que ces points n'avaient pas été étudiés de manière précise dans les autres travaux. Cependant, ce travail présente certaines limites :

(i) La spécification obtenue est contenue dans une seule machine, ce qui peut rendre la spécification illisible dans les applications de taille importante. En fait, la modularisation n'était pas l'objet de ce travail.

(ii) L'étude générale des opérations de création, de suppression d'instances d'entité, de suppression de lien d'association n'a pas non plus été abordée.

IV.2.4. De OMT vers B dans [MAL 98]

Cet article propose une spécification et une implémentation en B du système ACVIS (A Computerized Visitor Information System). La spécification B est obtenue à partir d'une spécification semi-formelle utilisant le diagramme d'objets de la méthode OMT.

IV.2.4.1. Description des règles

(i) Chaque entité est spécifiée dans une machine séparée. L'ensemble des instances d'une entité est spécifié dans la partie *Définition* comme un ensemble d'entiers.

(ii) Chaque attribut d'une classe est représenté par une variable.

(iii) Chaque association du même type (même contraintes de cardinalité) et sans attribut est définie une fois pour toutes dans une machine MA_ri. Cette machine contient :

Machine	MA_r
Variables	Vr
Invariant	Vr ∈ XX ↔ YY(ou plus contrainte)
Initialisation	Vr := {}
Definitions	XX ≙ ℕ ; YY ≙ ℕ
Operation	Ajout_Lien(xx, yy) ≙ ...
	Supprime_Lien(xx) ≙ ... End

(iv) Les contraintes d'intégrité sont spécifiées dans la partie *Invariant*.

(v) Les opérations peuvent être générées selon les différents scénarios (un scénario est une séquence d'opérations qui apparaissent durant une exécution particulière du système). Dans la pré-condition est déclarée le typage des paramètres de l'opération.

(vi) La machine Interface regroupe tous les paramètres réels de l'opération.

(vii) La machine Interface inclut toutes les machines du système.

IV.2.4.2. Commentaires

Le fait de pouvoir regrouper des associations du même type pour définir une machine qui peut être incluse par plusieurs associations permet de réutiliser des machines. Le nombre de machines est moins important, il y aura donc moins de preuves. Cependant, ce travail possède certaines limites :

(i) La règle concernant la spécification de chaque entité dans une machine séparée n'a pas été respectée dans l'exemple proposé.

(ii) L'auteur a donné les règles de traduction des attributs de classe dans le cas où leur type est un sous-ensemble des instances des entités. Concernant les attributs d'instance, aucune précision n'est donnée.

(iii) Le type de chaque entité est défini comme un ensemble d'entiers alors que cela n'est pas imposé dans la méthode OMT. Il y a donc sur-spécification. De plus, aucune vérification n'est effectuée concernant la disjonction entre les ensembles d'objets des entités.

(iv) Les opérations sont supposées être générées à partir des scénarios mais aucune description semi-formelle n'est donnée pour exprimer ni les scénarios, ni les opérations.

Le travail proposé ne semble pas pouvoir être appliqué tel quel à n'importe quel diagramme d'objets OMT de départ.

IV.2.5. Des diagrammes d'états de Harel vers B dans [SEK 98]

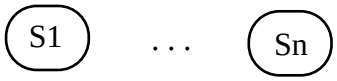
Ce travail présente des règles de traduction des diagrammes d'états suivant les définitions données par Harel dans [HAR 87] en machine abstraite B. L'auteur s'intéresse particulièrement aux systèmes réactifs, c'est-à-dire ayant constament à réagir aux stimuli de l'environnement comme les systèmes distribués, les systèmes embarqués ou les systèmes

temps réels. Dans un premier temps, une version simple des diagrammes d'états, dans laquelle les états, les événements, les conditions, les actions sont pris en compte, est considérée. Puis l'auteur propose des règles de traduction pour les diagrammes d'états hiérarchiques, les diagrammes avec concurrence et les diagrammes avec communication. Nous ne décrivons ici que les règles concernant la version simple des diagrammes d'états auxquels se rapporterons les comparaisons avec nos règles (cf. chapitre 5).

IV.2.5.1. Description des règles

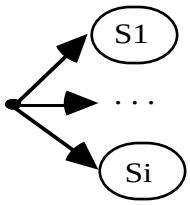
Les conventions graphiques utilisées ici pour représenter les états, les événements (avec ou sans paramètres), les transitions, les conditions et les actions sont les mêmes que celles employées précédamment pour décrire les diagrammes d'états de OMT (cf. IV.1.1.2). En effet, les diagrammes d'états utilisés par OMT sont inspirés des définitions de [HAR 87].

(i) Formalisation d'un état



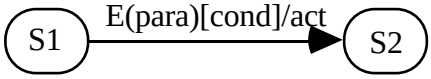
Sets S = {S1, ..., Sn}
Variables s
Invariant s∈ S

(ii) Formalisation de l'initialisation des états



Initialisation s :∈ {S1, ..., Si}

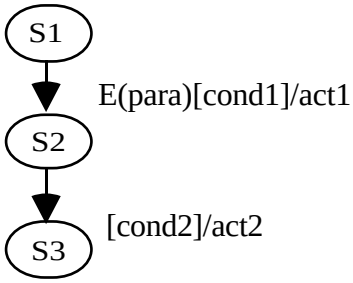
(iii) Formalisation d'une transition avec événement, condition et action



Operations
E(para) =
IF s=S1 ∧ cond THEN
s :=S2 || act
END
END

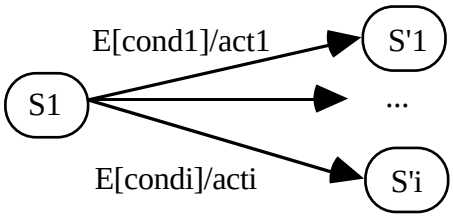
(iv) Formalisation d'une transition automatique

Dans le diagramme ci-dessous, la flèche de l'état S2 à l'état S3 est une transition automatique




```
Operations
   E(para) =
      IF  s=S1 ∧ cond1 ∧ cond2   THEN
            s :=S3 ||  act1  ||  act2
      END
   END
```

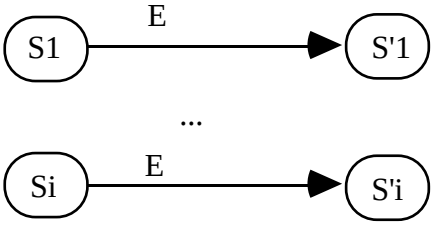
(v) Formalisation des transitions partant d'un même état et déclenché par le même événement




```
Operations
   E =
      SELECT  s=S1  ∧cond1   THEN
            s :=S'1 ||  act1
      ...

      WHEN    s=S1  ∧condi   THEN
            s :=S'i  || acti
      ELSE      skip
      END
   END
```

(vi) Formalisation des transitions partant de différents états et déclenchées par le même événement




```
Operations
   E =
      CASE  s  OF
         EITHER  S1   THEN  s:= S'1
            s :=S'1 ||  act1
         ...
         OR  Si     THEN   s:= S'i
      END
   END
```

IV.2.5.2. Commentaires

Les règles présentées dans ce travail sont très précises et couvrent la quasi totalité des cas. Toutefois, ce travail concerne uniquement les applications dans lesquelles on s'intéresse principalement à décrire le fonctionnement du système et dans lesquelles les données sont simples. En effet, l'aspect statique n'étant pas pris en compte, cette approche n'est pas applicable aux systèmes d'information orientés gestion de données. Notons par ailleurs que la modularisation de la spécification obtenue n'a pas été abordée.

V. Synthèse générale

Cette partie est consacrée à la discussion et à la comparaison des solutions proposées dans les travaux précédemment étudiés. Nous adoptons des critères à la fois quantitatifs et qualitatifs. Nos critères quantitatifs portent sur le choix des modèles et des concepts traités. Nos critères qualitatifs étudient la façon dont les concepts sont formalisés.

V.1. Comparaison selon des critères quantitatifs

Les méthodes de spécification semi-formelle présentent deux aspects principaux : l'aspect statique et l'aspect dynamique. À la lecture d'un article concernant la dérivation, à partir d'une spécification semi-formelle, d'une spécification formelle plusieurs questions se posent :

1. Concernant l'aspect statique, quel sont les concepts considérés ? Les opérations de base qui peuvent être générées à partir du modèle statique le sont-elles ?
2. L'aspect dynamique est-il ou non étudié ?
3. L'article prend-il en compte la modularisation de la spécification obtenue ?

Dans la suite, nous y répondons pour chaque article sus-mentionné.

V.1.1. L'aspect statique

La terminologie utilisée dans les différentes méthodes de spécification semi-formelle existantes est très variée et le même concept porte souvent plusieurs noms suivant les différents auteurs. Par exemple, le concept d'entité dans le modèle E/R [CHE 76] correspond au concept de classe dans le modèle OMT, Fusion et UML. Comme la plupart des concepts se retrouvent dans les différents méthodes, il est intéressant de savoir quels sont ceux qui sont traités par les différents auteurs.

Par ailleurs, à partir de la description des concepts et des contraintes dans le modèle statique, il est possible de déduire certaines opérations de base de mise à jour comme les opérations de création, de suppression d'instances d'entité ou de liens d'association, les opérations de modification de valeurs d'attributs. Cependant, la génération de ces opérations n'est pas systématique dans ces différents travaux.

Le tableau ci-dessous indique, pour chaque travail, les concepts étudiés et précise si les opérations de base sont générées. Notons que ce tableau ne mentionne pas le travail de [SEK 98] car les diagrammes étudiés ne concernent que l'aspect dynamique.

Concepts / Articles	Entité (Classe)	Attribut d'entité	Association (Relation)	Attribut d'association	Association n-aires	Agrégation	Héritage	Contraintes graphiques	Opérations de base
[NAG 94]	T	T	T	T	P	T	T⁻		A
[LAN 94b]	T	T	T				T⁻		
[HAD 96]	T		T					T⁻	T⁻
[MAL 98]	T	T	T						
[DUP 97]	T	T	T	T	T	T	T⁻		A
[FRA 97]	T	T	T			T	T⁻		A
[SHR 97]	T	T	T	T		T	T⁻		

Figure 2.1. *Tableau de synthèse de l'aspect statique*

NB : T : le concept est traité complètement
T⁻ : le concept n'est que partiellement traité
A : le concept est simplement abordé
P : le concept n'est pas traité mais pourrait l'être dans la démarche adoptée

V.1.2. L'aspect dynamique

Tous les auteurs étudiés ne choisissent pas de traiter l'aspect dynamique. Le tableau ci-dessous ne mentionne que les travaux qui examinent cet aspect et indique les concepts pris en compte.

Articles	Modèle semi-formel utilisé	Concepts pris en compte						
		Événement	État	Condition	Opération	Généralisation diagrammes E/T	Concurrence d'objets	Échange d'événements
[NAG 94]	Langage de règles E.C.A.	T		T	T			
[LAN 94b]	Diagrammes E/T de OMT		T	T	T			
[SEK 98]	Diagrammes d'états de Harel	T	T	T	T	T		T
[DUP 98]	Diagrammes E/T de OMT	T	T	T	T	T	T	T

Figure 2.2. *Tableau de synthèse de l'aspect dynamique*

V.1.3. La modularisation

La modularisation consiste à décomposer la spécification en plusieurs unités. Elle présente les intérêts suivants :

- elle rend possible une spécification incrémentale,
- elle améliore la compréhension de la spécification obtenue,
- elle facilite l'évolution de la spécification,
- elle permet la réutilisation des composants,
- elle simplifie les preuves.

Rappelons qu'en B, l'unité de spécification est une *machine abstraite*. Le lien entre les unités est assuré notamment par les clauses *Includes* et *Uses*. Parmi les travaux concernant la dérivation d'une spécification formelle B que nous avons cités, la modularisation de la spécification obtenue a été prise en compte dans [NAG 94], [LAN 94b] et [MAL 98].

En Z, la structure de *schéma* est un moyen d'obtenir une spécification modulaire. Remarquons que tous les auteurs étudiés se sont efforcés d'associer à chaque concept un ou plusieurs schémas. Ce qui traduit leur souci de modularisation.

V.2. Comparaison selon des critères qualitatifs

Dans cette synthèse, nous nous sommes jusqu'ici intéressée à la richesse des concepts pris en considération dans notre étude bibliographique. Pour ne pas nous arrêter à une vision trop superficielle de la formalisation des spécifications semi-formelles, il importe de tenter d'évaluer la façon dont la spécification formelle a été obtenue à partir des modèles semi-formels. Nous reprenons ici les points soulevés dans la comparaison quantitative (aspects statique, dynamique et modularisation) en les approfondissant. En outre, nous allons examiner, dans cette partie, la prise en compte des preuves de cohérence de la spécification qui constitue une garantie de qualité finale.

V.2.1. L'aspect statique

Il est intéressant de souligner que tous les travaux étudiés précédemment proposent une formalisation fidèle à l'approche par objets car ils font bien la distinction entre les objets possibles et leurs caractéristiques, en considérant l'ensemble des objets possibles comme un ensemble donné sans aucune structure pour leurs éléments : il s'agit de la prise en compte de la notion d'identité d'objet. Or certains travaux n'utilisent pas cette notion, ou pas systématiquement. Par exemple dans SAZ [POL 93] qui développe le passage de la méthode de conception SSADM à Z, les associations sont définies entre valeurs et non entre identités d'objet. Certes cela est d'abord lié au fait que SSADM est orienté valeur et non objet.

Cependant, au cours de notre étude plusieurs problèmes ont été rencontrés. Les solutions proposées pour la formalisation du concept d'héritage permettent seulement un héritage d'attributs. L'héritage d'opération n'a pas été précisé. De plus :

- La distinction entre instances possibles et instances existantes n'est pas toujours faite ([NAG 94]),
- [MAL 98] a défini le type de chaque entité comme un ensemble d'entiers. Cependant, un tel type n'est pas imposé dans OMT,
- Dans [FRA 97 & SHR 97], les identifiants sont considérés comme des attributs donc leur valeur peut être modifiée,
- Dans le travail de [NAG 94], les associations sont modélisées par des ensembles d'identités avec des fonctions de projection. Mais cette formalisation ne suffit pas pour exprimer les contraintes d'une association,

- La modélisation de l'héritage proposée par [NAG 94] entraîne une multiplication des ensembles d'instances existantes de la super-entité,
- D'une façon générale, les règles de génération de différents types d'opérations de base ne sont pas explicitement décrites.

V.2.2. L'aspect dynamique

Nous soulignons les points délicats suivants :

- Lorsque le concept d'événement existe dans le modèle utilisé, sa traduction manque de clarté : absence de règles de traduction [LAN 94b] ; manque de précision sur la description de l'événement [NAG 94].

- Le choix de spécifier les états dans [LAN 94b] et [DUP 98] peut donner lieu à des variables redondantes. Or le changement d'état doit toujours être assuré par un appel d'opération. Par conséquent, le nombre d'opérations est également augmenté. La spécification obtenue n'est donc pas toujours fidèle au modèle semi-formel initial. On ne rencontre pas ce problème dans [SEK 98] car étant donné que l'aspect statique n'est pas pris en compte, les variables ajoutées pour représenter les états n'occasionnent pas de redondance.

- En raison de contraintes techniques de B, une opération de la machine Interface ne peut pas faire appel à plusieurs opérations d'une machine incluse. Or cette contrainte n'est pas prise en compte dans [NAG 94].

V.2.3. La modularisation

La modularisation présente aussi certains inconvénients :

- En Z, un schéma ne présente pas les deux aspects intention et extension d'une classe. Pour chaque classe, deux schémas sont donc nécessaires, l'un représente l'intention de la classe et l'autre définit l'extension. Ceci rend la spécification finale plus lourde. À l'inverse, en B, ces deux aspects sont regroupés dans une seule structure : la machine abstraite.

- Le fait de créer systématiquement une machine pour chaque association dans [NAG 94] produit un nombre de machines important. Or il est inutile d'associer une machine aux associations qui ne possèdent pas d'opérations propres : les machines créées correspondantes sont inintéressantes.

- À l'inverse, [LAN 94b] propose de mettre l'association dans une machine d'entité (sans d'ailleurs préciser laquelle) ce qui produit une surabondance d'information dans une même machine.

- Toujours dans [LAN 94b], la présence de toutes les variables d'une même hiérarchie dans une même machine fait perdre le bénéfice de la spécification incrémentale apporté par l'héritage.

V.2.4. Les preuves

Les preuves permettent de s'assurer de la cohérence de la spécification. En effet, les opérations modifient les données. Or, il peut exister des contraintes portant sur ces données : il s'agit des invariants du système. Les preuves consistent à vérifier que les opérations préservent les invariants.

Dans tous les travaux étudiés concernant B, la préservation des invariants par les opérations a été abordée. Par exemple, dans [NAG 94] les preuves à apporter concernent les opérations générées à partir des règles de gestion. Dans [HAD 96], les contraintes graphiques entre les associations doivent être respectées par les opérations de base. Notons toutefois que [LAN 94b] et [SEK 98] qui utilisent les diagrammes états/transitions ne mentionne pas les obligations de preuve permettant de vérifier que chaque opération sur une transition aboutit bien à l'état d'arrivée.

En revanche, les travaux sur Z ont peu abordé le côté preuve, peut être du fait que Z est moins bien outillé que B. Cependant [DUP 98] justifie le choix d'un seul langage de spécification par le fait qu'il rend possible les preuves de cohérence entre les différents aspects de la spécification.

V.3. Conclusion

Tout au long de cette étude bibliographique, nous avons tenté de proposer un panorama des travaux existants dans la dérivation d'une spécification formelle à partir d'une spécification semi-formelle dans les systèmes d'information. Les articles ont été choisis dans le but de représenter le plus complètement possible les solutions apportées dans la littérature. Il est à noter que les travaux étudiés restent sur un plan théorique et qu'actuellement aucun repère précis pour la réalisation d'un outil complet permettant cette dérivation n'est donné. La conception d'un tel outil nécessiterait en effet des règles de passage précises qui ne sont pas toujours proposées.

Dans la synthèse générale, les trois aspects à prendre en compte (à savoir l'aspect statique, l'aspect dynamique et la modularisation) ont été examinés tant sous l'angle quantitatif que qualitatif.

Cette étude a permis de mettre en évidence les points nécessitant un plus grand approfondissement :

- La manière de spécifier les états,
- La prise en compte des associations n-aires,
- La formalisation des contraintes d'intégrité,
- La manière de spécifier les opérations,
- La génération de différents types d'opérations de base,
- La modularisation de la spécification,
- Les obligations de preuve.

Notre travail vise donc à combler ces manques et à répondre aux divers problèmes rencontrés lors de cette étude.

Chapitre 3

Étude des Méthodes Semi-Formelles

I. Introduction

En vu d'étudier la dérivation d'une spécification formelle à partir d'une spécification semi-formelle, ce chapitre est consacré à une étude approfondie des concepts des méthodes semi-formelles. Cette étude a pour but de retenir les concepts les plus répandus des méthodes semi-formelles et d'examiner en détail leur sémantique. Cela nous a amenée parfois à faire un choix entre plusieurs interprétations possibles des notations semi-formelles et à les compléter. Les méthodes semi-formelles ont été créées pour représenter n'importe quel type de systèmes d'information autrement qu'avec une description en langue naturelle. Elles utilisent divers schémas dans le but d'obtenir une description la plus fidèle possible du système. Ainsi, cette description apparaît moins ambiguë, plus synthétique et plus intuitive que celle en langue naturelle. Nous trouvons toujours dans ces méthodes deux modèles principaux. Le premier décrit la structure des données et leurs relations dans le système. Ce modèle est communément appelé le modèle d'objets. Il est représenté graphiquement par les diagrammes d'objets. Le deuxième décrit les traitements qui ont lieu sur les données. Leurs auteurs lui donnent souvent le nom de modèle dynamique. Ce modèle est représenté graphiquement par les diagrammes états/transitions. Une description complète de ces deux modèles est donnée dans ce chapitre.

II. Le modèle d'objets

Comme nous l'avons déjà mentionné, un vocabulaire différent est souvent utilisé dans la littérature pour désigner des concepts pourtant sémantiquement équivalents. Un premier travail a été d'étudier les différents concepts du modèle d'objets suivant les différentes méthodes et d'en extraire les concepts les plus répandus. En effet chacune de ces méthodes utilise un vocabulaire différent et le cadre d'utilisation n'est pas toujours clairement établi. Le modèle d'objets que nous définissons dans cette partie n'est pas nouveau : nous avons retenu les concepts les plus répandus tels que objet, entité, association, héritage. D'autres concepts plus élaborés tels que qualification, rôle, entité associative ... sont inspirés d'OMT [RUM 91]. Nous avons également étudié les différentes contraintes d'intégrité inter-associations proposées à l'origine dans le modèle NIAM [HAB 93] avec une représentation graphique.

II.1. Objet et concepts liés

II.1.1. Objet

La notion d'objet que nous utilisons relève de la modélisation du monde réel et de la représentation des connaissances. C'est donc une notion très large qui recouvre toute abstraction du monde réel qui peut être identifiée comme pertinente pour l'application ou plutôt le domaine d'application considéré.

II.1.2. Entité

Le concept d'entité est lui utilisé pour structurer les connaissances sur les objets : une entité regroupe les objets partageant des caractéristiques (attributs et opérations) communes. Nous préférons le terme d'entité qui relève de la modélisation à celui de classe, plus orienté programmation. Un objet est une instance d'une entité. Une entité est représentée graphiquement par un rectangle avec son nom.

Exemple :

Personne, Commande, Client ... sont des entités. Et Jean est une instance de l'entité Personne.

II.1.3. Attribut

Un attribut d'une entité est défini par son nom et son type qui définit l'ensemble de ses valeurs possibles. Les types des attributs peuvent être simples (entiers, booléens, chaînes de caractères) ou complexes (en utilisant les constructeurs ensembles, séquences, tuple…).

Un attribut peut être obligatoire ou optionnel. Si, à la création d'un objet quelconque d'une entité, un attribut doit toujours être valorisé, alors il est considéré comme *obligatoire*. Dans le cas contraire, il est *optionnel.*

Exemple :

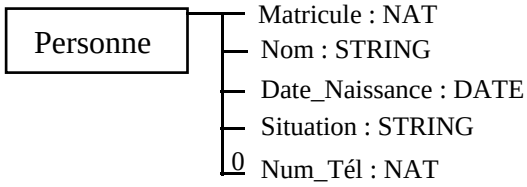


Figure 3.1. *Un exemple d'entité avec cinq attributs*

Dans cet exemple, Personne est une entité avec cinq attributs : le numéro de matricule (Matricule) de type NAT, le nom (Nom) de type STRING, la date de naissance (Date_Naissance) de type DATE, la situation (Situation) de type STRING et le numéro de téléphone (Num_Tél) de type NAT. L'attribut Num_Tél est optionnel car une personne peut ne pas avoir de téléphone.

II.1.4. Identité d'objet, Clé

Un aspect absolument fondamental de la modélisation par objets est la notion d'identité d'objet : les objets existent indépendamment des différentes valeurs qui leur sont attachées. Une personne peut changer de nom, d'adresse, de numéro de téléphone, ... il s'agit toujours de la même personne. Même une adresse, usuellement considérée comme une valeur composée (nº rue, nom de rue, ville, code postal) peut gagner à être considérée comme un objet (qui serait une instance d'une entité "Coordonnées géographiques") : la rue Lénine peut être rebaptisée en rue Charles de Gaulle, une personne habitant cette rue n'a pas pour autant changé d'adresse : l'adresse est restée la même mais une de ses caractéristiques (le nom de la rue) a été modifiée. Dans une spécification formelle adoptant cette approche, une opération de modification de nom de rue n'a aucune autre modification à répercuter sur les habitants de la rue.

Une clé est un sous-ensemble d'attributs dont les valeurs déterminent de manière unique un objet.

Exemple : Dans l'exemple précédent, l'attribut Matricule est une clé de l'entité Personne.

II.2. Association et concepts liés

II.2.1. Lien, Association

Il existe des liens entre objets : lien de type possède entre une personne et une voiture, lien de type auteur_de entre une personne et un livre, lien de type lignedecommande entre une commande et un produit, etc. Dans le modèle que nous retenons, un lien concerne exactement deux objets, un lien est donc un couple. Un lien peut évidemment posséder des attributs : date d'achat pour possède, quantité_commandée pour lignedecommande.

Un type de lien est appelé association. Pour une association donnée, un lien est totalement identifié par le couple d'objets connectés par ce lien. Une association est caractérisée par son nom et ses contraintes de cardinalité avec les valeurs (nombre minimum-maximum de liens pour chaque objet) *1,1* ou *1,n* ou *0,1* ou *0,n* ou *m,n* (*n* et *m* signifiant cardinalités quelconques avec $n \geq m > 1$).

Nous avons choisi de garder la représentation graphique d'une association de la méthode MERISE [TAR 83] car nous considérons que c'est la plus lisible pour la description des contraintes de cardinalité.

Exemple :

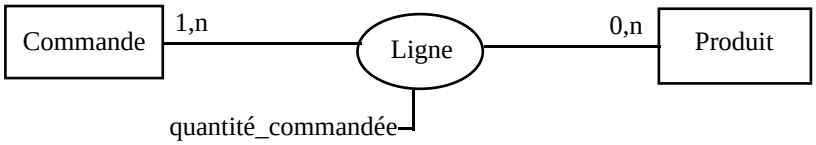


Figure 3.2. *Un exemple d'association*

Dans cet exemple, Ligne est une association entre les entités Commande et Produit. Les contraintes de cardinalité expriment qu'une commande contient au moins un produit et un produit peut être ou ne pas être commandé. Pour chaque produit commandé, il est nécessaire de préciser la quantité_commandée.

II.2.2. Rôle

Un rôle est une extrémité de l'association. Une association détermine toujours deux rôles. Si on veut utiliser un rôle, il faut lui donner un nom. Le nom de rôle est un nom qui identifie de façon unique une extrémité de l'association. Un rôle fournit pour chaque objet le ou les objets qui lui sont associés.

Exemple:

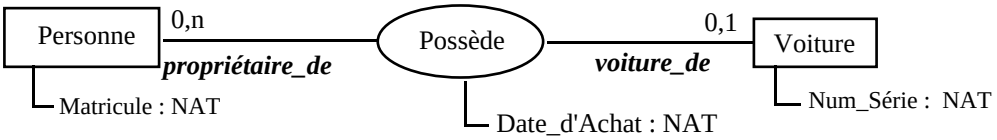


Figure 3.3. *Un exemple de rôles dans une association*

Dans cet exemple, Personne joue le rôle ***propriétaire_de*** vis-à-vis de Voiture, qui joue le rôle ***voiture_de*** vis-à-vis de Personne. Pour chaque voiture, ***propriétaire_de*** détermine la personne qui lui est associée. Pour chaque personne, ***voiture_de*** détermine la ou les voitures qui lui appartiennent (exemple : ***propriétaire_de*** pour la voiture de numéro de série "04105" donne la personne de matricule "15315").

II.2.3. Association fixe

Une association entre les entités *e* et *f* est **fixe pour e** si l'ensemble des liens concernant chaque objet de *e* est créé en même temps que l'objet et ne peut être modifié ensuite (donc un lien ne peut être détruit qu'à la disparition de l'objet).

Exemple :

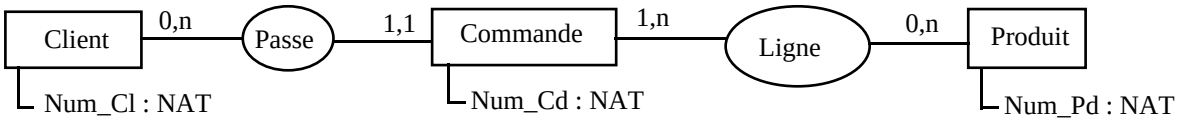


Figure 3.4. Passe : *Association fixe pour l'entité Commande*

Passe est une association fixe pour Commande. En effet, une Commande est définitivement liée au même Client, on ne peut pas changer de client. Par contre, un client peut passer de nouvelles commandes : passe n'est donc pas fixe pour Client. En revanche, Ligne_Commande n'est fixe ni pour Commande ni pour Produit : on peut toujours rajouter une nouvelle ligne de commande pour une commande (choix de conception).

Ce concept est équivalent aux concepts d'"association définitive" et "patte verrouillée" de MERISE [NAN 92]. Il est aussi à rapprocher de la notion CODASYL "retention is fixed" [COD 71].

Le concept d'association fixe permet de préciser la dynamique du système : une association fixe n'a pas d'opération propre. Autrement dit, une instance d'association est créée (resp. supprimée) par l'opération qui crée (resp. supprime) une instance de l'entité pour laquelle elle est fixe.

Remarque
Si les rôles sont définis pour une association fixe, c'est un rôle qui est fixe.

Dans l'exemple précédent, nous pouvons définir pour l'association Passe deux rôles :

- ***a_commandé*** détermine pour chaque client la ou les commandes qui lui appartiennent
- ***a_pour_client*** détermine pour chaque commande le client qui lui est associé.

Dans ce cas, c'est le rôle ***a_pour_client*** qui est fixe.

II.2.4. Qualification

Dans le paragraphe II.1 les clés d'entités sont limitées à des sous-ensembles d'attributs. Elles peuvent être étendues en considérant le concept OMT de qualification : une instance d'entité peut être déterminée de manière unique par une paire formée d'un sous-ensemble de valeurs de certains de ses attributs et d'une autre instance à laquelle elle est liée.

Exemple :

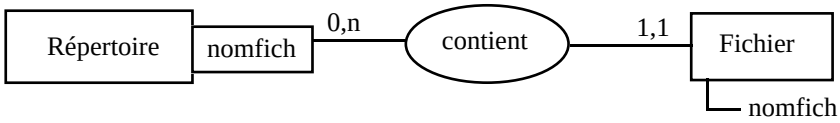


Figure 3.5. *Un exemple de qualification*

nomfich est une qualification dans l'association contient : dans un répertoire, chaque fichier a un nom différent. Autrement dit, un répertoire et un nom de fichier identifient un fichier.

Ce concept est à rapprocher du concept d'entité faible dans [CHE 76]. En fait, dans l'exemple précédent, avec la cardinalité (1,1) entre Fichier et Contient, nous retrouvons exactement le concept d'entité faible.

II.2.5. Entité associative

Dans le modèle que nous retenons, d'une part, on ne peut pas construire une association à partir d'autres associations, d'autre part, nous nous sommes limitée à des associations

binaires. Cela permet de formaliser une association en utilisant toute la puissance des opérateurs relationnels binaires. Il arrive, bien sûr, qu'on ait à considérer des associations d'associations ou des associations n-aires. Dans ce cas, nous utilisons des entités ayant des propriétés particulières, d'où leur nom d'entités associatives.

Exemple :
Soit une association Ligne_comm (ligne de commande) entre Commande et Produit. Si nous voulons créer une association entre Ligne_comm et Ligne_fact (ligne de facture), nous devons transformer Ligne_comm en entité. Cependant, il s'agit d'une entité particulière, dont l'existence -et les instances- sont dépendantes des entités associées Commande et Produit.

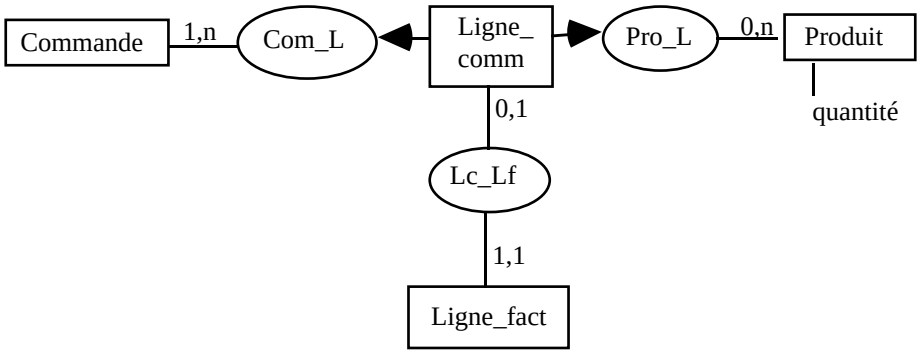


Figure 3.6. *Exemple d'entité associative (Ligne_comm)*

Chaque cardinalité de l'entité associative vers les entités associées vaut (1,1) et chaque tuple d'instances des entités associées détermine au maximum une instance de l'entité associative. En outre les associations avec les entités associées sont fixes pour l'entité associative. Graphiquement, une entité associative est représentée par un rectangle et une flèche pour chaque entité dont elle dépend.

Dans la figure 3.6, l'entité associative Ligne_comm est déterminée par les entités Commande et Produit. C'est-à-dire qu'un couple (c, p) - avec c instance de Commande et p instance de Produit- détermine au maximum une instance de Ligne_comm.

Nous retrouvons ce concept dans la méthode OMT sous le nom de "classe association".

Remarque
Une extension possible au cas précédent est d'admettre des attributs propres dans la détermination de l'entité associative, comme l'attribut date_emprunt dans la figure ci-dessous.

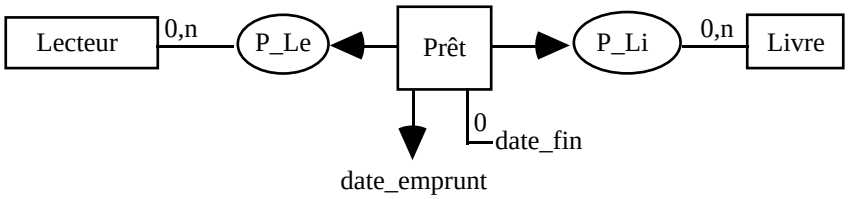


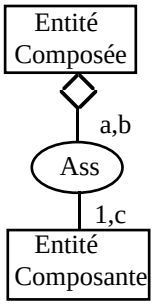
Figure 3.7. *Entité associative avec attribut propre dans les valeurs déterminantes*

Ici (l, o, d) - avec l instance de Lecteur, o instance de Livre et d valeur de date_emprunt détermine une seule instance de Prêt.

II.2.6. Agrégation

Il n'y a pas de concensus dans les différentes méthodes pour le concept d'agrégation. Nous considérons ici le concept d'agrégation comme une association particulière dans laquelle les objets d'une entité sont les composants d'objets d'une autre entité. Il permet de modéliser des objets complexes. Ce concept exprime la relation "composé-composant" ou "partie-de". Dans l'exemple ci-dessous, Système_Sanitaire est un composant de Maison (c'est-à-dire une partie de Maison) et une Maison est composé entre autre d'un Système_Sanitaire.

Nous avons le schéma général suivant :



avec $a \in \{0,1\}$
$b \in \{1,n\}$
$c \in \{1,n\}$

Un lien d'agrégation peut être soit exclusif, soit partagé. Si le lien est exclusif, un objet composant ne peut faire partie que d'un seul objet composé (c=1). En revanche, si le lien est partagé, un objet composant peut faire partie de plusieurs objets composés (c=n).

Exemple :

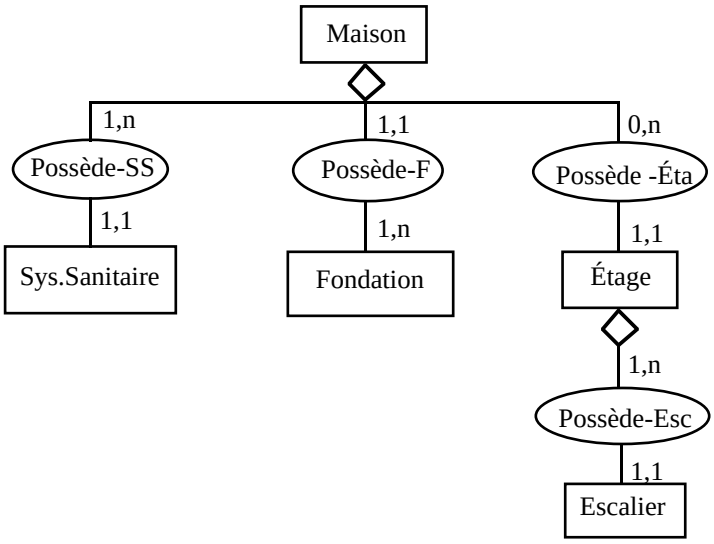


Figure 3.8. *Un exemple d'agrégation.*

Possède_SS, Possède_Éta, Possède_Esc sont des liens d'agrégation exclusifs car un Système_Sanitaire et un Étage font partie d'une et d'une seule Maison, un Escalier fait partie d'un et d'un seul Étage. En revanche, le lien Possède_F est partagé car une Fondation peut être commune à plusieurs Maisons.

Remarque
Une agrégation n'est pas nécessairement une association fixe. En effet, un objet composant n'est pas obligatoirement toujours lié au même objet composé.

II.2.7. Contraintes d'intégrité

II.2.7.1. Contrainte d'intégrité fonctionnelle (CIF)

Une contrainte d'intégrité fonctionnelle (MERISE [NAN 92]) indique qu'une entité e est totalement déterminée par la connaissance d'autres entités (e1, ..., en) qui sont liées à e par les associations r1, ..., rn. Dans l'exemple ci-dessous, e correspond à l'entité Professeur, e1 à l'entité Classe, e2 à l'entité Matière, r1 à l'association occupe_par et r2 à l'association enseigne_par.

Le concept d'entité associative est un cas particulier de la contrainte d'intégrité fonctionnelle car, d'une part, une instance d'une entité associative est déterminé par la connaissance des entités associées et d'autre part, les associations avec ces entités sont fixes.

Graphiquement, la contrainte d'intégrité fonctionnelle est présentée comme suit :

- un cercle dans lequel est indiqué CIF,
- une flèche qui indique l'entité e,
- les traits pleins qui indiquent les entités e1, ..., en,
- les traits en pointillé qui indiquent les associations r1, ..., rn.

Exemple :

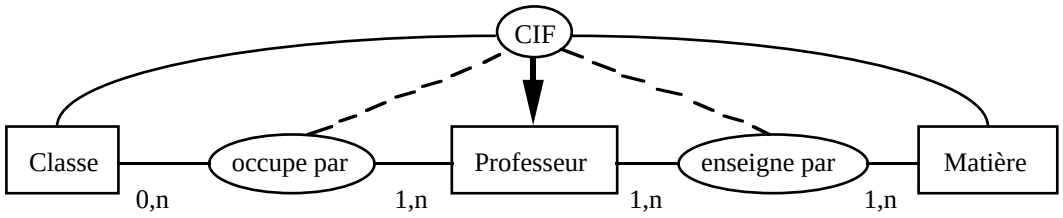


Figure 3.9. *Un exemple de CIF*

Cette contrainte exprime qu'une classe et une matière identifient un professeur.

II.2.7.2. Contraintes entre ensembles d'instances participant à deux associations

Ces contraintes peuvent être définies pour des associations ayant au moins une entité commune.

a. Contrainte d'inclusion

Soit deux associations r1 et r2 et une entité e commune à r1 et r2. Cette contrainte exprime que toute instance de e qui participe à r1 doit participer à r2. Elle est représentée graphiquement par une flèche dans le sens de l'inclusion comme le montre la figure suivante.

Dans l'exemple ci-dessous, e correspond à l'entité Commande, r1 à l'association Com_Fac et r2 à l'association Com_Li.

Exemple :

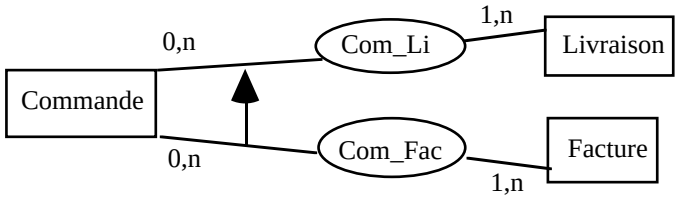


Figure 3.10. *Un exemple de contrainte d'inclusion entre Com_Li et Com_Fac avec l'entité Commande*

La contrainte d'inclusion de cette figure exprime que toute instance de Commande participant à l'association Com_Fac doit aussi participer à l'association Com_Li.

b. Contrainte d'égalité

Soit deux associations r1 et r2 et une entité e commune à r1 et r2. Cette contrainte exprime que toute instance de e qui participe à r1 doit participer à r2 et réciproquement. Elle est représentée graphiquement par le symbole = encerclé comme le montre la figure suivante.

Dans l'exemple ci-dessous, e correspond à l'entité Commande, r1 à l'association Com_Fac et r2 à l'association Com_Li.

Exemple :

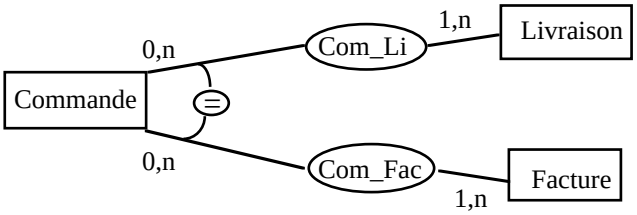


Figure 3.11. *Un exemple de contrainte d'égalité entre Com_Li et Com_Fac avec l'entité Commande*

La contrainte d'égalité de cette figure exprime que toute instance de Commande participant à l'association Com_Li, doit aussi participer à l'association Com_Fac, et réciproquement.

Remarque

La contrainte d'égalité est équivalente à une contrainte d'inclusion dans les deux sens

c. Contrainte d'exclusion

Soit n associations r1, r2, ..., rn et une entité e commune à r1, r2, ..., rn. Cette contrainte exprime qu'aucune instance de e ne peut participer à la fois à r1, r2, ...et à rn. Elle est représentée graphiquement par le symbole + encerclé comme le montre la figure suivante.

Dans l'exemple ci-dessous, e correspond à l'entité Cassette, r1 à l'association Contient_Film et r2 à l'association Contient_Épisode.

Exemple :

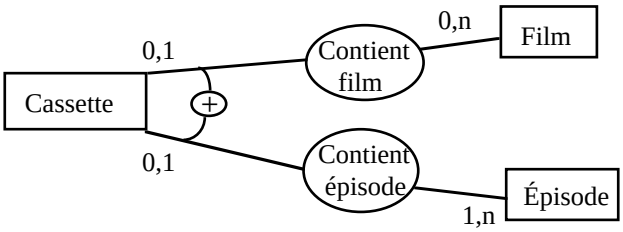


Figure 3.12. *Un exemple de contrainte d'exclusion entre Contient Film et*

Contient Épisode avec l'entité Cassette

La figure précédente exprime que chaque cassette peut contenir soit un Film, soit un Épisode mais pas les deux.

d. Contrainte de totalité

Soit n associations r1, r2, ..., rn et une entité e commune à r1, r2, ..., rn. Cette contrainte exprime que toute instance de e doit participer à au moins une association. Elle est représentée graphiquement par un T encerclé comme le montre la figure suivante.

Dans l'exemple ci-dessous, e correspond à l'entité Auteur, r1 à l'association Écrit_L et r2 à l'association Écrit_R.

Exemple :

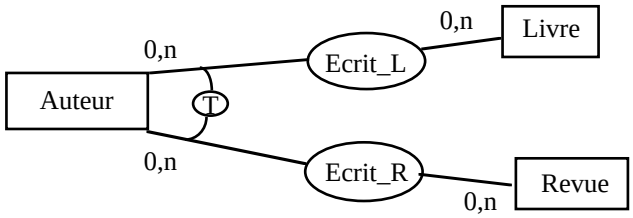


Figure 3.13. *Un exemple de contrainte de totalité entre Ecrit_L et Ecrit_R avec l'entité Auteur*

La contrainte de totalité de cette figure exprime que toute instance de Auteur doit être membre d'au moins une association, c'est-à-dire qu'un auteur doit avoir écrit au moins un Livre ou une Revue.

II.2.7.3. Contraintes entre ensembles de liens

Ces contraintes peuvent être définies pour des associations reliant les mêmes entités.

Dans cette partie, nous donnons un exemple de contrainte d'inclusion. Remarquons que les contraintes d'exclusion, de totalité, d'égalité sont définies de manière analogue.

Une contrainte d'inclusion entre deux associations r1 et r2 qui relient deux entités e et f exprime que tout lien de r1 est un lien de r2. Elle est représentée graphiquement par une flèche de r1 vers r2 comme le montre la figure suivante.

Dans l'exemple ci-dessous, e correspond à l'entité Personne, f à l'entité Comité, r1 à l'association Président_du et r2 à l'association Membre_du.

Exemple :

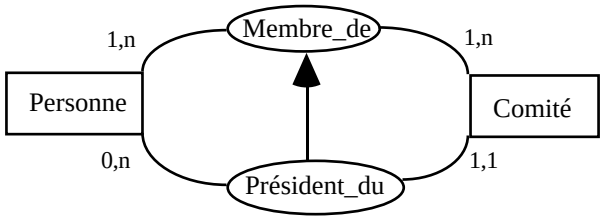


Figure 3.14. *Contrainte d'inclusion entre ensembles de liens*

Dans cet exemple le président d'un comité doit être membre de ce comité. Autrement dit, tout lien Président_du est un lien Membre_de.

Remarque

Notons que cette contrainte d'inclusion entre ensemble de liens implique évidemment la contrainte d'inclusion entre ensembles d'instances définie précédemment.

II.2.7.4. Contrainte d'homomorphisme

Dès qu'un système d'information utilise des méta-données (données décrivant d'autres données) les associations entre méta-données reflétent le plus souvent les associations entre données.

Exemple :

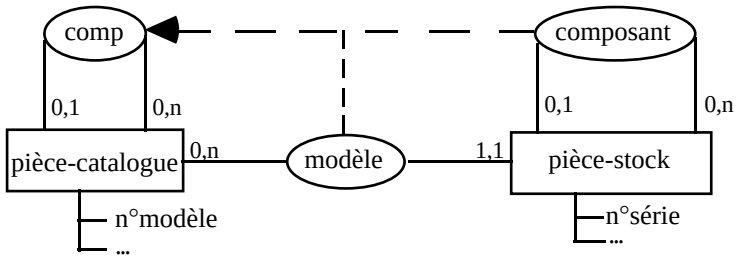


Figure 3.15. *Homomorphisme de composant vers comp, via modèle*

La figure précédente est un exemple d'une telle contrainte entre deux associations : deux pièces en stock sont liées par l'association composant ssi leurs modèles qui leur sont liés par l'association modèle sont liés par l'association comp.

II.3. Héritage et concepts liés

II.3.1. Lien d'héritage

Nous considérons ici le concept de lien d'héritage tel que défini dans [SMI 77]. Il est utilisé pour spécialiser une entité (la "super-entité") en une ou plusieurs autres entités (les "sous-entités"). Toute instance d'une entité est instance de ses super-entités.

Exemple :

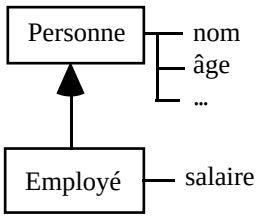


Figure 3.16. *Exemple de lien d'héritage*

Dans cet exemple, Employé est une sous-entité de l'entité Personne.

Par ce lien, une sous-entité hérite des propriétés (attributs et opérations) de sa super-entité. Une sous-entité peut possèder des propriétés supplémentaires qui lui sont propres. Tout objet d'une sous-entité est objet de sa super-entité mais avec éventuellement des propriétés supplémentaires.

II.3.2. Contrainte de couverture entre sous-entités

La couverture est une contrainte définie entre une super-entité et ses sous-entités. Elle impose à chaque instance de la super-entité d'appartenir à l'une de ses sous-entités. La super-entité est alors appelée généralisation ou entité abstraite.

Exemple :

Dans l'exemple suivant, Lecteur est une entité abstraite : un lecteur est soit un lecteur Adulte, soit un lecteur Enfant.

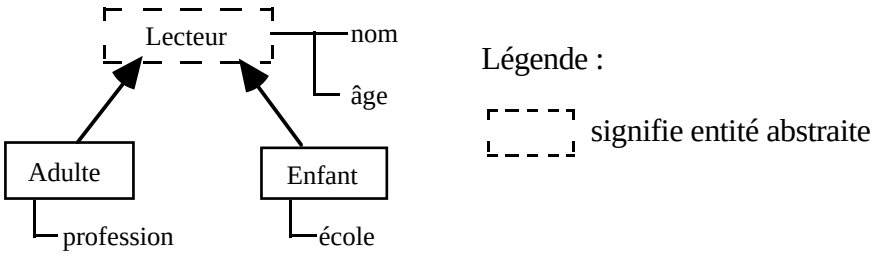


Figure 3.18. *Exemple de contrainte de couverture*

II.3.3. Contrainte de disjonction entre sous-entités

La disjonction entre sous-entités d'une super-entité traduit le fait que les sous-entités en question sont disjointes.

Exemple :
Dans l'exemple suivant, l'ensemble des ingénieurs et des secrétaires sont disjoints.

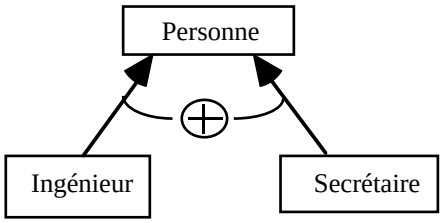


Figure 3.19. *Exemple de contrainte de disjonction*

II.4. Opération

Les opérations font partie de la dynamique du système. Cependant, le modèle d'objets fournit déjà le nom des opérations associées à chaque entité. Une description plus détaillée des opérations est donnée dans le modèle dynamique (cf. partie III).

Ce concept est décrit en OMT, mais le contexte dans lequel il est utilisé n'a pas été clairement défini par les auteurs. Dans cette partie, nous allons préciser son utilisation et ce qui nous a amené à le développer.

Dans la méthode OMT, une opération est associée à une entité. Une opération est définie comme étant une fonction ou une transformation qui peut être appliquée aux objets d'une entité. De ce fait, une association ne possède pas d'opération propre. Si on a besoin d'associer des opérations à une association, il faut alors définir une "classe association".

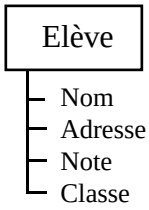
Nous trouvons que cette contrainte ne se justifie pas vraiment. C'est pourquoi, nous proposons de définir une opération soit sur une entité, soit sur une association. Les noms des opérations sont distincts dans une entité ou une association.

II.4.1. Opération sur une entité

Une entité peut comporter deux types d'opérations : opération d'instance et opération d'entité.

Une opération d'instance possède toujours comme paramètre un objet cible sur lequel va s'appliquer cette opération. Elle peut avoir des paramètres, en plus de son objet cible. Les paramètres sont écrits entre parenthèses après le nom de l'opération et sont séparés par des virgules.

Exemple : Étant donné l'entité Élève comme ci-dessous :



Change_Adresse, Supprime_Élève sont des opérations d'instance de l'entité Élève. L'opération Change_Adresse(e, n_adresse) possède comme paramètres : l'élève e et sa nouvelle adresse n_adresse.

En revanche, une opération d'entité s'applique à un ou plusieurs objets non mentionnés directement en tant que paramètres de l'opération.

Exemple : Surclasser est une opération d'entité car, avant son exécution, il faut d'abord sélectionner un ensemble d'élèves à surclasser.

Tous les objets d'une entité partagent les mêmes opérations.

II.4.2. Opération sur une association

Nous pouvons définir également des opérations pour les associations. Par exemple, les opérations de modification de valeur d'attributs (dans le cas où l'association possède des attributs), d'ajout ou de suppression d'un lien d'association ...

Exemple :
Reprenons le diagramme de la figure 3.2, nous pouvons définir, pour l'association Ligne, trois opérations :

- Change_Quantité : changera la quantité_commandée pour un produit donné dans une commande.
- Ajout_Ligne : ajoutera un nouveau produit dans une commande.
- Supprime_Ligne : enlèvera une ligne d'une commande.

Il est préférable que la liste des opérations soit donnée dans la description détaillée de chaque entité ou chaque association afin de ne pas alourdir le diagramme d'objets.

III. Le modèle Dynamique

Face à la très grande variété des modèles dynamiques proposés par les méthodes semi-formelles, une démarche d'extraction de concepts communs, comme nous l'avons fait pour ceux du modèle d'objets, serait difficilement réalisable. Dans notre travail de recherche, nous avons opté pour le modèle dynamique de la méthode OMT [RUM 91] car c'est, d'une part, une méthode très utilisée et largement répandue dans l'industrie, et d'autre part, un modèle que l'on retrouve dans plusieurs méthodes (UML [UML 97], OOA [SHL 92], ...).

Le modèle dynamique en OMT est basé sur les diagrammes états/transitions dérivés des notations de [HAR 87]. Un diagramme états/transitions est un réseau d'états et d'événements dans lequel les états représentent les valeurs des objets et les événements représentent les stimuli externes. Il est défini pour décrire le comportement dynamique de chaque objet.

Nous allons donner les définitions des concepts du diagramme états/transitions telles qu'elles apparaissent dans [RUM 91]. Cependant, pour certaines d'entre elles, nous avons été amenés à les étendre afin de préciser leur utilisation.

III.1. Concepts de base

III.1.1. Événement, État, Transition, Diagramme états/transitions, Condition

III.1.1.1. Définitions de [RUM 91]

Un événement est un moyen de transmission d'informations. Il est écrit en italique. Les événements peuvent avoir des paramètres qui sont représentés entre parenthèses après le nom de l'événement.

Il est important de faire la distinction entre type d'événement et instance d'événement. Un type d'événement est la description de l'ensemble des instances d'événement ayant le même nom, les mêmes paramètres. Le terme événement est souvent utilisé dans la littérature avec ambiguïté. Parfois, un événement fait référence à une instance d'événement, tandis qu'à d'autres moments, il désigne le type. Dans la suite, le terme événement désignera type d'événement.

Exemple :
Dans la figure 3.20, nous décrivons les événements de l'entité Personne dont la description des attributs a été donnée dans la figure 3.1 : Naissance, Mariage, Divorce, Remariage . Ces événements marquent les changements de situation d'une personne donnée. L'événement Naissance possède trois paramètres : le numéro de matricule, le nom et la date de naissance du bébé. Les événements Mariage, Divorce, Remariage sont sans paramètres. En effet, dans l'entité Personne, nous ne nous intéressons pas au conjoint.

Un état est une abstraction des valeurs des attributs et des liens d'un objet. Un état détermine la réponse de l'objet aux événements associés. Les noms des états dans les différents diagrammes états/transitions sont distincts.

Exemple :
Dans la figure 3.20, Ange, Célibataire, Mariée, Divorcée sont des états.

Une transition permet à un objet de passer d'un état (source) dans un autre état (arrivée) à l'apparition d'une instance d'événement.

Exemple :
Dans la figure 3.20, la flèche de l'état Ange à l'état Célibataire est une transition.

Un diagramme états/transitions (noté diagramme E/T) relie des événements à des états et concerne une seule entité. Toutes les transitions quittant un état donné doivent correspondre à différents événements. En effet, il est préférable d'avoir un système déterministe.

Une transition peut-être accompagnée d'une condition de déclenchement, elle est alors appelée transition gardée. Une transition gardée est franchie seulement si la condition associée

est satisfaite. Une telle condition est une fonction booléenne sur les valeurs des attributs de l'objet. Elle est représentée entre crochets à la suite du nom de l'événement.

Exemple :
Dans la figure 3.20, la condition [Age > 18] décrit qu'une personne célibataire peut se marier si elle a plus de 18 ans.

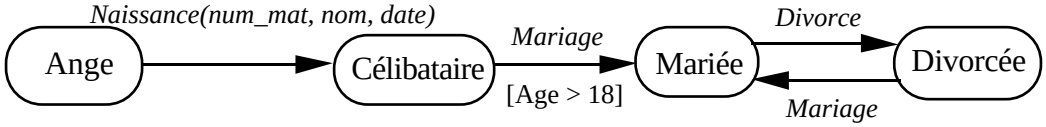


Figure 3.20. *Un exemple de diagramme E/T de l'entité Personne*

III.1.1.2. Extensions

Il y a en fait trois types d'événement :

(i) Événement d'entrée : regroupe tous les événements venant de l'extérieur du système. Un événement d'entrée peut être soit :
- un événement aléatoire qui est une demande de l'utilisateur du système.
- un événement temporel qui décrit une situation faisant référence au temps. Les références au temps peuvent être exprimées en temps absolu (ex, le 19/10/95) ou en temps périodique (ex, tous les mardis).

(ii) Événement de sortie : regroupe les événements sortant du système. Ces événements sont utilisés pour transmettre vers l'extérieur un message, un résultat.

Ces deux types d'événement sont des événements externes.

(iii) Événement interne : est utilisé pour échanger les informations entre les diagrammes E/T (cf. III.2.3).

Dans le diagramme d'objets, toutes les caractéristiques d'un objet ne sont pas obligatoirement décrites par un attribut ou un lien , quand leur valeur n'a pas besoin d'être stockée.

Exemple :
Concernant les voitures, on veut stocker les informations relatives au Numéro, à la Couleur, au Modèle ... Mais il est possible que l'on n'ait pas sauvegardé le fait qu'une voiture soit en état

Marche ou Arrêt. Au niveau du diagramme d'objets, il n'y a alors aucun attribut qui représente les états du diagramme E/T de l'entité Voiture.

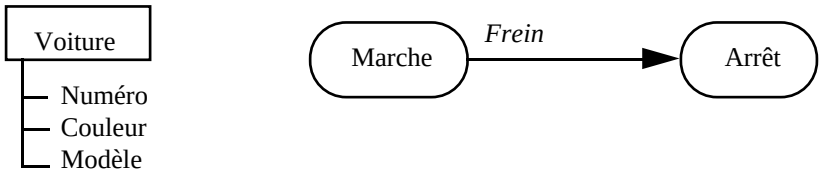


Figure 3.21. *Un exemple d'états n'étant pas une abstraction des valeurs des attributs et des liens d'un objet*

C'est pourquoi un état ne peut pas toujours être une abstraction des valeurs des attributs et des liens d'un objet. La définition précédente devient donc : un état est une abstraction de certaines caractéristiques d'un objet.

Puisque tous les objets d'une entité ont la même structure et le même comportement, ils partagent tous le même diagramme E/T. Mais, de la même façon que chaque objet a ses propres valeurs d'attribut, chaque objet a également son propre état, résultat de la séquence des instances d'événements qu'il a reçues. De plus, par définition, un état déterminant la réponse de l'objet aux événements d'entrée, nous déduisons que chaque événement possède implicitement comme paramètre l'objet sur lequel porte la transition. Cet objet est appelé l'objet courant du diagramme E/T et dans la suite noté o par convention.

La définition de OMT spécifie qu'un diagramme E/T concerne une seule entité. Cependant, une entité est, la plupart du temps, reliée à d'autres entités par des liens d'association. C'est pourquoi nous permettons à un événement de prendre en compte comme paramètres les liens d'association directs et les objets des entités liées par ces liens d'association. Cette option est indispensable pour prendre en compte les conditions de déclenchement d'un événement qui dépendent des valeurs de ces liens et de ces objets.

Notons qu'avec l'extension précédente, des conditions éventuelles concernant les liens d'association et les entités voisines peuvent être précisées. Dans ce cas, la définition précédente est étendue comme suit : une condition est une fonction booléenne sur les caractéristiques d'un objet ou (et) celles des liens d'association directs ou (et) celles des objets des entités liées par ces liens d'association.

Exemple :

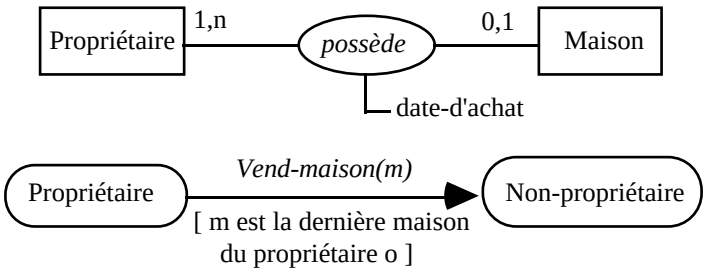


Figure 3.22. *Un exemple de condition avec extension dans le diagramme E/T associé à l'entité* Personne

Dans cet exemple, m désigne une instance de l'entité Maison.

Par définition, toutes les transitions quittant un état donné correspondent à différents type d'événements. Cependant, une transition est franchie quand survient une instance d'événement mais seulement si la condition de déclenchement éventuelle est vraie. Il est donc possible d'avoir plusieurs transitions (quittant un état) correspondant au même type d'événement et associée à différentes conditions. La définition précédente entraîne donc : toutes les transitions quittant un état donné et correspondant au même type d'événement doivent avoir des conditions disjointes. Il s'agit toujours d'un système déterminisme.

Exemple :

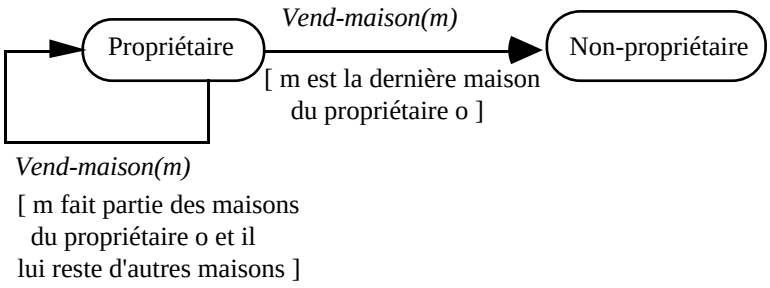


Figure 3.23. *Un exemple de diagramme E/T avec deux transitions associées au même type d'événement et portant deux conditions différentes*

III.1.2. Opération

Nous conservons les définitions des différents types d'opération proposés par [RUM 91]. Il existe deux types d'opération : activité et action.

III.1.2.1. Activité

Une activité est une opération qui nécessite un certain temps d'exécution, qui est associée à un état et qui ne change pas l'état où elle est définie. Elle est décrite après le mot clé *faire* (a_3 de la figure 3.25).

Exemple :
L'affichage d'une image sur un écran.

III.1.2.2. Action

Une action est une opération instantanée associée à un événement. Elle est exécutée à l'apparition d'une instance de cet événement si l'éventuelle condition de déclenchement est vraie. La notation ev_i/a_1 (figure 3.25) indique que a_1 est une action associée à l'événement ev_i.
Les paramètres d'une action sont ceux de l'événement associé.

Convention d'écriture dans le diagramme E/T :
Les événements sont notés en italiques, les actions en caractères simples. Si le nom de l'événement et celui de l'action associée sont les mêmes alors ils sont confondus et écrits en gras.

Exemple :

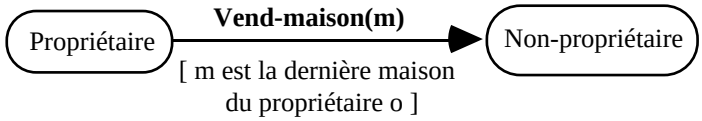


Figure 3.24. *Un exemple d'action et d'événement confondus*

Vend-maison représente à la fois l'événement qui déclenche la transition et l'action associée (qui supprime le lien entre la maison m et son propriétaire).

Une action peut-être externe ou interne.

Une action externe est associée à une transition.

Dans le cas où toutes les transitions vers un état exécutent la même action, il est conseillé d'attacher l'action à l'entrée de l'état dans le but d'alléger la description du diagramme états/transitions. Une action peut donc être associée à une entrée ou une sortie d'un état. Une action associée à une entrée d'état (resp. sortie) est exécutée à chaque fois qu'on rentre dans l'état (resp. on sort).

Exemple : Dans la figure 3.25 :

- a_2 est une action d'entrée : c'est-à-dire, à l'arrivée de l'événement ev_i, on exécute a_1 puis a_2.
- a_5 est une action de sortie : c'est-à-dire, à l'arrivée de l'événement ev_j, on exécute a_5 puis a_6.

Une action interne (a_4 dans la figure 3.25) est associée à un état. Nous avons une action interne lorsqu'un événement provoque l'exécution de cette action sans changer d'état (à l'arrivée de l'événement ev_k, on exécute a_4). Notons la différence entre une action interne et une action sur une transition réflexive : la transition réflexive (ev_l/ a_7) provoque l'exécution des actions d'entrée et de sortie pour l'état.

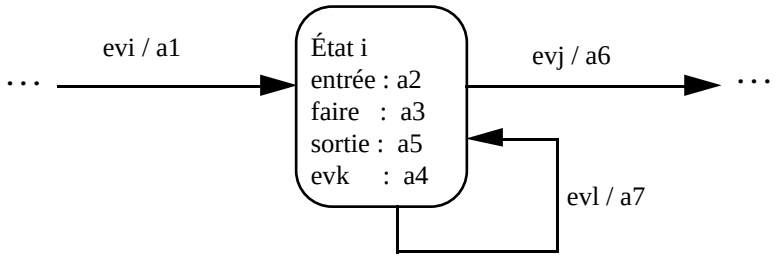


Figure 3.25. *Résumé des différents types d'opération*

III.1.3. Liaison entre le modèle d'objets et le modèle dynamique

Dans la méthode OMT, nous trouvons une description des liens entre les différents modèles (modèle d'objets, modèle dynamique, modèle fonctionnel). Ainsi, par exemple, pour leur auteurs, les événements d'un diagramme états/transitions deviennent des opérations attachées aux objets dans le modèle d'objets. Nous considérons que cette définition est imprécise simplement par le fait qu'un événement n'est que le déclencheur des actions à exécuter, il ne peut donc pas possèder à l'intérieur de lui-même d'opération. Cette définition est alors valable quand on confond événement et action. Par conséquent, le lien, s'il y a lieu, doit être entre les actions associées à un événement et les opérations d'une entité. Dans ce cas, l'action associée à un événement d'un diagramme états/transitions se compose d'opérations attachées aux objets dans le modèle d'objets.

III.2. Concepts évolués

III.2.1. Transition automatique

Une transition automatique est un cas particulier de transition. La transition automatique n'a pas d'événement associé et peut être accompagnée d'une condition.

Si l'état source d'une transition automatique contient une activité, la transition automatique est franchie quand l'activité est achevée. Sinon, elle débute aussitôt que l'on entre dans l'état (mais les actions d'entrée et de sortie sont toujours exécutées).

Dans le cas où la transition automatique est accompagnée d'une condition, elle sera franchie quand la condition associée est remplie. S'il y a plusieurs transitions automatiques associées à un état, chaque transition est gardée par une condition, toutes les conditions doivent être disjointes. Il s'agit toujours d'un système déterministe.
Exemple :

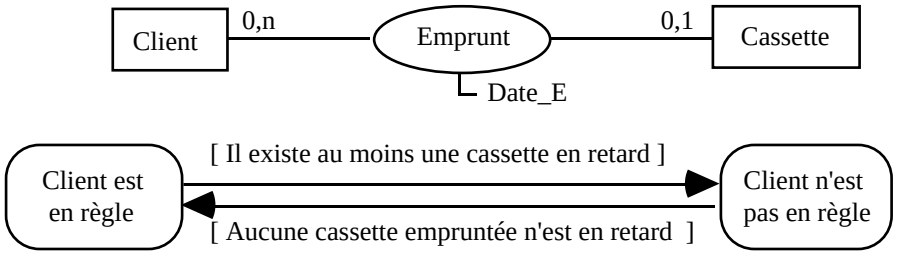


Figure 3.26. *Un exemple de transition automatique avec condition (cas où l'événement Rend_K7 arrive à l'entité Cassette)*

Dans cet exemple, la condition sur la transition de l'état Client est en règle à l'état Client n'est pas en règle exprime qu'un client en état en règle devient un client n'étant pas en règle dès qu'il garde au moins une cassette plus de huit jours. Sur la transition de l'état Client n'est pas en règle à l'état Client est en règle, la condition décrit le fait que toutes les cassettes empruntées par le client respectent le délai de huit jours.

III.2.2. Concurrence d'objets

La "concurrence d'agrégations" est définie dans [RUM 91] pour aider à décrire la dynamique d'un ensemble d'entités liées entre elles par le concept d'agrégation. L'état de l'agrégat correspond aux états combinés des diagrammes E/T des composants. L'état agrégé est constitué d'un état de chacun des diagrammes E/T des entités de l'agrégation. Dans le cas où les états composants interagissent, c'est-à-dire que les transitions gardées pour un objet d'un diagramme E/T dépendent d'un autre objet dans un état donné d'un autre diagramme E/T, OMT

utilise "La concurrence d'agrégations" pour exprimer l'interaction entre les diagrammes E/T, tout en préservant leur modularité.

Exemple :
Reprenons l'exemple présenté dans [RUM 91]. Supposons que l'on décrive une voiture comme agrégation d'objets composants : Contact, Boîte-de-transmission, Accélérateur et Frein.

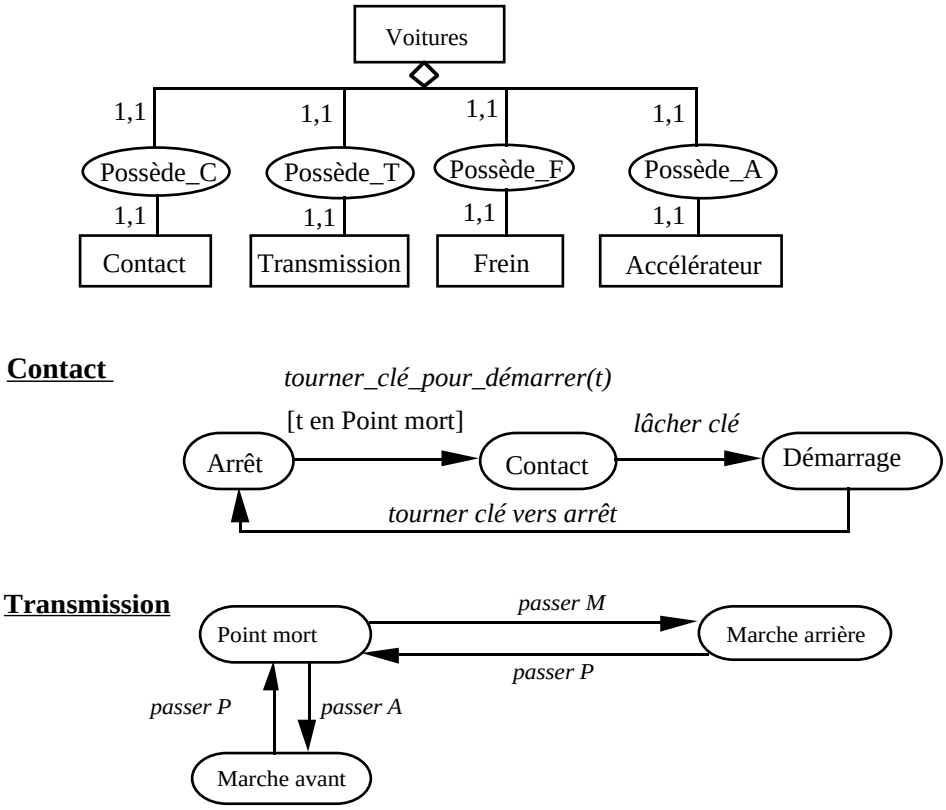


Figure 3.27. *Une agrégation et ses diagrammes E/T concurrents*

N.B. Tous les états du système ayant des noms différents, la condition t en Point mort implique que t est une transmission.

Les diagrammes E/T des composants ne sont pas indépendants à cause de la condition : la voiture ne démarrera pas tant que la transmission ne sera pas au point mort. Cela est mis en évidence par la condition [t en Point mort] sur la transition gardée de l'état Arrêt à Contact.

Remarques

- Nous pensons que le concept de "Concurrence d'agrégations" de la méthode OMT pourrait être appliqué à d'autres types de relation. En effet, si les relations entre objets sont des

associations quelconques, les diagrammes E/T des objets liées peuvent aussi interagir entre eux. On utilise alors le même principe. Dans ce cas là, nous préférons le terme "concurrence d'objets" que "concurrence d'agrégations".

Exemple :
Soient le diagramme d'objets et ses diagrammes états/transitions correspondants suivants :

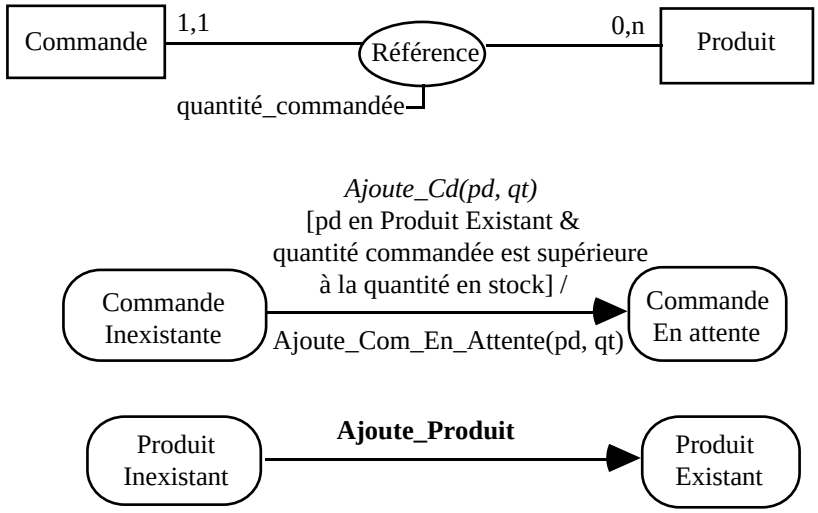


Figure 3.28. [pd en Produit Existant] : *un exemple de transition gardée en utilisant la concurrence d'objets*

La transition gardée [pd en Produit Existant] exprime que le produit pd commandé doit être un produit existant.

- Il y a en fait deux types de conditions sur les transitions gardées. Le premier regroupe les conditions qui portent sur les caractéristiques (attributs, liens d'association) de l'objet courant (c.f. I.1.2.). Le deuxième regroupe les conditions qui vérifient qu'un autre objet est dans un état donné de son diagramme E/T. Cet objet doit être un paramètre de l'événement associé à la transition.

III.2.3. Échange d'événements

Les objets d'un système interagissent par "échange d'événements". Un objet peut exécuter une action d'envoi d'un événement à un autre objet lors du franchissement d'une transition. Graphiquement, l'envoi d'un événement est représenté par une flèche en pointillés pointant sur l'entité de l'objet récepteur et étiquetée par cet objet récepteur et le nom de l'événement avec ses paramètres éventuels. Cet événement est appelé événement interne.

Exemple :
Reprenons le diagramme d'objets de la figure 3.28.

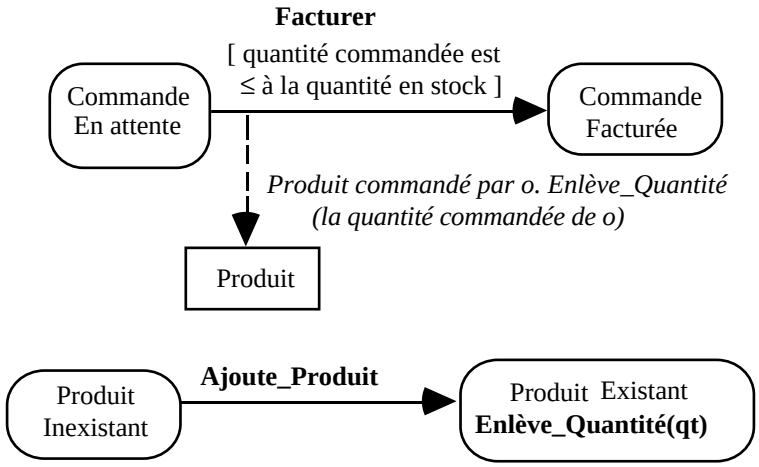


Figure 3.29. *Produit commandé par o.Enlève_Quantité(la quantité commandée de o) : Un exemple d'envoi d'événement à un objet d'une autre entité*

Dans cet exemple, quand la quantité commandée d'une commande o, en état En attente, est inférieure ou égale à la quantité en stock du produit commandé, on facture cette commande puis on envoie l'événement *Enlève_Quantité* au produit commandé par o de l'entité Produit afin de diminuer sa quantité en stock.

Remarques

Comme la concurrence d'objets, "Échange d'événements" fournit un autre moyen de communiquer entre les diagrammes E/T. Mais cette fois-ci, les informations sont transférées grâce aux paramètres des événements.

III.2.4. Diagramme associé à l'administrateur

Dans les paragraphes précédents, nous avons présenté les concepts des diagrammes E/T. Ces diagrammes décrivent dans quel contexte, un objet (dit courant) est modifié par une opération donnée. L'opération fait passer l'objet courant d'un état dans un autre. Cela n'est possible que si l'objet courant est connu avant la réalisation de l'opération c'est-à-dire qu'il fait partie des paramètres de l'opération. Une telle opération est communément appelée opération d'instance.

Lorsqu'on ne connaît pas l'objet sur lequel va s'appliquer une opération, cette opération ne peut pas être une opération d'instance. De manière naturelle, une telle opération doit d'abord identifier l'objet (ou les objets) à traiter avant sa réalisation. Pour décrire ce type d'opération, nous utilisons les opérations d'entité. Ce concept a été étendu à partir du concept d'opération de classe de la méthode OMT en précisant le cadre d'utilisation. Nous allons présenter maintenant l'extension que nous lui avons apporté.

Une *opération d'entité* s'applique à un ou plusieurs objets non mentionnés directement en tant que paramètres de l'opération. Donc, avant son exécution, il faut d'abord sélectionner ces objets selon un critère. Il existe deux manières :

(i) Non-déterministe : nous avons ce cas lorsque le critère de sélection donne plusieurs solutions. N'importe laquelle de ces solutions convient à l'opération.

Exemple :
Cas d'un livre et ses exemplaires. Un lecteur qui désire emprunter un livre se présente au guichet. Si au moins un exemplaire est disponible, la bibliothécaire va lui en prêter un. Dans ce cas, la sélection de l'exemplaire à prêter est faite de manière non-déterministe parmi les exemplaires disponibles concernant ce livre.

(ii) Déterministe : contrairement au cas précédent où le choix est libre, cette fois-ci, l'objet (ou les objets) traité par l'opération est la seule solution répondant au critère de sélection.

Exemple :
À la fin de l'année, on veut surclasser en C.P. tous les élèves de maternelle deuxième année ayant une moyenne supérieure ou égale à 14. Dans ce cas, la sélection des élèves à surclasser est faite de manière déterministe. Un seul résultat est retourné. Il n'y a donc pas de possibilité de choix des objets à traiter.

Remarquons que les diagrammes E/T ne peuvent pas être utilisés tels quels pour décrire les opérations d'entité. En fait, l'objet traité par l'opération n'est plus l'objet courant du diagramme E/T. La méthode OMT introduit le concept d'opération d'entité mais ne donne aucun moyen pour le décrire dans le cadre de l'expression des diagrammes E/T. Il n'y a donc pas de possibilité de décrire les critères de sélection associés à ces opérations, ce qui arrive pourtant fréquemment. L'expression de ces critères est indispensable afin de spécifier plus précisément le système.

C'est pour cette raison que nous avons jugé utile d'introduire une nouvelle représentation graphique : le diagramme associé à l'administrateur de l'entité. Nous n'avons pas chercher à concevoir un nouveau type de diagrammes pour exprimer ces situations, mais nous essayons plutôt de compléter les diagrammes E/T.

Le terme "Administrateur d'une classe" n'est pas nouveau. Dans [CAS 94], ce terme désigne une instance d'une classe CL-j associée à une classe CL-i. La classe CL-j est créée spécialement pour gérer les caractéristiques collectives de la classe CL-i. Les caractéristiques collectives d'une classe peuvent être les attributs sur l'ensemble des intances (DernierTarif, NombreTarif, ...) ou les servives collectifs (services qui gèrent des attributs collectifs ou services qui administrent les instances d'une classe comme la création, la suppression ...).

Afin de ne pas alourdir le diagramme d'objets, nous ne considérons pas que l'administrateur d'une entité est un objet d'une nouvelle entité mais c'est un composant spécial, de l'entité qu'il administre, sans attributs, ni liens d'association. Par contre, il a une vision globale qui lui permet de gérer l'ensemble de tous les objets de l'entité. Son rôle est de sélectionner l'objet (ou les objets) sur lequel va s'appliquer une opération d'entité selon les critères exprimés. Son diagramme doit pouvoir caractériser le fait qu'il se trouve toujours prêt à recevoir les événements et à effectuer la recherche des objets. Les actions déclenchées par ces événements consistent d'abord à faire un choix de l'objet (ou les objets) puis à exécuter les opérations d'entité.

Dans le diagramme associé à l'administrateur, il y a un seul état qui représente le fait que l'administrateur est toujours prêt à recevoir les événements. Nous pouvons alors représenter chaque événement qui arrive à l'administrateur sur une transition réflexive.

La sélection de l'administrateur fournit soit un objet, soit un ensemble d'objets répondant aux critères souhaités. Dans les deux cas, l'administrateur doit envoyer un événement vers le diagramme E/T de l'objet pour signifier l'action à réaliser et le changement d'état de l'objet (ou les objets) sélectionné. Si un seul objet correspondant au choix de l'administrateur, cette émission est représentée classiquement par le concept "échange d'événements" (flèche en direction du nom de l'entité à envoyer). Dans le cas où un ensemble d'objets est sélectionné, l'administrateur doit émettre un événement pour chaque élément de l'ensemble. Nous pouvons représenter l'ensemble de ces événements par une seule flèche étiquetée par l'expression *pour Ens* vers le nom de l'entité car le même événement doit être envoyé à tous les objets de cet ensemble de l'entité indiquée.

Nous avons choisi d'exprimer les critères de choix par une *clause* portant sur la flèche d'envoi d'événement vers les diagrammes E/T. Cette *clause* se compose d'une expression ou condition qui décrit la façon de récupérer l'objet (ou les objets) traité. Bien entendu, les objets choisis doivent être dans l'état initial de la transition recevant pour que la transformation soit possible.

Nous avons le schéma général suivant :

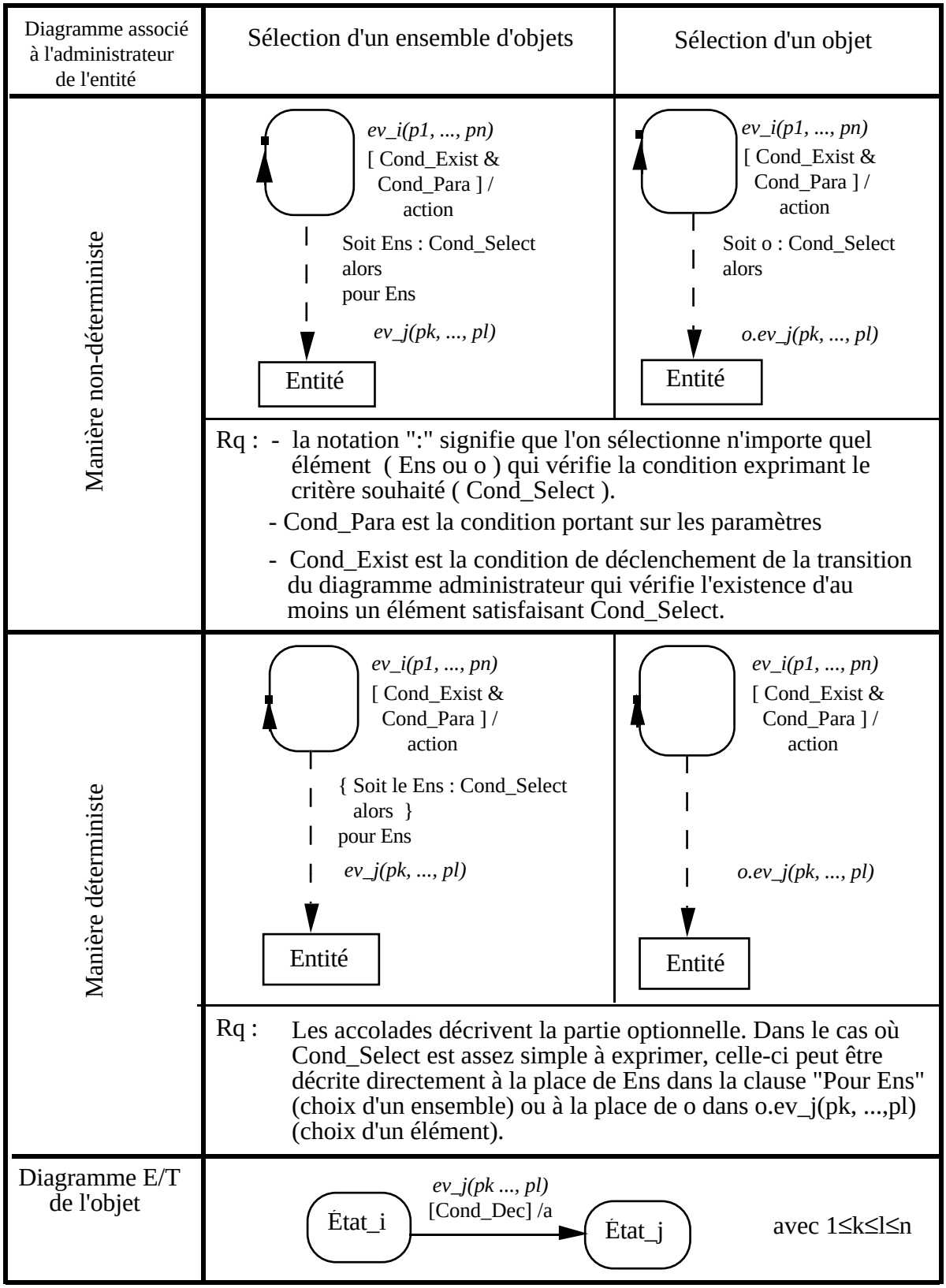


Figure 3.30. *Les différents cas de diagramme associé à l'administrateur d'entité*

Exemple :

1. Examinons le cas d'un emprunt d'exemplaire.

Diagramme d'objets :

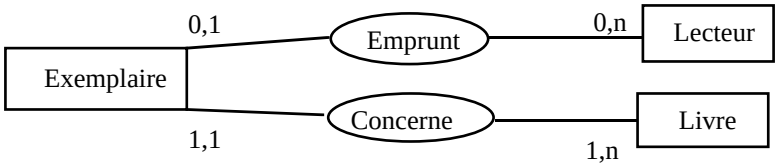


Diagramme associé à l'administrateur d'Exemplaire :

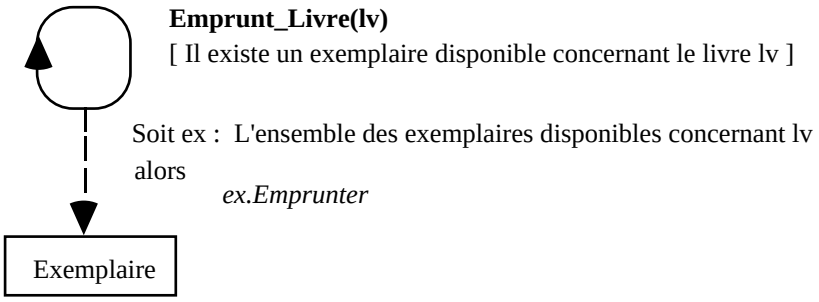


Diagramme E/T d'Exemplaire :

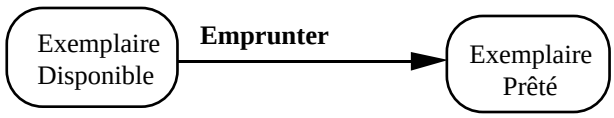


Figure 3.31. *Les différents diagrammes de l'exemple* Exemplaire_Livre_Lecteur *avec l'effet de l'événement Emprunt_Livre*

2. Le diagramme associé à l'administrateur de l'entité Élève et son diagramme E/T.

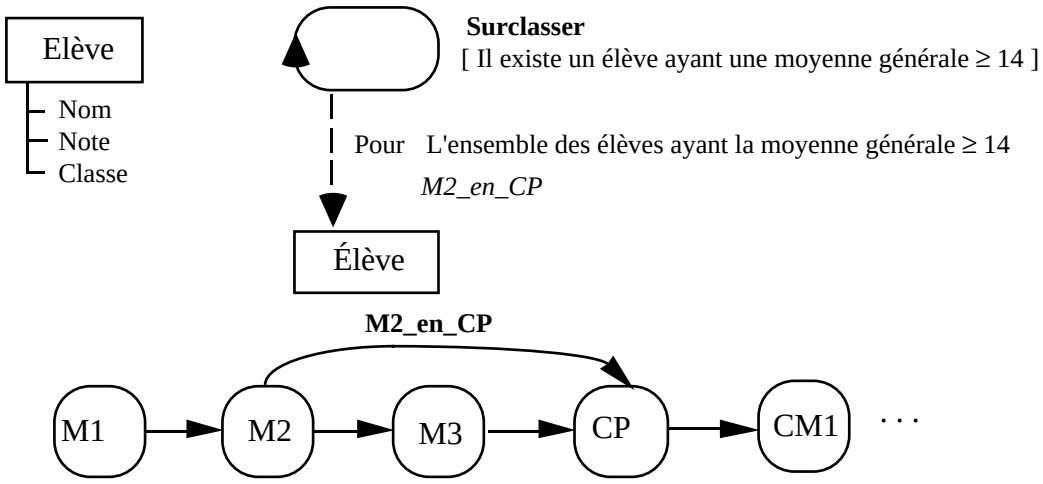


Figure 3.32. *Les différents diagrammes de l'entité* Élève *avec l'effet de l'événement Surclasser*

III.2.5. Scénario et Diagramme de suivi d'événements

III.2.5.1. Scénario

Dans la méthode OMT, un scénario est une suite d'événements associés à une exécution particulière (cf. exemples de la figure 3.33) du système. Un système peut être défini par plusieurs scénarios, correspondant à des vues particulières de son fonctionnement.

Rappelons que nous nous intéressons principalement aux systèmes d'information où l'aspect prépondérant est la gestion des données. Ce type d'applications peut la plupart du temps se décomposer en deux parties : une "couche externe" gérant les interactions avec l'utilisateur et la partie centrale dans laquelle est décrite la transformation des données. L'objet principal de notre étude est cette partie centrale.

De ce fait, nous nous limitons à des scénarios débutant par un événement d'entrée (aléatoire ou temporel), suivi d'un enchaînement d'événements internes et se terminant éventuellement par des événements de sortie.

Il existe une analogie entre la notion de scénario dans notre étude et la notion de transaction base de données.
Exemple :
Dans la figure 3.33 ci-dessous, Scénario 1 se compose de trois événements : Création_Cd, Ajoute_Cd, Enlève_Quantité.

III.2.5.2. Diagramme de suivi d'événements

Chaque événement transmet des informations d'un objet vers un autre. Par exemple, Enlève_Quantité est envoyé d'une commande à un produit. L'étape suivante, après avoir décrit les scénarios, est d'identifier les objets émetteurs et récepteurs de chaque événement. On obtient ainsi le diagramme de suivi d'événements.

Ce diagramme représente chaque entité par une ligne verticale et chaque événement par une flèche horizontale reliant l'objet émetteur à l'objet récepteur. Nous avons étendu ce diagramme en ajoutant une autre ligne verticale. Cette ligne est consacrée à l'objet Extérieur afin d'exprimer tous les événements venant de l'extérieur. Concernant les événement temporels, nous ajoutons une deuxième ligne verticale pour l'objet Horloge afin de préciser le temps.

Le diagramme de suivi d'événements se compose de tous les scénarios du système.

Exemple :

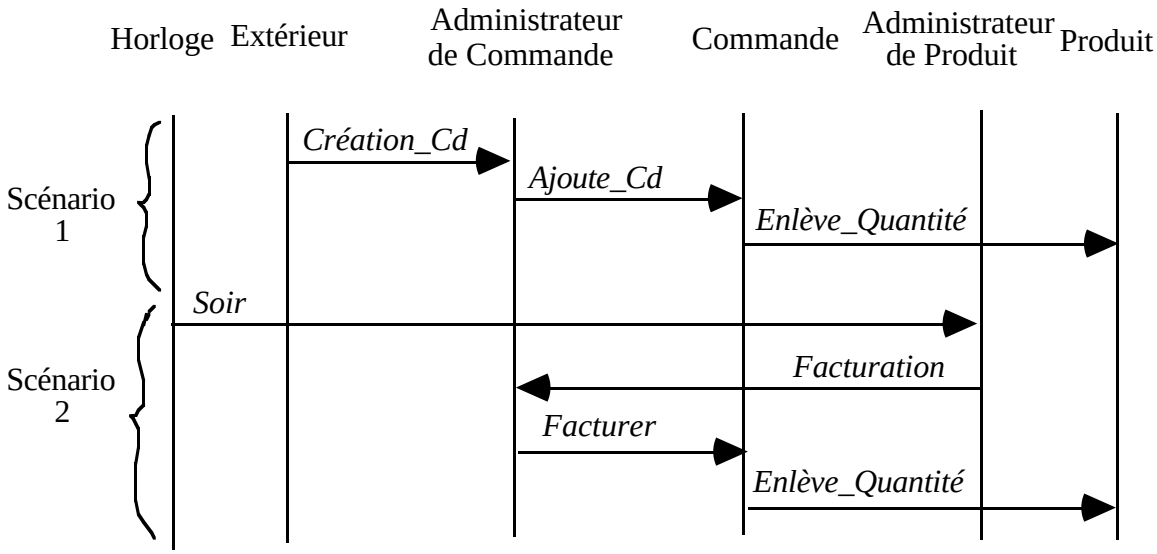


Figure 3.33. *Le diagramme de suivi d'événements avec deux scénarios*

Les diagrammes E/T correspondant à ces scénarios :

Diagramme E/T de l'Administrateur de Produit

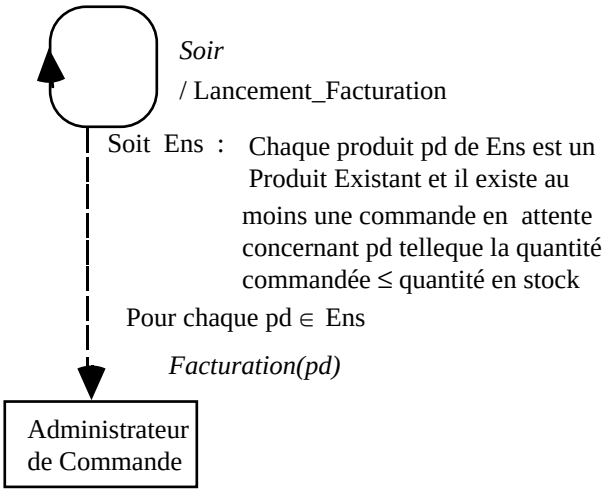


Figure 3.34. *Le diagramme de l'Administrateur de Produit pour*

le lancement de facturation de tous les produits

Diagramme E/T de l'Administrateur de Commande

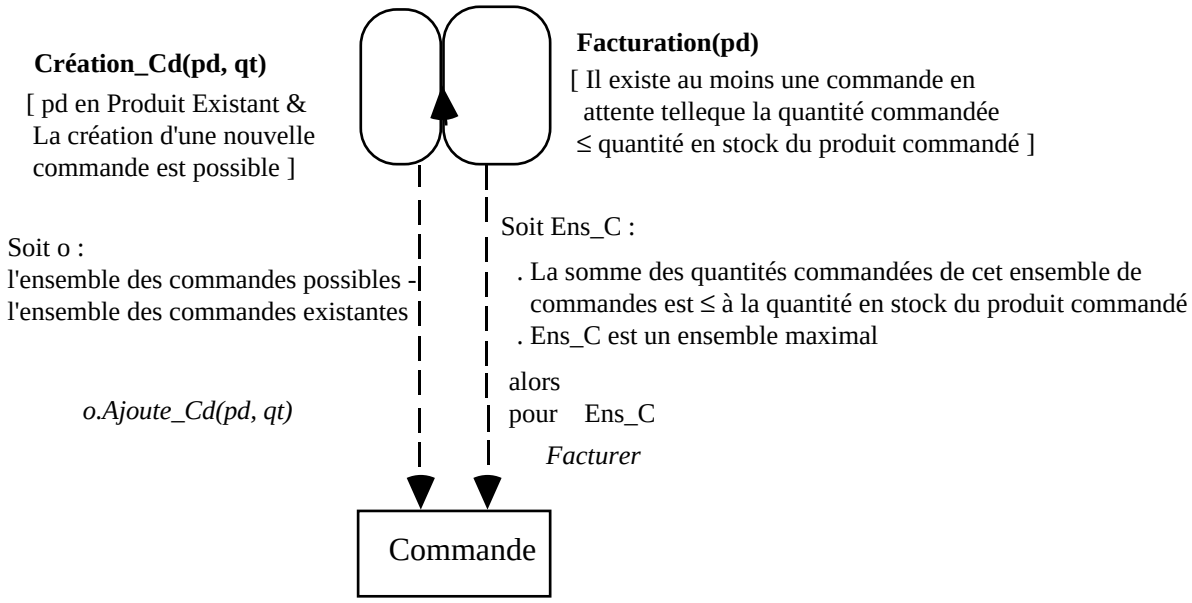


Figure 3.35. *Le diagramme de l'Administrateur de Commande pour la création d'une commande et la facturation d'un ensemble de commandes*

Diagramme E/T de l'entité Commande

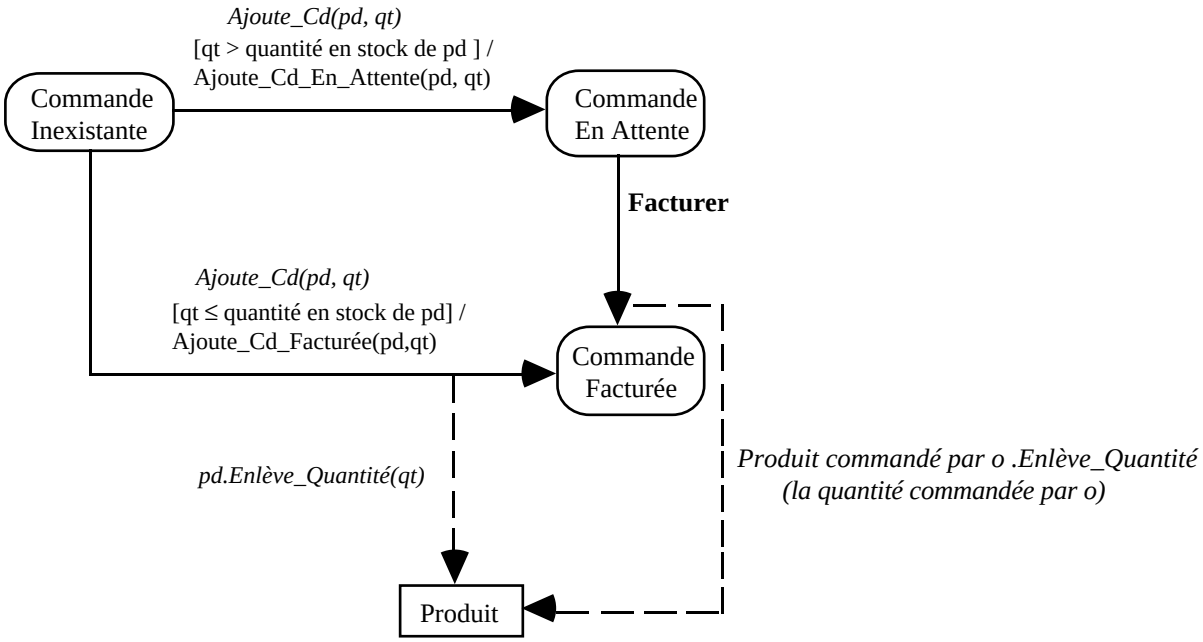


Figure 3.36. *Le diagramme E/T de l'entité Commande*

Diagramme E/T de l'entité Produit

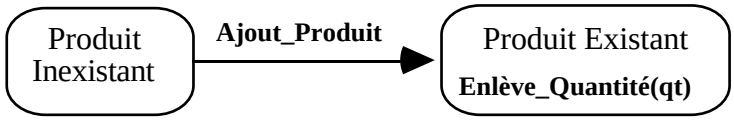


Figure 3.37. *Le diagramme E/T de l'entité Produit*

Scénario 1 :

Création_Cd est un événement externe envoyé à l'Administrateur de Commande. L'opération de l'administrateur associée à l'événement va choisir une nouvelle commande à ajouter si la condition sur la transition est remplie, c'est-à-dire que le produit existe et que la création d'une nouvelle commande est possible. La nouvelle commande à ajouter est choisie de manière non-déterministe (figure 3.35) : l'administrateur choisit n'importe quel o dans l'ensemble des commandes possibles moins l'ensemble des commandes existantes. À cet objet o est envoyé l'événement interne Ajoute_Cd. Selon la quantité en stock du produit commandé (supérieur ou inférieur à la quantité commandée), on va ajouter une commande facturée ou une commande en attente (figure 3.36). Dans le cas de l'ajout d'une commande facturée, un événement interne Enlève_Quantité doit être envoyé au produit commandé de l'entité Produit afin de diminuer la quantité en stock (figure 3.37).

Scénario 2 :

Soir est un événement périodique qui déclenche l'opération Lancement_Facturation de l'Administrateur de Produit. Pour chaque produit pd tel qu'il existe au moins une commande en attente concernant pd et tel que la quantité commandée soit inférieure ou égale à la quantité en stock, un événement Facturation(pd) est envoyé à l'Administrateur de Commande. L'opération Facturation va choisir un ensemble de commandes en attente à facturer concernant pd (Ens_C). Les critères de choix de cet ensemble sont les suivants (clauses sur la flèche de droite de la figure 3.35) :

- La somme des Quantité_Commandée de cet ensemble de commandes est inférieure ou égale à la Quantité_Stock du produit commandé,

- Cet ensemble est un ensemble maximal, c'est-à-dire que si nous ajoutons une autre commande dans l'ensemble, nous ne pouvons plus facturer toutes les commandes de cet ensemble.

Puis, à chaque commande de Ens_C, est envoyé l'événement *Facturer* (figure 3.36). Implicitement, les commandes de Ens_C sont en état En_Attente. L'événement interne Enlève_Quantité doit être envoyé ensuite au produit commandé de l'entité Produit pour diminuer la Quantité_Stock.

III.2.5.3. Les relations entre le diagramme de suivi d'événements et les diagrammes E/T

Le diagramme de suivi d'événements se compose de l'ensemble de tous les scénarios du système. Il décrit l'évolution du système et est utilisé dans la première phase d'analyse afin d'obtenir une vue synthétique et globale du système. Il permet de mieux construire ensuite les diagrammes E/T. Un scénario peut avoir plusieurs instances selon les conditions de déclenchement associées aux transitions des diagrammes E/T correspondant. Par exemple, deux instances possibles du scénario 1 de la figure 3.33 sont :

- Création_Cd, Ajoute_Cd, Enlève_Quantité (pour une commande facturée immédiatement),
- Création_Cd, Ajoute_Cd (pour une commande qui reste en attente car il n'y a pas assez de quantité en stock du produit commandé).

Le diagramme E/T est défini pour décrire l'évolution des objets d'une entité. Les paramètres des événements sont décrits dans les diagrammes E/T ainsi que les conditions de déclenchement. En revanche, ces paramètres et conditions peuvent être omis dans les scénarios afin d'alléger leur description.

En fait, on pourrait extraire les scénarios à partir des diagrammes E/T mais il faut, dans ce cas, effectuer un travail de recensement des objets concernés. Rappelons qu'un scénario est une suite d'événements qui débute par un événement d'entrée suivi par un enchaînement d'événements internes.

Lorsqu'un événement apparaît une seule fois dans l'ensemble des diagrammes E/T, cela signifie qu'il ne vient pas d'autres diagrammes E/T mais de l'extérieur. C'est donc un événement d'entrée (appelons le *ev_e*). À la prise en compte de cet événement *ev_e*, il peut se produire un événement interne *ev_i1* déclenché par *ev_e*. Si cet événement interne *ev_i1* ne

déclenche aucun autre événement interne, il est le dernier événement du scénario. Nous obtenons donc un scénario qui se compose de *ev_e* suivi de *ev_i1*. Sinon, nous suivons le nouvel événement interne (*ev_i2*) déclenché par *ev_i1* jusqu'au moment où un événement interne *ev_in* n'entraîne pas d'autres événements internes. La séquence *ev_e, ev_i1, ..., ev_in* forme alors ce scénario. Répétons cela pour tous les événements externes d'entrée, nous obtenons l'ensemble de tous les scénarios du système.

L'introduction de diagrammes de suivi d'événements facilite cette tâche et nous fournit une possibilité de vérification de la cohérence du modèle dynamique, en particulier la vérification de la cohérence entre les événements des différents diagrammes E/T. Par exemple, dans le scénario 1 de la figure 3.33, Enlève_Quantité est un événement interne du système venant de l'entité Commande vers Produit. On peut donc vérifier facilement dans les diagrammes E/T de Commande et Produit l'existence de cet événement.

Nous utilisons donc, dans cette recherche, ces deux types de diagrammes car ils sont complémentaires.

III.2.5.4. Choix de construction des scénarios et du diagramme états/transitions

Une fois élaboré le diagramme de suivi d'événements qui décrit l'évolution du système, il faut ensuite construire les diagrammes E/T qui représentent l'évolution des objets des entités. Pour cela, est-ce-que l'on prendra en compte l'ensemble de tous les états possibles du système à l'arrivée d'un événement donné ? Deux solutions sont à envisager.

Une première possibilité est de prévoir, dans le diagramme E/T, tous les cas (y compris les cas d'erreur) et d'assurer, pour chaque événement, qu'il y a toujours une et une seule transition dont la condition est vraie (approche "défensive"). Dans ce cas, il y a risque d'un manque de lisibilité dans les diagrammes E/T.

Dans la deuxième solution, qui a été adoptée pour ce travail, les diagrammes E/T ne traitent que les cas nominaux et sont contruits sous l'hypothèse suivante : un événement peut apparaître seulement si la condition sur la transition est vraie. Cela suppose qu'il existe par ailleurs un mécanisme de vérification des conditions de déclenchement (approche "offensive").

Cette deuxième solution correspond mieux à la modélisation de la partie centrale des systèmes considérés. Cette partie centrale suppose en effet que les données et les événements reçus sont cohérents.

III.2.5.5. Les différents types de scénarios

Dans cette partie, nous allons examiner les différents types de scénario correspondant à différentes structures de diagrammes E/T.

a). Scénario mono-entité

Il agit sur une seule entité. Nous pouvons distinguer trois cas :

a1) Scénario correspondant à une seule transition avec une seule action

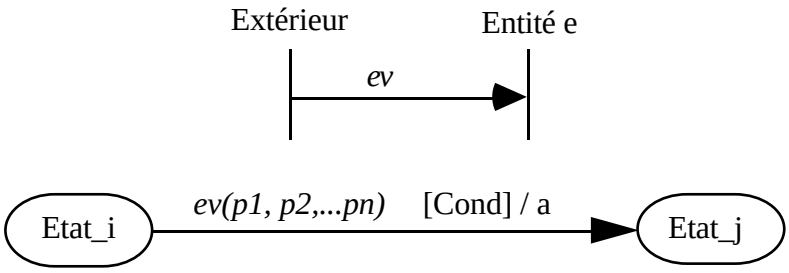


Figure 3.38. *Scénario correspondant à une seule transition avec une seule action*

L'événement externe ev, ayant comme paramètre p1, p2, ..., pn, est envoyé à l'objet courant o qui est dans l'État_i. La condition Cond étant supposée vérifiée, l'action a est exécutée et elle fait passer l'objet o dans l'État_j.

a2) Scénario correspondant à une seule transition avec plusieurs actions

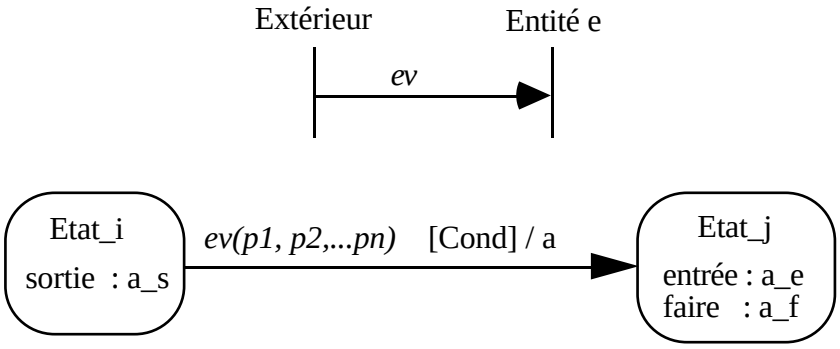


Figure 3.39. *Scénario correspondant à une seule transition avec plusieurs actions*

Comme dans le cas précédent, à la réception d'un événement ev (la condition de déclenchement est satisfaite), l'objet courant o passe de l'État_i à l'État_j. La différence avec le premier cas réside dans le fait que les deux états contiennent des actions. Plus concrètement, à l'arrivée de l'événement ev, la condition Cond est vraie, on exécute a_s puis a puis a_e et enfin a_f.

a3) Scénario correspondant à plusieurs transitions partant d'un même d'état avec des conditions de déclenchement différentes

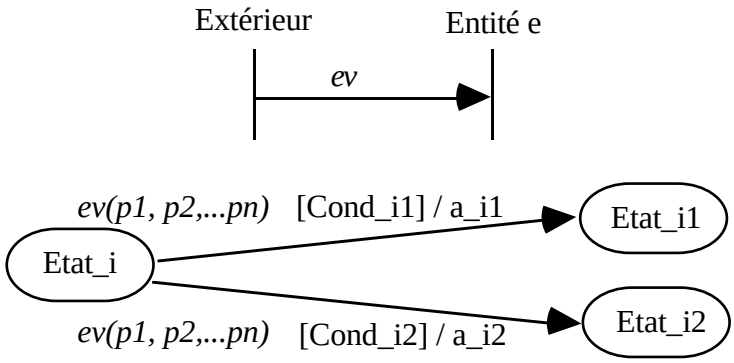


Figure 3.40. *Scénario correspondant à plusieurs transitions partant d'un même état avec des conditions de déclenchement différentes*

Nous avons ici un scénario avec choix. À l'arrivée de l'événement ev, une seule transition sera déclenchée en fonction des conditions sur les transitions. Plus concrètement, suivant que Cond_i1 ou Cond_i2 soit vraie, l'action a_i1 ou a_i2 est exécutée et fait passer l'objet courant dans État_i1 ou État_i2 (rappelons que Cond_i1 et Cond_i2 sont disjointes et qu'il y en a nécessairement une des deux qui est vraie).

a4) Scénario correspondant à plusieurs transitions partant d'états différents

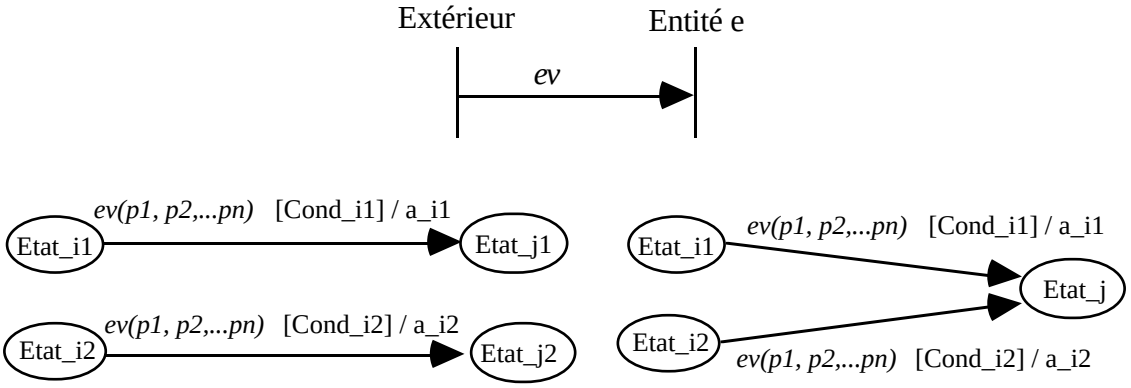


Figure 3.41. *Scénario correspondant à plusieurs transitions partant d'états différents*

À l'arrivée de l'événement ev, une seule transition sera déclenchée en fonction de l'état où se trouve l'objet courant o et des conditions sur les transitions. Contrairement au scénario a3, les conditions Cond_i1 et Cond_i2 peuvent être les mêmes car les états État_i1 et État_i2 sont distincts.

b). Scénario multi-entités

Comme nous avons vu dans la partie III.2.3, un transfert d'information d'un objet d'une entité vers un autre objet de la même ou d'une autre entité se fait en utilisant le mécanisme "Échange d'événements". Un scénario correspondant à ce type de diagramme est appelé scénario multi-entités car il fait évoluer plusieurs objets d'entités différentes. Nous avons le schéma général suivant :

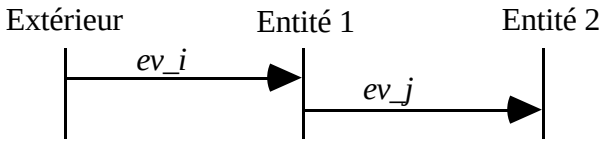


Diagramme de l'Entité 1 :

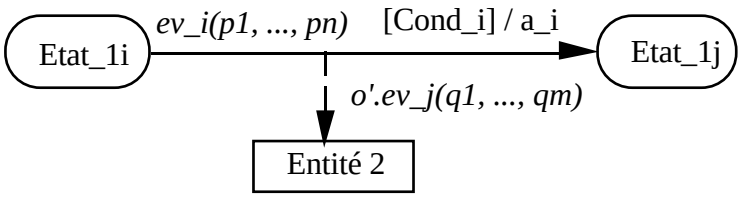


Diagramme de l' Entité 2 :

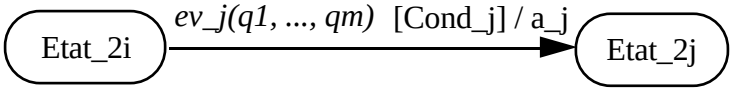


Figure 3.42. *Scénario multi-entités*

NB :

o' est : - soit calculé à partir de o (l'objet courant du diagramme E/T de l'Entité 1)
- soit reçu en tant que paramètre de l'événement ev_i

q1,...,qm sont des paramètres à transmettre à o' de l'Entité 2 qui dépendent du contexte.

Pour être cohérent avec la sémantique associée à la condition de déclenchement, quand la transition du diagramme de l'Entité 1 est franchie, l'événement ev_j est envoyé à l'objet o' du diagramme de l'Entité 2 si o' se trouve dans l'État_2i et si la condition Cond_j est vraie. Dans ce cas, a_i est exécutée et o passe dans l'État_1j, a_j est exécutée et o' passe dans l'État_2j. Dans le cas contraire, a_j ne sera pas exécutée.

c) Scénario associé au diagramme administrateur d'une entité

Reprenons les différents cas de diagrammes associés à l'administrateur de la figure 3.30. Nous remarquons que, dans le diagramme associé à l'administrateur, l'événement ev_i ne contient plus (implicitement) d'objet courant du diagramme. La transition décrit alors la sélection d'un objet ou d'un ensemble d'objets vérifiant le critère exprimé. L'action *ev_j* (du diagramme E/T de l'objet) ne porte pas obligatoirement, dans ce cas, sur un objet unique, mais son effet peut s'étendre simultanément sur plusieurs objets (cas choix d'un ensemble).

IV. Conclusion

Nous avons étudié, dans ce chapitre, les concepts des deux modèles : modèle d'objets et modèle dynamique. Concernant le modèle d'objets, les concepts les plus répandus tels que objet, entité, association, héritage ont été étudiés. D'autres concepts inspirés de diverses méthodes (MERISE, OMT) et les contraintes liées aux concepts sont également abordés. Les concepts du modèle d'objets étant suffisamment bien établis, nous nous sommes fixés comme objectif d'introduire un ensemble de concepts assez riche pour ne pas s'écarter de la réalité lors de la modélisation.

Devant la diversité des modèles dynamiques et la sémantique différence des concepts eux-mêmes, nous n'avons pas suivi la démarche appliquée pour le modèle d'objets. Le modèle dynamique de OMT a été choisi du fait de sa richesse en concepts et sa large diffusion.

Nous avons présenté les concepts de base proposés par OMT tels que événement, état, transition, condition, opération. Nous avons également éclairci certains points imprécis et fixé leur cadre d'utilisation. De la même manière, les concepts évolués de OMT ont été aussi examinés.

Nous avons introduit la notion de diagramme associé à l'administrateur d'une entité pour spécifier les opérations d'entité. Nous avons également établi les relations entre les diagrammes états/transitions et les diagrammes de suivi d'événements. L'introduction des types de scénarios nous permet de classifier les constructions possibles des diagrammes états/transitions. Ce classement a un rôle déterminant dans la suite de l'étude. Les concepts étudiés dans ce chapitre constituent les éléments de base du travail de dérivation dune spécification formelle à partir d'une spécification semi-formelle présenté dans le chapitre 5.

Chapitre 4

Présentation du langage de spécification de la méthode B

Nous présentons, dans ce chapitre, les notions de B nécessaires à la compréhension de ce travail.

I. Présentation générale

I.1. La méthode B

B [ABR 96] est une méthode de développement formelle du logiciel. Elle sert à formaliser la spécification, à expliciter la conception et à simplifier la programmation. Parmi l'ensemble des aspects centraux du cycle de vie des logiciels (à savoir : analyse des besoins, spécification, conception, codage, test et maintenance) la méthode B concerne les étapes suivantes : *spécification, conception et codage*. Chaque élément est considéré comme une activité qui doit être validée par des preuves. Une telle méthode permet de détecter les erreurs le plus tôt possible, d'être un guide à la conception et à la maintenance.

I.2. Le langage B

Le langage utilisé dans cette méthode (communément appelé le langage B) est constitué de trois langages complémentaires : le langage de la logique des prédicats et de la théorie des ensembles ; le langage des substitutions généralisées et le langage des machines abstraites.

- Le langage logique de base de B est basé sur la logique des prédicats du premier ordre avec égalité et paires ordonnées. Le langage de la théorie des ensembles de B est une simplification du langage de la théorie des ensembles classique. À partir des ensembles, sont définis, en particulier, les entiers, les relations (ensembles de paires), les fonctions (relation particulières) et les séquences (fonctions particulières).
  De nombreux opérateurs, en particulier relationnels, sont utilisés afin de permettre des spécifications très concises. La théorie est entièrement axiomatisée. Ce langage ensembliste permet de spécifier l'aspect statique, c'est-à-dire les données des systèmes, en particulier d'exprimer l'invariant qu'elles doivent vérifier.

- Le langage des substitutions généralisées sert à décrire l'aspect dynamique (opérations). Il comprend des "instructions" de spécification qui utilisent les notations de pré-condition et d'indéterminisme pour la phase de spécification et des "instructions" de programmation comme la séquence et la boucle pour les étapes finales de développement uniquement. La sémantique des substitutions généralisées est définie en terme de transformateurs de prédicats.

- Le concept de machine abstraite est semblable aux concepts de module, paquetage, classe et autres objets des langages de programmation. Il sert à construire les logiciels de manière modulaire et structurée. Une machine abstraite encapsule des données et les opérations qui les manipulent. Une machine peut être construite à partir d'autres machines.

Puisque le langage des machines abstraites se sert des deux autres langages, dans ce chapitre, nous allons nous restreindre à la présentation des machines abstraites. Cependant, les autres langages seront introduits au fur et à mesure dans cette présentation.

I.3. La notion de "machine abstraite"

En B, la spécification d'un logiciel est décomposée en sous-système : les "machines abstraites". Les composants d'une "machine abstraite" sont :

- l'en-tête ;
- les données : ensembles, variables
- les opérations

Les variables constituent l'état de la machine. Elles ne peuvent être directement modifiées de l'extérieur de la machine, mais le sont à travers les opérations de la machine. Les opérations constituent l'interface entre les variables et l'extérieur de la machine abstraite. L'état de la machine est contraint par un invariant, formule logique entre les variables de la machine. Cet invariant doit être vérifié après l'initialisation des variables, avant et après chaque opération. À chaque machine sont donc associées des obligations de preuve.

I.4. Les outils

Il existe deux ateliers de génie logiciel associés à la méthode B : l'Atelier B, développé par la société DIGILOG (France) [DIG 96] ; et le B-Toolkit développé par la société B-Core (Royaume-Uni) [COR 96]. Ces deux outils disposent principalement :

- d'un analyseur syntaxique des machines (au niveau de la spécification, raffinement ou implémentation),
- d'un contrôleur de types des machines,
- d'un générateur des obligations de preuves,
- d'un démonstrateur automatique de preuves,
- d'un démonstrateur interactif de preuves,
- d'un traducteur en langages de programmation impératifs (C ou ADA).

Ce chapitre est organisé comme suit. La partie II présente la syntaxe des données d'une "machine abstraite". La partie III présente la syntaxe des opérations d'une "machine abstraite". Dans la partie IV, nous décrivons les mécanismes de modularité dans les spécifications. La partie V est consacrée au raffinement de la spécification. La partie VI est dédiée à la présentation des obligations de preuve. Dans la dernière partie, nous donnons un exemple illustrant l'utilisation du langage B.

II. Syntaxe des données d'une "machine abstraite"

Les données d'une "machine abstraite" sont décrites au moyen des clauses suivantes :

MACHINE
en-tête de la machine
SETS
liste des ensembles abstraits et définition des ensembles énumérés
DEFINITIONS
liste des définitions
VARIABLES
liste des variables
INVARIANT
définition du type et des propriétés des variables
INITIALISATION
initialisation des variables

II.1. La clause "MACHINE"

La clause "MACHINE" introduit le nom qui identifie la machine.

II.2. La clause "SETS"

La clause "SETS" représente les ensembles introduits par la machine. Ce sont soit des ensembles abstraits, soit des ensembles énumérés définis par la liste de leurs éléments. Les ensembles abstraits sont utilisés pour désigner des objets dont on ne veut pas définir la structure au niveau de la spécification. Tout ensemble abstrait est défini non vide et fini, par exemple : SETS PERSONNES. Les ensembles énumérés sont définis par les éléments de leur énumération, par exemple : SETS SEXE = {Féminin, Masculin}.

II.3. La clause "DEFINITIONS"

La clause "DEFINITIONS" contient une liste d'abréviations pour un prédicat, une expression ou une substitution. Les définitions peuvent être utilisées dans la suite de la machine et sont locales à la machine où elles sont définies.

II.4. La clause "VARIABLES"

La clause "VARIABLES" introduit la liste des variables de la machine. Elles représentent l'état de la machine. Les variables sont les seuls objets qui peuvent être modifiés directement dans l'initialisation et les opérations de la machine. Cette clause doit être accompagnée des clauses "INVARIANT" et "INITIALISATION". Elle peut être absente si la machine n'a pas de variables.

II.5. La clause "INVARIANT"

La clause "INVARIANT" regroupe un ensemble de prédicats. Ces prédicats définissent les propriétés mathématiques des variables de la machine, et permettent de typer les variables et de définir certaines contraintes que doit vérifier l'état de la machine à tout moment. Cette clause est obligatoire si la clause "VARIABLES" est présente.

II.6. La clause "INITIALISATION"

La clause "INITIALISATION" se compose des substitutions qui définissent les valeurs initiales de chaque variable propre à la machine. Toute variable propre à la machine doit être initialisée. Cette initialisation doit satisfaire l'invariant de la machine. Cette clause est obligatoire si la clause "VARIABLES" est présente.

II.7. Expressions de la théorie des ensembles de B

La syntaxe utilisée dans l'invariant et les définitions reprend la syntaxe mathématique utilisée dans la théorie des ensembles. Nous y trouvons les quantificateurs (universel et existentiel), les connecteurs (et, ou, implication, ...), les types de base (NAT, STRING), les

opérateurs ensemblistes (union, inclusion, intersection, appartenance, ...), les relations (inverse, composition, ...), et autres opérateurs relationnels. Nous avons récapitulé les principales expressions dans le tableau suivant :

Nom	Syntaxe	Définition	Pré-condition
Produit cartésien	$s \times t$	$\{x \mapsto y \mid x \in s \wedge y \in t\}$*	
Relation	$s \leftrightarrow t$	$\mathbb{P}(s \times t)$ avec $\mathbb{P}(S)$: ensemble des parties de S	
Inverse	r^{-1}	$\{y,x \mid (y,x) \in t \times s \wedge (x \mapsto y) \in r\}$	$r \in s \leftrightarrow t$
Domaine	dom(r)	$\{x \mid x \in s \wedge \exists\ y.\ (y \in t \wedge (x \mapsto y) \in r)\}$	$r \in s \leftrightarrow t$
Codomaine	ran(r)	$dom(r^{-1})$	
Composition	q;r	$\{x,z \mid (x,z) \in s \times u \wedge \exists\ y.\ (y \in t \wedge (x \mapsto y) \in q \wedge (y \mapsto z) \in r)\}$	$q \in s \leftrightarrow t \wedge r \in t \leftrightarrow u$
Restriction	$u \lhd r$	$\{x,y \mid (x,y) \in r \wedge x \in u\}$	$r \in s \leftrightarrow t \wedge u \subseteq s$
Corestriction	$r \rhd v$	$\{x,y \mid (x,y) \in r \wedge y \in v\}$	$r \in s \leftrightarrow t \wedge v \subseteq t$
Anti-restriction	$u \lhd\!\!\!- r$	$\{x,y \mid (x,y) \in r \wedge x \notin u\}$	$r \in s \leftrightarrow t \wedge u \subseteq s$
Anti-Corestriction	$r \rhd\!\!\!- v$	$\{x,y \mid (x,y) \in r \wedge y \notin v\}$	$r \in s \leftrightarrow t \wedge v \subseteq t$
Image	r[w]	$\{y \mid y \in t \wedge \exists\ x.\ (x \in w \wedge (x \mapsto y) \in r)\}$	$r \in s \leftrightarrow t \wedge w \subseteq s$
Produit direct**	$f \otimes g$	$\{x,(y,z) \mid (x,(y,z)) \in s \times (u \times v) \wedge (x \mapsto y) \in f \wedge (x \mapsto z) \in g\}$	$f \in s \leftrightarrow u \wedge g \in s \leftrightarrow v$
Fonction partielle	$s \nrightarrow t$	$\{f \mid f \in s \leftrightarrow t \wedge \forall\ x,y,z.\ ((x \mapsto y) \in f \wedge (x \mapsto z) \in f \Rightarrow y = z)\}$	
Fonction totale	$s \rightarrow t$	$\{f \mid f \in s \nrightarrow t \wedge dom(f) = s\}$	
Injection partielle	$s \rightarrowtail t$	$\{f \mid f \in s \nrightarrow t \wedge f^{-1} \in t \nrightarrow s$	
Injection totale	$s \rightarrowtail t$	$s \rightarrowtail t \cap s \rightarrow t$	
Surjection partielle	$s \twoheadrightarrow t$	$\{f \mid f \in s \nrightarrow t \wedge ran(f) = t\}$	
Surjection totale	$s \twoheadrightarrow t$	$s \nrightarrow t \wedge s \rightarrow t$	
Bijection	$s \rightarrowtail t$	$s \rightarrowtail t \cap s \twoheadrightarrow t$	

Figure 4.1. *Les expressions principales de la théorie des ensembles de B*

* se lit ensemble des couples (x,y) telque x appartient à s et y appartient à t
la notion "$\mapsto$" correspond à la notion de couple

** nous utilisons souvent dans la suite le produit direct particulièrement entre deux fonction : soit par exemple, la fonction Père qui associe à chaque personne son père et la fonction Mère qui associe à chaque personne sa mère. Alors, le produit direct Père $\otimes$ Mère associe à chaque personne x le couple (Père de x, Mère de x).

III. Syntaxe des opérations d'une "machine abstraite"

Les opérations d'une "machine abstraite" sont spécifiées après la clause "OPERATIONS". Elles constituent la partie dynamique de la machine, par opposition aux données (les ensembles et les variables de la machine) qui constituent la partie statique. Elles permettent de modifier l'état de la machine (les variables) dans la limite de l'invariant.

La syntaxe d'une opération se compose de deux parties : un *en-tête* et un *corps*.

en-tête = corps

III.1. En-tête d'une opération

L'en-tête d'une opération est constitué d'un identificateur désignant le nom de l'opération et des éventuels paramètres formels d'entrée et de sortie de l'opération. Les paramètres d'entrée sont représentés par une liste parenthésée d'identificateurs qui suit le nom de l'opération. Les paramètres de sortie sont représentés par une liste d'identificateurs précédant le nom de l'opération. Les paramètres d'entrée et de sortie d'une opération doivent être deux à deux distincts. La syntaxe de l'en-tête d'une opération est sous la forme :

paramètres de sortie <- nom-op(paramètres d'entrée)

En B, lors de l'appel d'une opération, la sémantique du passage des paramètres est la copie. Les paramètres d'entrée de l'opération permettent de paramétrer un appel d'opération à l'aide de valeurs. Lors d'un appel d'opération, la valeur de chaque paramètre effectif d'entrée est recopiée dans le paramètre formel. Les paramètres de sortie de l'opération permettent de renvoyer les résultats d'un appel d'opération sous la forme de valeurs. Après un appel d'opération, la valeur de chaque paramètre formel de sortie est recopiée dans le paramètre effectif.

III.2. Corps d'une opération

Le corps d'une opération est constitué d'un ensemble d'instructions qui sont décrites par le langage des substitutions généralisées.

III.3. Langage des substitutions généralisées

L'ensemble des substitutions utilisables dans le langage B est décrit par le langage des substitutions généralisées. Ce langage est utilisé pour décrire le corps de l'initialisation et des opérations. Dans cette partie nous donnons la description de quelques substitutions principales.

Nom	Notation mathématique	Pré-condition
Skip	skip	
Devient égal	X := E	X est une variable E est une expression quelconque
Devient tel que	X : (P)	X est une variable P est un prédicat
Devient un élément de	X :∈ S	X est une variable S est un ensemble
Précondition	PRE P THEN S END	P est un prédicat S est une substitution quelconque
Begin	BEGIN S END	S est une substitution

IF	IF P THEN
	S
	ELSEIF Q THEN
	T
	ELSE
	U
	END
		P, Q sont des prédicats
		S, T, U sont des substitutions quelconques

ANY	ANY X WHERE P THEN S END	X est une liste de noms simples deux à deux dictincts P est un prédicat S est une substitution quelconque
Appel d'opération	[u ←] OP [(v)] la partie entre [] est optionnelle	
Simultanée	‖	

Figure 4.2. *Les principales substitutions généralisées*

III.3.1. Substitution "Skip"

La substitution *skip* est la substitution identité. Elle ne modifie pas les prédicats. Elle est notamment utile pour décrire que certaines branches d'une substitution IF ne font rien.

III.3.2. Substitution "Devient égal"

La substitution *devient égal* sert à remplacer des variables par des expressions. C'est la substitution de base à partir de laquelle seront définies les autres substitutions.

Exemple :
La substitution : X := 5
remplace l'ancienne valeur de X par "5".

III.3.3. Substitution "Devient tel que"

La substitution *devient tel que* permet de remplacer des variables par des valeurs qui satisfont à un prédicat donné. Si plusieurs valeurs satisfont le prédicat, la substitution ne précise pas laquelle est effectivement choisie. Elle définit un comportement non-déterministe.

Exemple :
On veut remplacer x par n'importe quelle valeur supérieure à sa valeur actuelle.

x : (x > x$0)

La valeur avant substitution d'une variable *x* peut être référencée par *x$0* dans le prédicat *P*. Cette possibilité est une facilité d'écriture qui évite d'introduire une variable intermédiaire.

III.3.4. Substitution "Devient un élément de"

La substitution *devient un élément de* permet de remplacer des variables par des valeurs appartenant à un ensemble. Si l'ensemble a plusieurs valeurs, la substitution ne précise pas laquelle est effectivement choisie. Elle définit un comportement non-déterministe.

Exemple :
Soient un ensemble S = {pêche, poire, prune} et la variable "fruit".
La substitution : fruit :∈ S
signifie que la variable "fruit" prend pour valeur un élément quelconque de l'ensemble S. On ne sait pas à priori s'il s'agit de l'élément poire, pêche ou prune.

III.3.5. Substitution "Pré-condition"

La substitution *pré-condition* permet d'introduire le prédicat qui fixe les conditions sous lesquelles une opération est appelée. Autrement dit, le comportement décrit par une substitution avec pré-condition n'est garanti que si dans le contexte d'utilisation sa pré-condition est valide.

Exemple :
Soit la substitution : PRE x ∈ NAT1 THEN x := x-1 END
Si la pré-condition (x ∈ NAT1) est vraie, alors la substitution "x := x-1" aura un sens.

III.3.6. Substitution "If"

La substitution *if* permet de définir pour une spécification plusieurs comportements possibles en fonction de la validité d'un ou de plusieurs prédicats. Le comportement défini par la substitution *If* est déterministe.

Exemple :
Soit la substitution :

```
IF      (x> 10)         THEN
        x := x-10
ELSEIF      (x=0)       THEN
        x := x+1
ELSE
        x := 1
END
```

Cette substitution définit trois comportements possibles selon la valeur de x.

Remarques
La substitution : IF x>10 THEN x:= x-10 END
est équivalente à IF x>10 THEN x:= x-10 ELSE skip END

III.3.7. Substitution "Any"

La substitution *ANY X WHERE P THEN S END* permet d'utiliser, dans la substitution *S*, les données abstraites déclarées dans la liste *X* et vérifiant le prédicat *P*.

Si plusieurs valeurs satisfont le prédicat *P*, la substitution ne précise pas laquelle est effectivement choisie. Elle définit alors un comportement non déterministe. Les données abstraites de la liste *X* sont alors accessibles en lecture, mais pas en écriture dans *S*, car ce ne sont pas des variables locales mais des données abstraites définies par le prédicat *P*.

Exemple :
Soit la substitution : ANY x WHERE $x \in NAT \wedge x \leq 10$ THEN y := x+1 END
Elle signifie que, pour n'importe quel x, tel que x soit un entier naturel, positif strictement et inférieur à 10, alors le comportement de la substitution est définie par "x := x+1".

III.3.8. Substitution "Appel d'opération"

La substitution *appel d'opération* permet d'appliquer la substitution d'une opération, en remplaçant les paramètres formels par des paramètres effectifs. Les paramètres d'entrée éventuels sont des expressions et les paramètres de sortie éventuels sont des données accessibles en écriture.

L'appel d'opération se définit sous quatre formes différentes, selon la présence de paramètres d'entrée et de sortie.

Exemple : opa ;
opb(x+1, TRUE) ;
res1, res2 ← opc ;
res, flag ← opd(x).

III.3.9. Substitution "Simultanée"

La substitution *simultanée* correspond à l'exécution simultanée de deux substitutions. Le caractère de simultanéité dénote le fait que les substitutions doivent pouvoir se réaliser

indépendamment l'une de l'autre. La substitution simultanée est commutative et associative. La substitution simultanée de deux substitutions *S* et *T* modifie des variables différentes.

Exemple : x := y ||
y := x

Dans l'exemple ci-dessus, les valeurs des variables *x* et *y* sont échangées.

IV. Modularité dans les spécifications

Dans la conception de logiciel de taille importante, la décomposition en plusieurs machines s'impose. Pour cela, J.R.Abrial introduit plusieurs clauses permettant de construire, de hiérarchiser et de composer des machines. Ces clauses sont SEES, USES, INCLUDES, EXTENDS, IMPORTS, REFINES. Ces mécanismes autorisent un développement progressif de la spécification, la séparation des preuves pour chaque machine. Comme l'étude porte sur la partie spécification initiale (la plus abstraite) et de ce fait utilise seulement les clauses USES, INCLUDES, PROMOTES, cette présente partie se restreindra à ces trois clauses.

IV.1. La clause "INCLUDES"

La clause "INCLUDES" permet de définir une liste de noms de machines qui seront incorporées dans la machine courante. Conceptuellement, les clauses SETS, INVARIANT et INITIALISATION de la machine incluse et de la machine courante sont regroupées : les ensembles et les variables d'une machine incluse deviennent des ensembles et des variables de la machine.

IV.1.1. Visibilité

Les variables de la machine incluse ne sont accessibles qu'en lecture dans la machine incluante. Les opérations sont visibles à partir des opérations de la machine incluante, et permettent de modifier ses variables (l'état de la machine incluse). Il est important de noter cependant qu'une opération de la machine incluante ne peut appeler qu'au plus une opération de la machine incluse, et ceci à cause des substitutions simultanées. En effet, si deux opérations

issues d'une même machine se font en même temps, il existe un risque de ne pas valider l'invariant de la machine incluse, lors de la modification de son état.

Soit M_A, une machine qui inclut la machine M_B. La table de visibilité suivante précise pour chaque constituant de M_B, les modes d'utilisation applicables dans les clauses de M_A.

Clauses de M_A / Constituants de M_B	INVARIANT	OPERATIONS
Ensembles	oui	oui
Variables	oui	en lecture
Opérations		oui

Figure 4.3. *Table de visibilité pour la clause INCLUDES _ M_A includes M_B*

IV.1.2. Transitivité

La clause INCLUDES est transitive. C'est-à-dire que les constituants d'une machine incluse qui deviennent des constituants de la machine incluante M peuvent être référencés dans les machines qui incluent M.

IV.1.3. Renommage

Dans certaines applications il est utile d'inclure une ou plusieurs machines identiques. La méthode B permet, par le mécanisme du renommage, d'éviter une collision de noms. Il se fait simplement par l'affectation d'un préfixe à la machine incluse dans la clause INCLUDES. Cela a pour but de renommer toutes les variables et les opérations de la machine renommée. Cependant, il est à noter que les ensembles ne sont pas renommées.

IV.2. La clause "USES"

La clause "USES" permet de définir une liste de noms de machines qui seront utilisés par la machine courante. La clause USES est utilisée uniquement pour le partage des données entre machines. La machine utilisée et toutes les machines qui l'utilisent doivent être incluses

dans une seule et même machine afin de contrôler que les invariants des machines utilisatrices qui s'appuient sur les variables de la machine utilisée ne sont pas contradictoires.

IV.2.1. Visibilité

Les ensembles de la machine utilisée sont visibles à partir de la machine utilisatrice. Les variables, quant à elles, peuvent figurer dans l'invariant de la machine utilisatrice et peuvent être consultés dans ses opérations. Les opérations d'une machine utilisée ne peuvent pas être utilisées dans la machine utilisatrice.

Soit M_A, une machine qui utilise une machine M_B. Nous décrivons les règles de visibilité dans le tableau suivant :

Clauses de M_A / Constituants de M_B	INVARIANT	OPERATIONS
Ensembles	oui	oui
Variables	oui	en lecture
Opérations		

Figure 4.4. *Table de visibilité pour la clause USES _ M_A uses M_B*

IV.2.2. Transitivité

La clause USES n'est pas transitive. Si une machine M_1 utilise une machine M_2 qui elle-même utilise une machine M_3, alors les ensembles et les variables de M_3 ne sont pas accessibles par M_1.

IV.3. La clause "PROMOTES"

Cette clause ne s'utilise que si la clause INCLUDES est déclarée. Elle contient une liste de noms d'opération des machines mentionnées dans la clause INCLUDES.

La clause PROMOTES permet d'inclure, dans la machine incluante, certaines opérations de la machine incluse. Ces opérations deviennent des opérations de la machine incluante. Par conséquent, une machine qui inclut de nouveau la machine incluante peut utiliser également ces opérations. La méthode B propose la clause EXTENDS qui correspond au PROMOTES de toutes les opérations de la machine incluse.

V. Raffinement

Un raffinement est la concrétisation d'une machine ou d'un autre raffinement. Un raffinement permet d'obtenir une machine plus concrète, c'est-à-dire plus proche d'un langage de programmation. Au cours d'un raffinement, les obligations de preuves assurent que la machine raffinant établit les invariants de la machine raffinée.

Le raffinement est assuré par les clauses REFINEMENT et REFINES. La clause REFINEMENT contient le nom de la machine produite par le raffinement. La clause REFINES indique le nom de la machine que l'on raffine.

Une machine peut être raffinée en plusieurs étapes dans le but de ne pas compliquer la réalisation des preuves à travers une succession de preuves incrémentales. La dernière étape du raffinement est dite "implémentation". Le principe général du raffinement est montré par le schéma ci-dessous :

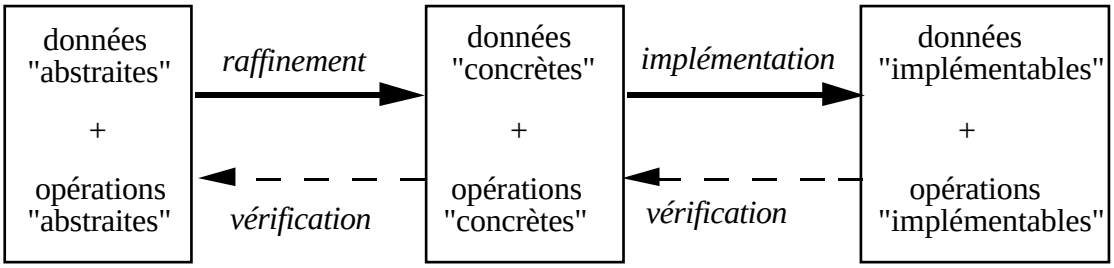


Figure 4.5. *Principe général du raffinement*

Un raffinement est dit algorithmique s'il transforme uniquement les opérations du composant raffiné. Il est dit un raffinement des données s'il introduit de nouvelles variables pour représenter, sous une forme plus proche des variables d'un langage de programmation, les variables du composant raffiné. Il peut aussi être une combinaison de ces deux types de

raffinement. Il faut prouver que la machine obtenue par un raffinement conserve les propriétés de la machine raffinée.

V.1. Raffinement des données

Le raffinement des données se fait en plusieurs étapes. Au cours des raffinements intermédiaires, les variables sont transformées en variables plus concrètes. Parfois, l'ajout de nouvelles variables est nécessaire par besoin de conception. La dernière étape (appelée aussi "implémentation") consiste à transformer les variables en structures de données implémentables et à attribuer les valeurs pour les ensembles.

V.2. Raffinement des opérations

Les opérations sont raffinées également en plusieurs phases : raffinements intermédiaires et raffinement final.

Pendant les phases de raffinement intermédiaire, il s'agit de :

- réduire le non-déterminisme,
- affaiblir les pré-conditions,
- introduire la séquence (éliminer la substitution simultanée),
- ajouter les détails de conception.

Le dernier raffinement consiste à :

- supprimer le non-déterministe,
- enlever les pré-conditions,
- supprimer la simultanéité,
- ajouter les détails de conception,
- introduire les itérations.

ce qui permet d'obtenir une implémentation de la spécification de départ.

Ce travail de recherche s'intéresse spécialement à la spécification. Nous avons donc introduit brièvement le raffinement. Une description détaillée peut être trouvée dans [ABR 96].

VI. Preuves

Soient S une substitution et P un prédicat. Alors, la notation :

[S] P (lire : "la substitution S établit le prédicat P")

représente le prédicat obtenu après transformation de P par la substitution S. Le vocabulaire suivant est également employé pour désigner cette transformation : on parle de l'établissement par la substitution S de la post-condition P. On parle aussi de la condition la plus large telle que, après avoir exécuté S, P soit vrai.

Étant donnée la machine suivante :

```
MACHINE          M
INVARIANT        I
INITIALISATION   INIT
OPERATION   OP =
            PRE  PC
            THEN      C
            END
END
```

Prouver l'exactitude de cette machine consiste à vérifier que l'initialisation INIT établit l'invariant I et que chaque opération OP, si elle est appelée sous sa pré-condition PC, avec invariant vrai, préserve l'invariant, c'est-à-dire :

$$[INIT]\ I$$

et $I \wedge PC \Rightarrow [C]\ I$

Rappelons que le corps de chaque opération se compose des substitutions généralisées (présentées dans la partie III.3). Par conséquent, l'effet d'une opération est défini à partir de celui des substitutions qui la composent.

Dans la suite, nous adoptons la convention suivante :

$\Leftrightarrow$: signifie "équivalent à prouver",

$\widehat{=}$: signifie une "définition",

pour la présentation de l'effet de chaque substitution abordée précédemment.

VI.1 Substitution "Skip"

Soit P un prédicat, alors :

[skip] P $\Leftrightarrow$ P

VI.2. Substitution "Devient égal"

1. Soit x une variable, E une expression et P un prédicat, alors :

[x := E] P $\Leftrightarrow$ dans P, on remplace les occurences libres de x par E

2. On définit, la substitution "devient égal" pour une liste de variables distinctes :

[x := E || y := F] P $\Leftrightarrow$ dans P, on remplace simultanément les occurences libres de x par E et de y par F

VI.3. Substitution "Devient tel que"

1. Soient P un prédicat, X une liste de variables modifiables et deux à deux distinctes, Y une liste de variables intermédiaires ayant autant d'éléments que X mais ne figurant pas dans X et P, alors :

X : (P) $\widehat{=}$ ANY Y WHERE [X := Y] P THEN X:=Y END

2. Soit une variable y de X. La notation y$0 est utilisable au sein de P. Elle représente la valeur de la variable y avant l'application de la substitution "devient tel que". Alors :

X : (P) $\widehat{=}$ ANY Y WHERE [X, y$0 := Y, y] P THEN X:=Y END

VI.4. Substitution "Devient un élément de"

Soient E une expression représentant un ensemble, X une liste de variables modifiables non vide et Y une liste de variables intermédiaires ayant autant d'éléments que X mais ne figurant ni dans X ni dans E. Alors :

$$X :\in E \;\widehat{=}\; \text{ANY } Y \text{ WHERE } Y \in E \text{ THEN } X := Y \text{ END}$$

VI.5. Substitution "Pre-condition"

Soient P et R des prédicats et S une substitution.

$$[\text{PRE } P \text{ THEN } S \text{ END}]\, R \Leftrightarrow P \wedge [S]\, R$$

VI.6. Substitution "If"

Soient P1, P2, ..., Pn et R des prédicats (avec $n \geq 1$) et soient S1, S2, ..., Sn et T des substitutions, alors :

1. $[\text{ IF P1 THEN S1 ELSE T END}]\, R \Leftrightarrow (P1 \Rightarrow [S1]\, R) \wedge (\neg P1 \Rightarrow [T]\, R)$

2. IF P1 THEN S1 END $\widehat{=}$ IF P1 THEN S1 ELSE skip END

3. [IF P1 THEN S1 ELSEIF P2 THEN S2 ... ELSEIF Pn THEN Sn ELSE T END] R $\Leftrightarrow (P1 \Rightarrow [S1]\, R) \wedge ((\neg P1 \wedge P2) \Rightarrow [S2]\, R) \wedge ... \wedge ((\neg P1 \wedge ... \wedge \neg Pn\text{-}1 \wedge Pn) \Rightarrow [Sn]\, R) \wedge ((\neg P1 \wedge ... \wedge \neg Pn) \Rightarrow [T]\, R)$

4. IF P1 THEN S1 ELSEIF P2 THEN S2 ... ELSEIF Pn THEN Sn END $\widehat{=}$ IF P1 THEN S1 ELSEIF P2 THEN S2 ... ELSEIF Pn THEN Sn ELSE skip END

VI.7. Substitution "Any"

Soient X une liste non vide de variables deux à deux distinctes, S une substitution et P et R deux prédicats, alors :

$$[\text{ANY } X \text{ WHERE } P \text{ THEN } S \text{ END}]\, R \Leftrightarrow \forall X.\, (P \Rightarrow [S]\, R)$$

VI.8. Substitution "Appel d'opération"

1. Soit op une opération sans paramètres de sortie et sans paramètres d'entrée, définie par op = S, alors la signification d'un appel de OP est :

$$[op]\ P \Leftrightarrow\ [S]\ P$$

2. Soit op une opération sans paramètres de sortie et avec des paramètres d'entrée, définie par op(X) = S où X est une liste d'identificateurs désignant les paramètres formels d'entrée de op, et soit E une liste d'expressions représentant les paramètres d'entrée effectifs de op, alors la signification d'un appel de op (E) est :

$$[op(E)]\ P \Leftrightarrow\ [X := E]\ [S]\ P$$

3. Soit op une opération avec paramètres de sortie et sans paramètres d'entrée, définie par Y ← op = S, où Y est une liste d'identificateurs désignant les paramètres formels de sortie de op, et soit R et une liste d'identificateurs éventuellement renommés désignant les paramètres effectifs de sortie de op, alors la signification d'un appel de R ← op est :

$$[R \leftarrow op]\ P \Leftrightarrow\ [S]\ [R := Y]\ P$$

4. Si op est une opération avec des paramètres de sortie et des paramètres d'entrée, définie par Y ← op(X) = S, la signification d'un appel de R ← op(E) est :

$$[R \leftarrow op(E)]\ P \Leftrightarrow\ [X := E]\ [S]\ [R := Y]\ P$$

VI.9. Substitution "Simultanée"

La substitution *simultanée* se définit de manière inductive à partir de certaines propriétés et par rapport aux autres substitutions du langage. Soient S1, S2, S3, T et U des substitutions, x et y des variables, X une liste d'identificateurs, E une expression, l1 et l2 des listes de constantes, P1, P2 et R des prédicats, alors :

Propriétés de la substitution simultanée :

1. S1 || S2 = S2 || S1

2. S1 || (S2 || S3) = (S1 || S2) || S3

Définition de la substitution "devient égal" simultanée :

3. x := E || y := F = x,y := E,F

Comportement de la substitution simultanée par rapport aux autres substitutions :

4. skip || S = S

5. X : (P) || S = ANY Y WHERE [X := Y] P THEN X := Y || S END

6. X :∈ E || S = ANY Y WHERE Y :∈ E THEN X := Y || S END

7. PRE P THEN S END || T = PRE P THEN S || T END

8. BEGIN S END || T = BEGIN S || T END

9. Si aucune variable élémentaire de X n'est libre dans T, alors :

ANY X WHERE P THEN S END || T = ANY X WHERE P THEN S || T END

10. IF P1 THEN S1 ELSE S2 END || T = IF P1 THEN S1 || T ELSE S2 || T END

VII. Exemple

VII.1. La machine abstraite

Afin d'illustrer l'utilisation du langage de la méthode B, nous présentons, dans cette partie, un exemple inspiré de [ABR 96, chapitre 4].

Dans cet exemple, nous décrivons une machine d'une base de données qui gère des informations sur un ensemble de personnes.

Chaque personne est supposée avoir les caractéristiques suivantes : sexe, situation, conjoint. On veut décrire les opérations suivantes :

- ajouter un homme célibataire,
- ajouter une femme célibataire,
- mariage,
- divorce,
- femme (d'un homme donné),
- mari (d'une femme donnée).

MACHINE BD-Personnes

SETS	PERSONNES,		
	SEXE={H,F}		
VARIABLES	Personnes, Sexe, Épouse		
INVARIANT	Personnes	$\subseteq$	PERSONNES $\wedge$
	Sexe	$\in$	Personnes $\rightarrow$ SEXE $\wedge$
	Épouse	$\in$	Hommes $\rightarrowtail\!\!\!\!\!\!\rightarrow$ Femmes
DEFINITIONS	Hommes	$\widehat{=}$	$Sexe^{-1}[\{H\}]$;
	Femmes	$\widehat{=}$	Personnes-Hommes ;
	Époux	$\widehat{=}$	$Épouse^{-1}$;
	Mariés	$\widehat{=}$	dom(Épouse) $\cup$ dom(Époux) ;
	Célibataires	$\widehat{=}$	Personnes-Mariés
INITIALISATION	Personnes, Sexe, Épouse := {}, {}, {}		

OPERATIONS	Ajoute_H_célibataire(h) =		
	PRE	h $\in$ PERSONNES - Personnes	
	THEN	Personnes	:= Personnes $\cup$ {h} $\|\|$
		Sexe(h)	:= H
	END ;		
	Ajoute_F_célibataire(f) =		
	PRE	f $\in$ PERSONNES - Personnes	
	THEN	Personnes	:= Personnes $\cup$ {f} $\|\|$
		Sexe(f)	:= F
	END ;		
	Mariage(h,f) =		
	PRE	h $\in$ Hommes - dom(Épouse) $\wedge$	
		f $\in$ Femme - dom(Époux)	
	THEN	Épouse(h)	:= f
	END ;		
	Divorce(h,f) =		
	PRE	(h,f) $\in$ Épouse	
	THEN	Épouse	:= {h} $\lhd$ Épouse
	END ;		
	f <- Femme(h) =		
	PRE	h $\in$ dom(Épouse)	

```
                    THEN        f := Épouse(h)
                    END ;
                    m <- Mari(f) =
                    PRE         f ∈ dom(Époux)
                    THEN        h := Époux(f)
                    END
END
```

VII.2. Obligations de preuve liées à la machine

Nous avons mentionné plus haut que les obligations de preuve liées à chaque machine consistent à vérifier que son initialisation établit son invariant et que chacune de ses opérations, si elle est appelée sous sa pré-condition, avec invariant vrai, préserve cet invariant.

Dans cette partie, nous présentons à titre d'exemple deux obligations de preuves de la machine BD-Personne. La première (1) vérifie que la partie initialisation établit l'invariant de la machine. La deuxième (2), l'opération Ajoute_H_célibataire(h) préserve l'invariant.

Nous avons :

(1) $[\ Personnes, Sexe, Épouse := \{\}, \{\}, \{\}\ ]\ Personnes \subseteq PERSONNES \ \wedge$
$Sexe \in Personnes \rightarrow SEXE \wedge Épouse \in Hommes \rightarrowtail Femmes$

(2) $Personnes \subseteq PERSONNES \wedge Sexe \in Personnes \rightarrow SEXE \wedge$
$Épouse \in Hommes \rightarrowtail Femmes \wedge h \in PERSONNES - Personnes$

$\Rightarrow [\ Personnes := Personnes \cup \{h\}\ \|$
$Sexe(h) := H\ ]\ Personnes \subseteq PERSONNES \ \wedge$
$Sexe \in Personnes \rightarrow SEXE \wedge Épouse \in Hommes \rightarrowtail Femmes$

VIII. Conclusion

Au terme de cette présentation, nous pouvons remarquer que la méthode B répond bien à certains critères présentés dans le chapitre 1 concernant le choix d'un langage formel.

En effet, le langage de spécification de la méthode B permet facilement de :

- gérer des ensembles d'objets existants et des ensembles d'objets possibles dans une même structure
- définir des identités d'objet indépendamment de leur valeur
- modéliser le concept d'association à un haut niveau d'abstraction
- bien modulariser la spécification (en plusieurs machines abstraites avec des liens *Uses, Includes*,...)

De plus, la méthode B couvre les étapes centrales du cycle de vie d'une application à savoir la spécification, la conception par raffinement et le codage, chacune de ces étapes étant validée par des preuves. En outre, elle est supportée par des outils utilisés industriellement. Toutes ces raisons nous ont poussé à adopter cette méthode pour notre étude.

Chapitre 5

Règles de dérivation d'une spécification formelle B à partir d'une spécification semi-formelle

I. Introduction

Dans les chapitres 3 et 4, nous avons présenté les concepts des méthodes semi-formelles ainsi qu'une présentation de la méthode formelle B. Nous nous intéressons maintenant à la dérivation d'une spécification formelle B à partir d'une spécification semi-formelle. Cette dérivation est constituée de deux phases.

La première phase consiste à traduire le modèle d'objets en B (partie II). Elle comprend trois étapes. La première concerne la formalisation des concepts du modèle d'objets (sous-partie II.1). La deuxième est consacrée à l'étude de la modularisation de la spécification obtenue par l'étape précédente (sous-partie II.2). Puis, nous examinons la génération des opérations de base des entités et des associations (sous-partie II.3).

La deuxième phase traduit le modèle dynamique (partie III). Notons que, dans les diagrammes états/transitions, certaines informations sont décrites en langue naturelle (condition, critères de choix d'un ensemble d'objets, ...). Donc, une traduction automatique comme celle du modèle d'objets n'est pas possible. Nous devons, dans un premier temps, intégrer les notations B dans les diagrammes états/transitions (sous-partie III.1) puis donner une formalisation de chaque état en B (sous-partie III.2). Les deux étapes suivantes s'adressent à la génération de squelettes d'opérations B et d'obligations de preuve à partir des diagrammes états/transitions et scénarios (sous-partie III.3) et à la répartition des opérations dans les machines (sous-partie III.4). Dans la sous-partie III.5, nous présentons une autre technique possible utilisant le raffinement.

Nous terminons ce chapitre par la description du prototype implémentant les règles proposées (partie IV).

II. Traduction du modèle d'objets

II.1. Formalisation des concepts du modèle d'objets

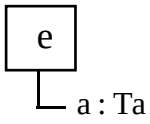
Dans cette partie, nous présentons les règles de traduction de chaque concept et contrainte du modèle d'objets en B. Notons que, lorsqu'une règle donne lieu à un prédicat invariant, cela ne signifie pas que le prédicat donné apparaît nécessairement tel quel (syntaxiquement) comme un conjoint dans l'invariant de la machine concernée, mais qu'il doit être conséquence de cet invariant. La traduction des opérations sur les entités, associations dépend fortement de la manière dont nous allons décomposer la spécification en machines abstraites et sera reportée donc dans la partie II.3.

II.1.1. Objet et concepts liés

Une spécification formelle, fidèle à l'approche par objets, doit bien séparer l'existence des objets de leurs caractéristiques. Certes, l'identité d'objet peut être réalisée à l'aide de clés (sous-ensemble de caractéristiques dont les valeurs identifient un objet) ou à l'aide d'identifiants internes (comme dans les bases de données objets), mais cela ne doit certainement pas apparaître au premier niveau d'une spécification. En fait, la formalisation de l'existence des objets ne requiert que des ensembles d'identités d'objet, sans aucune structure et donc pas d'autre opérateur que l'égalité. Cela correspond typiquement aux "given sets" de B, ensembles donnés sans aucune structure pour leurs éléments.

La plupart des contraintes et des opérations portant sur des ensembles d'objets existants, nous aurons donc :

Règle S1



Chaque **entité *e*** donne lieu à :

- l'ensemble des (identités d') objets possibles de *e* : **Set S_e**
- l'ensemble des (identités d') objets existants de *e* : **Variables V_e**

- l'invariant: $\mathbf{V_e \subseteq S_e}$

Exemple:

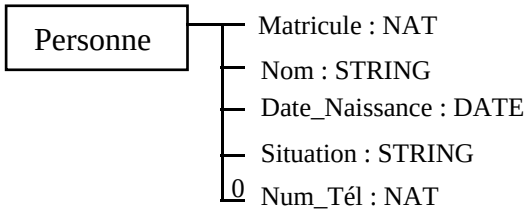


Figure 5.1. *Un exemple d'entité avec cinq attributs*

L'entité Personnes donnera lieu à l'ensemble PERSONNES des personnes possibles, à la variable Personnes ensemble des personnes existantes et à l'invariant Personnes $\subseteq$ PERSONNES, toute personne p existante vérifie donc p $\in$ Personnes.

La formalisation des attributs se fait naturellement à l'aide de relations et fonctions entre les ensembles d'identités d'objet et les types des attributs. Cette approche est similaire à celle de [LAN 94b], [NGU 95] et [LED 95] (ce dernier se situe dans le contexte de Z). Notons qu'une relation *r* est un ensemble de paires, *dom(r)* est l'ensemble des premiers éléments dans ces paires (i.e. éléments ayant une image), *ran(r)* est l'ensemble des seconds éléments (i.e. images). On obtient donc :

Règle S2

Chaque **attribut *a*** de type $\mathbf{T_a}$ de l'**entité *e*** donne lieu à :

- la relation définissant les valeurs de *a* pour les objets de *e* : **Variables** $\mathbf{V_a}$
- l'**invariant** de typage de V_a : $\mathbf{V_a \in V_e \leftrightarrow T_a}$ (cas le moins contraint)

Selon que l'attribut est obligatoire ou optionnel, multi ou mono-valué, la relation peut devenir une fonction partielle (symbole $\rightarrow\!\!\!\!\!\!\cdot\;$) ou totale (symbole $\rightarrow$). Par exemple, si l'attribut est mono-valué et optionnel, V_a est une fonction partielle : $V_a \in V_e \rightarrow\!\!\!\!\!\!\cdot\; T_a$; si il est multi-valué et obligatoire : $V_a \in V_e \leftrightarrow T_a \wedge dom(V_a) = V_e$.

Exemple :

Pour l'entité Personne ci-dessus, nous obtenons la machine suivante :

MACHINE	MA_Personne
SETS	PERSONNES
VARIABLES	Personnes, Matricule, Nom, Date_Naissance, Situation, Num_Tél
INVARIANT	Personnes $\subseteq$ PERSONNES $\wedge$

Matricule ∈ Personnes ⤖ NAT ∧
Nom ∈ Personnes → STRING ∧
Date_Naissance ∈ Personnes → DATE ∧
Situation ∈ Personnes → STRING ∧
Num_Tél ∈ Personnes ⇸ NAT

INITIALISATION

Personnes, Matricule, Nom, Date_Naissance, Situation, Num_Tél := {}, {}, {}, {}, {}

END

Remarques

- Identités d'objet et clés :

  Comme indiqué ci-dessus, l'identité d'objet est formalisée en définissant, pour chaque entité, l'ensemble des identités des objets possibles et l'ensemble des identités des objets existants.
  En revanche, le concept de clé est formalisé par un invariant exprimant que les objets sont en injection avec leurs clés : si *(a1, a2)* est une clé, alors nous avons :

  $$Va1 \otimes Va2 \in Ve \rightarrowtail T_{a1} \times T_{a2}$$

- Ensembles d'identités d'objets :

  Il est naturel d'utiliser, pour chaque entité, l'ensemble des identités des objets possibles et l'ensemble des identités des objets existants, comme nous l'avons fait :

  Personnes ⊆ PERSONNES, Voitures ⊆ VOITURES, etc.

  Mais il faut noter que ce n'est pas la seule solution possible ; nous pourrions avoir un unique ensemble OBJETS pour toutes les identités d'objets possibles avec : Personnes ⊆ OBJETS, Voitures ⊆ OBJETS … (avec des contraintes de disjonction).

  Une troisième solution est d'avoir non seulement l'ensemble OBJETS mais aussi l'ensemble de tous les objets existants (objets ⊆ OBJETS), l'ensemble de tous les noms d'entité et une relation instance_de entre objets et entités. Cette solution offre une possibilité de définir des fonctions générales entre objets et de formaliser explicitement les concepts du méta-modèle. Elle a été étudiée plus profondément dans [MON 97].

  Avec différents ensembles d'identités existantes et plus encore différents ensembles d'identités possibles (solution adoptée ici), il est possible d'obtenir une encapsulation des

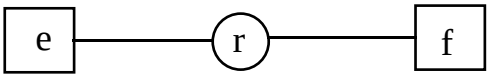
entités dans différentes machines (comme nous le verrons dans la partie modularisation, II.2).

II.1.2. Association et concepts liés

II.1.2.1. Lien, Association

Comme une instance d'association (i.e. un lien) est identifiée par la paire d'objets connectés, il n'est pas nécessaire de fournir un ensemble des instances possibles associé à chaque association comme pour une entité. La formalisation la plus directe d'une association est donc une relation entre les ensembles d'identités d'objets existants. Nous obtenons donc :

Règle S3



Chaque **association *r*** entre les **entités *e* et *f*** donne lieu à :

- une **variable Vr**, ensemble des paires d'instances de *e* et *f* liées par *r*
- l'**invariant** de typage : $\mathbf{V_r \in V_e \leftrightarrow V_f}$ (cas le moins contraint)

Selon les cardinalités, cette relation, ou son inverse V_r^{-1}, peut être une fonction partielle, une injection, etc. Nous donnons quelques exemples où la première (resp. seconde) cardinalité est celle de *e* (resp. *f*) dans l' association *r* :

(0,n),(0,n)	=>	$V_r \in V_e \leftrightarrow V_f$	(relation)
(1,n),(1,n)	=>	$V_r \in V_e \leftrightarrow V_f$	$\wedge\ dom(V_r)=V_e \wedge ran(V_r)=V_f$
(0,n),(1,1)	=>	$V_r^{-1} \in V_f \rightarrow V_e$	(fonction totale)
(0,1),(0,1)	=>	$V_r \in V_e \rightarrowtail V_f$	(injection)
(1,1),(1,1)	=>	$V_r \in V_e \twoheadrightarrow V_f$	(bijection)

Pour les attributs d'une association, nous appliquons la même règle (règle S2 ci-dessus) que pour les attributs des entités.

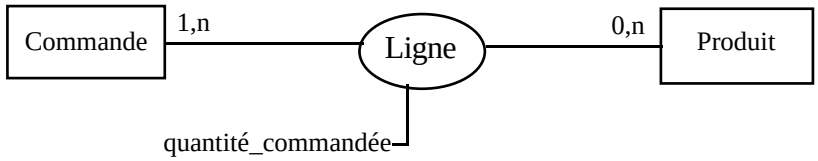


Figure 5.2. *Un exemple d'association*

MACHINE	Diagram_fig_2
SETS	COMMANDES, PRODUITS
VARIABLES	Commandes, Produits, Ligne, quantité_commandée
INVARIANT	Commandes $\subseteq$ COMMANDES $\wedge$ Produits $\subseteq$ PRODUITS $\wedge$ Ligne $\in$ Commandes $\leftrightarrow$ Produits $\wedge$ dom(Ligne) = Commandes $\wedge$ quantité_commandée $\in$ Ligne $\rightarrow$ NAT
END	

Remarque

- le sens de la relation (soit $V_e \leftrightarrow V_f$ soit $V_f \leftrightarrow V_e$) n'est, en général, pas déductible du schéma lui-même mais nécessite soit une interprétation du nom de l'association soit une convention graphique supplémentaire.

- les contraintes ensemblistes entre associations s'expriment très facilement, car les relations sont aussi des ensembles (de paires). Par exemple, le diagramme d'objets peut indiquer, par une quelconque notation, que l'association dirige entre personne et service implique l'association membre (toute personne qui dirige un service en est membre). On ajoutera simplement l'invariant : dirige $\subseteq$ membre. Nous verrons plus loin que des contraintes entre associations nettement plus complexes s'expriment également de manière très concise.

II.1.2.2. Rôle

Il s'agit, en fait, de définir ***propriétaire_de*** et ***voitures_de*** à partir de la variable formalisant l'association (de type : *Possède* $\in$ *Personne* $\leftrightarrow$ *Voiture*).

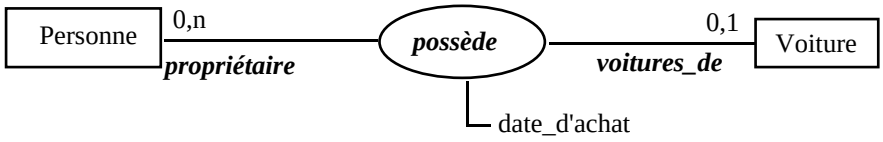
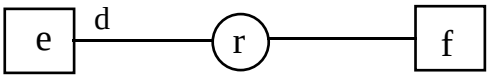


Figure 5.3. *Un exemple de rôles dans une association*

Un rôle est ce qu'en B, on appelle une définition.

Règle S4



chaque **rôle *d* d'une association *r*** donne lieu à :

- une **définition d(x) ≙ DEF**

avec DEF définie à partir de la variable **Vr**, selon le sens et les cardinalités de *r*

Exemple :
Sur l'exemple précédent, on obtient :

DEFINITIONS	propriétaire_de(x) ≙ Possède^{-1}(x) ; voitures_de(x) ≙ Possède[{x}]

Remarque

Les définitions sont également utilisées pour les éventuels attributs dérivés, attributs dont la valeur est calculable à partir de celles d'autres attributs. Par exemple, Âge est calculé à partir de l'attribut Date_Naissance et de la date courante.

II.1.2.3. Association fixe

Nous avons défini ce concept car il est important pour le découpage en machines abstraites de la spécification formelle finale. Le fait d'être fixe ne concerne pas l'aspect statique de l'association qui suit donc la règle générale de représentation en B. Cela concerne l'aspect dynamique. Le fait que l'ensemble des objets $V_r[\{x\}]$ déjà associés à un objet *x* de *e* ne peut être modifié par aucune opération qui conserve *x* (i.e. qui préserve $x \in V_e$) est traduit par l'obligation de preuve suivante :

Règle S5

Chaque **association *r* fixe pour l'entité *e*** *d*onne lieu à :

l'**obligation de preuve** pour chaque opération OP :

$$(x \in V_e \wedge V_r[\{x\}] = q) \Rightarrow [OP]\,(x \in V_e \Rightarrow V_r[\{x\}] = q)$$

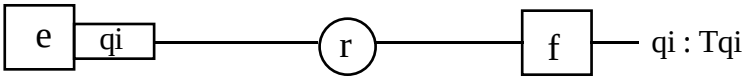
où *[OP]Q* est le prédicat signifiant que l'opération *OP* établit la condition *Q*.

Autrement dit, l'association n'a pas d'opération propre : une instance d'association est créée (resp. supprimée) par l'opération qui crée (resp. supprime) une instance de l'entité pour laquelle elle est fixe. C'est cette remarque qui justifie l'importance de cette notion pour la décomposition modulaire : une association sans opération propre n'a pas à avoir de machine propre.

II.1.2.4. Qualification

Ce concept est traduit en appliquant le même principe que pour le concept de clé.

Règle S6



Chaque **qualification *(e, {q1, q2, …})*** de l'entité ***f*** dans l'association ***r***
donne lieu à l'**invariant :**

$$(V_r^{-1} \otimes V_{q1} \otimes V_{q2} \otimes \dots)^{-1} \in V_e \times T_{q1} \times T_{q2} \times \dots \nrightarrow V_f$$

Selon la cardinalité de *f* dans l'association, la fonction précédente peut devenir une injection (0,1), une surjection (1,n) ou une bijection (1,1) mais est généralement partielle avec l'ensemble de départ ci-dessus.

S'il n'y a qu'un seul attribut dans la qualification, nous pouvons restreindre l'ensemble de départ et obtenir une fonction totale : $(V_r^{-1} \otimes V_q)^{-1} \in Vr ; Vq \rightarrow V_f$. En effet on a toujours : $ran(R \otimes S) = R^{-1};S$.

Exemple :

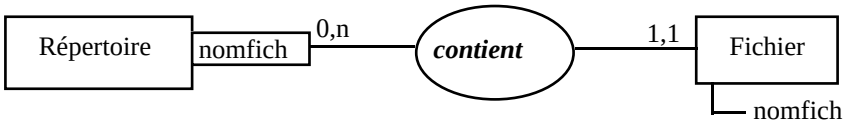


Figure 5.4. *Un exemple de qualification*

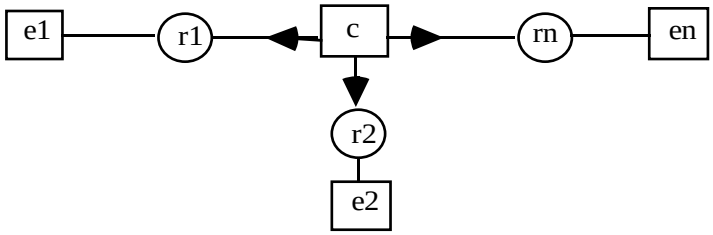
Nous obtenons dans cet exemple :

$$(\text{contient}^{-1} \otimes \text{nomfich})^{-1} \in \text{contient} ; \text{nomfich} \rightarrowtail\!\!\!\!\!\rightarrow \text{Fichier}$$

II.1.2.5. Entité associative

La formalisation en B d'une entité associative est la même que pour une entité avec un invariant supplémentaire exprimant les contraintes : chaque cardinalité de l'entité associative vers les entités associées vaut (1,1) et chaque tuple d'instances des entités associées détermine au maximum une instance de l'entité associative.

Règle S7



Chaque **entité associative *c* déterminée par les entités e_1, e_2, …, e_n, vers les associations r_1, r_2, …, r_n,** donne lieu à l'**invariant** supplémentaire :

$$\mathbf{Vr_1 \otimes Vr_2 \otimes ... \otimes Vr_n \in V_c \rightarrowtail\!\!\!\!\!\rightarrow V_{e1} \times V_{e2} \times ... \times V_{en}}$$

Remarques

- Les contraintes sur chaque V_{ri} ($V_{ri} \in V_c \rightarrow V_{ei}$) sont déductibles de l'invariant précédent.
- Concernant la dynamique, il faut ajouter les obligations de preuve provenant du fait que chaque V_{ri} est fixe pour V_C (en appliquant la règle S5).

- Si un attribut *a* de type T_a participe à la détermination de l'entité associative, nous avons :

$$Vr_1 \otimes \ldots \otimes Va \otimes \ \ldots \otimes Vr_n \in V_c \rightarrowtail V_{e1} \times \ldots \times T_a \times \ldots \times V_{en}$$

Exemple :

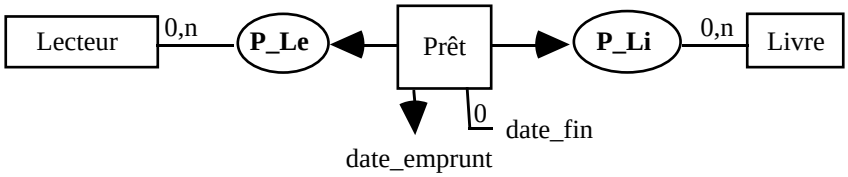


Figure 5.5. *Entité associative avec attribut propre dans les valeurs déterminantes*

Ainsi, le diagramme de la figure 5.5 donne la machine :

MACHINE Diagram_fig_5
SETS LECTEURS, LIVRES, PRETS
VARIABLES Lecteurs, Livres, Prêts, P_Le, P_Li, date_emprunt, date_fin
INVARIANT Lecteurs $\subseteq$ LECTEURS $\wedge$
Livres $\subseteq$ LIVRES $\wedge$
Prêts $\subseteq$ PRETS $\wedge$
date_emprunt $\in$ Prêts $\rightarrow$ DATE $\wedge$
date_fin $\in$ Prêts $\nrightarrow$ DATE $\wedge$
P_Le$\otimes$P_Li$\otimes$date_emprunt $\in$ Prêts $\rightarrowtail$ Lecteurs $\times$ Livres $\times$ DATE
END

(les typages P_Le $\in$ Prets $\rightarrow$ Lecteurs et P_Li $\in$ Prets $\rightarrow$ Livre se déduisent de l'invariant).

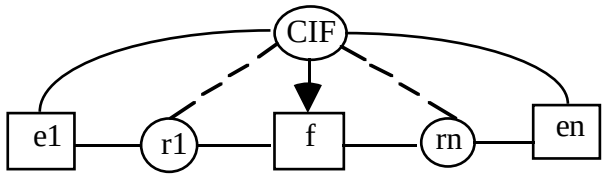
II.1.2.6. Agrégation

Pour chaque lien d'agrégation, nous appliquons les règles de modélisation des liens d'association. Il s'agit, en effet, d'un cas particulier d'association.

II.1.2.7. Contraintes d'intégrité

a. Contrainte d'intégrité fonctionnelle

Règle S8



Chaque **CIF (e1, e2, ..., en)** de l'**entité f** dans les **associations r1, r2, ..., rn** ($Vr_i \in Vf \leftrightarrow Ve_i$) donne lieu à l'**invariant** :

$$(Vr_1 \otimes Vr_2 \otimes \cdots \otimes Vr_n)^{-1} \in Ve_1 \times Ve_2 \times \cdots \times Ve_n \rightarrow Vf$$

Exemple :

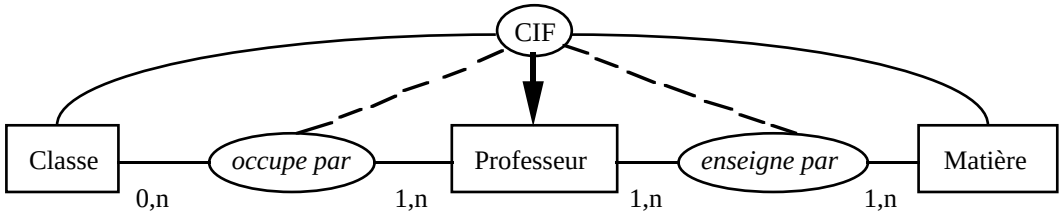


Figure 5.6. *Un exemple de CIF entre Classe, Matière et Professeur*

Les deux associations occupe-par et enseigne-par donnent lieu aux invariants suivants :

occupe-par $\in$ Professeur $\leftrightarrow$ Classe $\wedge$
dom(occupe-par) = Professeur $\wedge$
enseigne-par $\in$ Professeur $\leftrightarrow$ Matière $\wedge$
dom(enseigne-par) = Professeur $\wedge$
ran(enseigne-par) = Matière

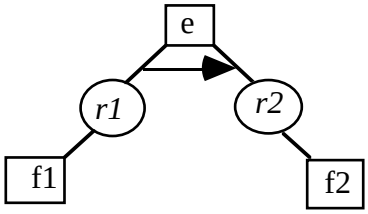
La contrainte d'intégrité fonctionnelle donne l'invariant suivant :

$$(\text{occupe-par} \otimes \text{enseigne-par})^{-1} \in \text{Classe} \times \text{Matière} \rightarrow \text{Professeur}$$

b. Contraintes entre ensembles d'instances participant à deux associations

- Contrainte d'inclusion

Nous avons le schéma général suivant :



Cette contrainte signifie que :

Toute instance de *e* qui participe à *r1* doit participer à *r2* .

Avec $Vr_i \in Ve \leftrightarrow Vf_i$, cela veut dire que :

Pour tout *x* tel que *x* appartient à *dom(Vr1)* implique *x* appartient à *dom(Vr2)* .

Cette contrainte est traduit en B de la manière suivante :

Règle S9

Chaque contrainte d'inclusion entre deux associations r1 et r2 et une entité e commune à r1 et r2 donne lieu à l'invariant :

$$\mathbf{dom(Vr_1) \subseteq dom(Vr_2)}$$

Exemple :

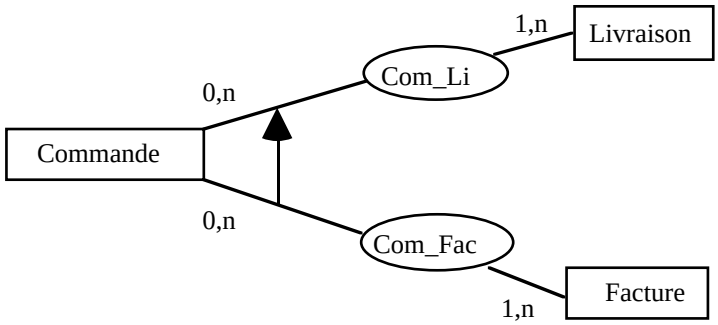
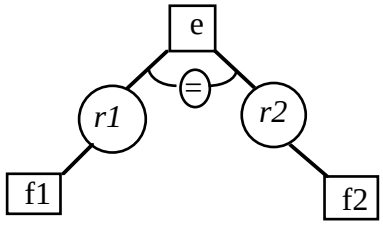


Figure 5.7. *Un exemple de contrainte d'inclusion entre Com_Li et Com_Fac avec l'entité Commande*

Nous obtenons donc, pour cet exemple : dom(Com_Fac) $\subseteq$ dom(Com_Li)

- Contrainte d'égalité

Règle S10



Chaque contrainte d'égalité entre deux associations r1 et r2 et une entité e commune à r1 et r2 donne lieu à l'invariant : **$dom(Vr_1) = dom(Vr_2)$**

Exemple :

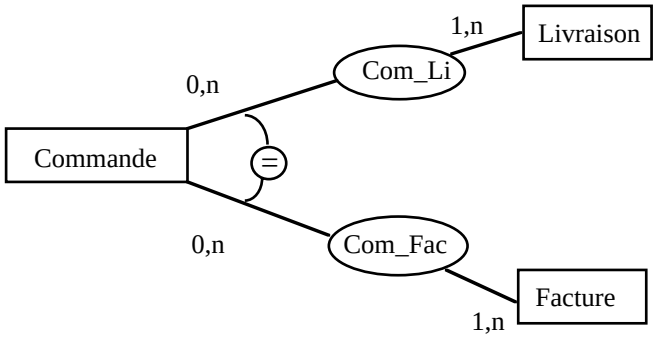
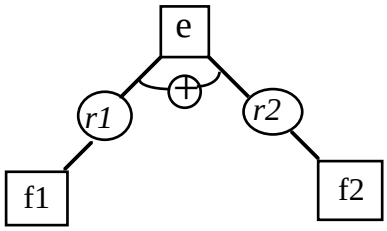


Figure 5.8. *Un exemple de contrainte d'égalité entre Com_Li et Com_Fac avec l'entité Commande*

Nous obtenons donc, pour cet exemple : dom(Com_Fac) = dom(Com_Li)

- Contrainte d'exclusion

Règle S11



Chaque contrainte d'exclusion entre deux associations r1 et r2 et une entité e commune à r1 et r2 donne lieu à l'invariant :

$$dom(Vr_1) \cap dom(Vr_2) = \emptyset$$

Exemple :

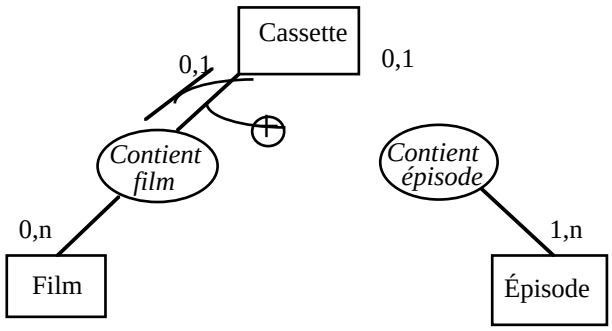


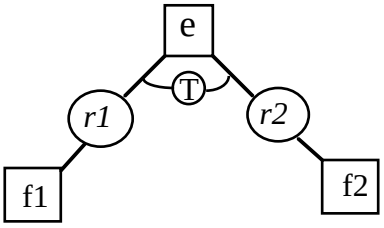
Figure 5.9. *Un exemple de contrainte d'exclusion entre Contient Film et Contient Episode avec l'entité Cassette*

Nous obtenons donc, pour cet exemple :

$$dom(Contient\_Film) \cap dom(Contient\_Épisode) = Ø$$

- Contrainte de totalité

Règle S12



Chaque contrainte de totalité entre deux associations r1 et r2 et une entité e commune à r1 et r2 donne lieu à l'invariant :

$$\mathbf{dom(Vr_1) \cup dom(Vr_2) = Ve}$$

Exemple :

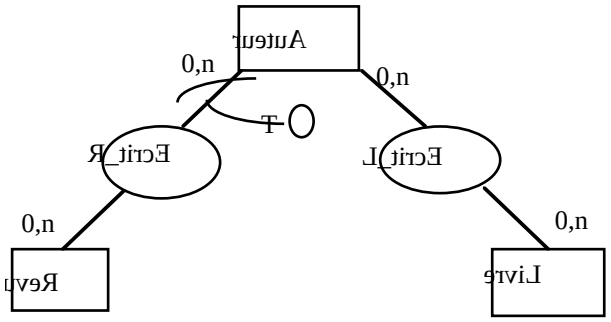


Figure 5.10. *Un exemple de contrainte de totalité entre Livre et Revue avec l'entité Auteur*

Nous obtenons donc, pour cet exemple :

$$dom(Contient\_Film) \cup dom(Contient\_Épisode) = Auteur$$

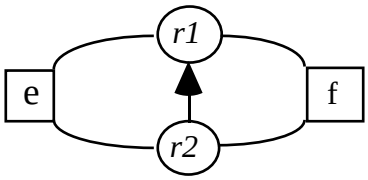
Remarque

La généralisation des règles S11 et S12 à n associations est immédiate.

c. Contrainte entre ensembles de liens

Nous donnons comme exemple dans cette partie la contrainte d'inclusion entre deux associations.

Règle S13



Chaque contrainte d'inclusion entre deux associations r1 et r2 qui relient deux entités e et f donne lieu à l'invariant :

$$Vr_2 \subset Vr_1$$

Exemple :

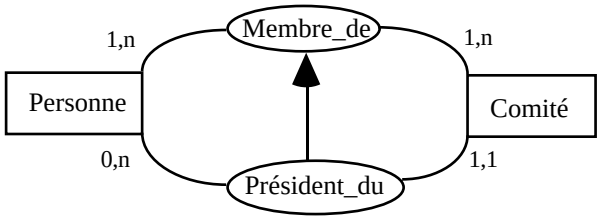


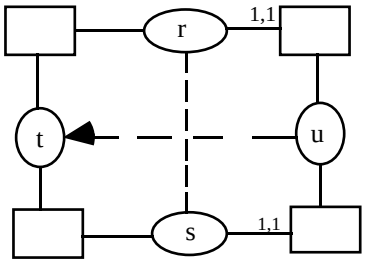
Figure 5.11. *Contrainte d'inclusion entre ensembles de liens*

Dans cet exemple le président d'un comité doit être membre de ce comité ; l'association Président_du est donc un sous-ensemble de l'association Membre_de. Nous obtenons :

Président_du ⊂ Membre_de

d. Contrainte d'homomorphisme

De manière générale, nous avons le schéma :



Une contrainte d'homomorphisme (indiquée par la flèche en pointillé) entre l'association *u* et l'association *t* via les associations *r* et *s* signifie que :

x,y sont associés par *u* ssi leurs images *r(x),s(y)* sont associées par *t.*

Donc la paire *x,y* appartient à *u* ssi :

il existe *x' ,y'* tels que la paire *x,x'* appartient à *r*, la paire *x',y'* appartient à *t* , la paire *y',y* appartient à s^{-1} .

Soit finalement : $u = r ; t ; s^{-1}$ (où ; dénote la composition de relations).

On obtient donc la règle :

Règle S14

Chaque contrainte d'homomorphisme entre une association *u* et une association *t* via les associations *r* et *s*, donne lieu à l'invariant :

$$\boldsymbol{u = r ; t ; s^{-1}}$$

Exemple :

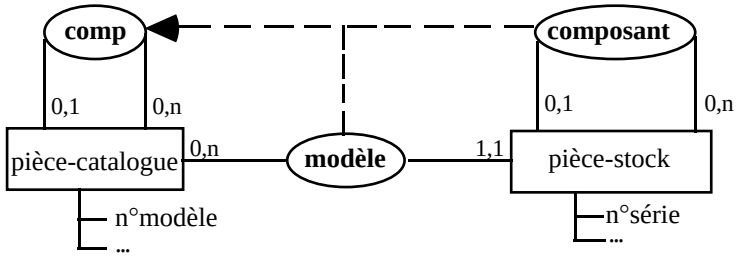


Figure 5.12. *Homomorphisme de composant vers comp, via modèle*

Cet exemple est un cas particulier où les associations *r* et *s* sont les mêmes (à savoir l'association *modèle*). On obtient donc :

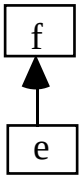
$$composant = modèle ; comp ; modèle^{-1}$$

II.1.3. Héritage et concepts liés

II.1.3.1. Lien d'héritage

Dans ce paragraphe, nous ne considérons que les aspects statiques et nous nous limitons à l'héritage simple. L'aspect dynamique (héritage d'opérations) est lié à la décomposition en plusieurs machines et est donc étudié dans le paragraphe suivant. Ces concepts sont formalisés en raffinant la règle S1 par :

Règle S15



Pour chaque **sous-entité *e* d'une entité *f*** :

- le "given set" (instances possibles) **pour e est le même que pour f**

- invariant supplémentaire: **Ve ⊆ Vf**

Remarques

- *Ve* ⊆ *Se* est conséquence de cet invariant et, du fait que *Se* et *Sf* , sont identiques.

- Il n'y a qu'un "given set" pour toute la hiérarchie d'héritages. Il est associé à l'entité de plus haut niveau et représente donc l'ensemble de tous les objets possibles dans la hiérarchie. Par contre, il y a toujours une variable d'état par entité.
  Exemple :

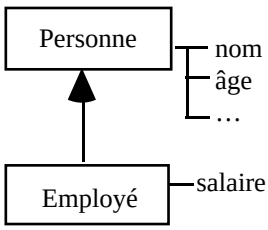


Figure 5.13. *Exemple de lien is-a*

- En appliquant la règle S15, la figure 5.13 donne :
  Personnes ⊆ PERSONNES ∧ Employés ⊆ Personnes

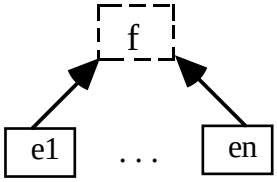
- L'héritage d'attributs se fait automatiquement comme conséquence de la contrainte d'inclusion : une fonction applicable sur un sur-ensemble de Ve est applicable sur Ve. Par exemple, comme nom ∈ Personnes → STRING, nom est applicable à toute instance de Employés, puisque Employés ⊆ Personnes. Par contre comme salaire ∈ Employés → STRING, salaire n'est pas applicable pour Personnes.

II.1.3.2. Contrainte de couverture entre sous-entités

Soit e_i, i=1 ..n, des sous-entités de *f*, la contrainte de couverture s'exprime comme suit :
Chaque instance de *f* doit appartenir à l'une des sous-entités (e_1, e_2, ..., e_n).
C'est-à-dire : Pour tout *x* tel que *x* appartient à *f* implique :

x appartient à e_1 ou
x appartient à e_2 ou
...
x appartient à e_n

Règle S16



Chaque **contrainte de couverture** est traduite en B par l'**invariant** suivant :

$$\mathbf{V_f = V_{e1} \cup V_{e2} \cup ... \cup V_{en}}$$

Exemple :

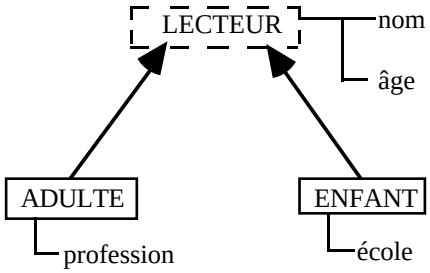


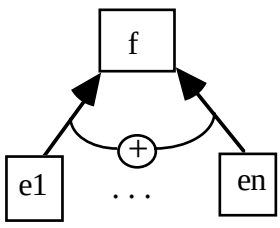
Figure 5.14. *Exemple de contrainte de couverture*

Cette exemple donne : Lecteur = Adulte ∪ Enfant

II.1.3.3. Contrainte de disjonction entre sous-entités

Soit e_i, i=1 ... n, des sous-entités de f, la contrainte de disjonction exprime que :
Les ensembles d'instances des sous-entités sont disjoints.

Règle S17



Chaque **contrainte de disjonction** est traduite en B par l'**invariant** suivant :

$$\mathbf{V_{e1} \cap V_{e2} \cap ... \cap V_{en} = Ø}$$

Exemple :

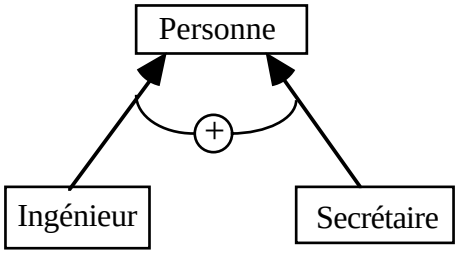


Figure 5.15. *Exemple de contrainte de disjonction*

Cette exemple donne : Ingénieur $\cap$ Secrétaire = Ø

II.2. Modularisation de la spécification

La décomposition en unités faiblement couplées est indispensable pour maîtriser la complexité de tout système, qu'il s'agisse d'une spécification ou d'une réalisation. Elle doit en particulier permettre une conception incrémentale, faciliter la compréhension, l'évolutivité et la réutilisation de composants du système. En B, l'unité de conception est une *machine abstraite*. Rappelons que les machines sont reliées par des clauses *includes* (permettant d'appeler les opérations d'une autre machine) et *uses* (permettant d'accéder aux variables d'une autre machine en lecture uniquement).

Dans cette partie, nous étudions la décomposition en machines d'une spécification obtenue par les règles précédentes. Notre objectif est de suivre le plus fidèlement possible la structure initiale du modèle d'objets afin de retrouver facilement, dans la spécification finale, les éléments du schéma, facilitant ainsi le va-et-vient entre la spécification et le schéma. Chaque méthode a ses propres critères de modularisation qui ne sont pas nécessairement compatibles (même si on peut penser qu'il y des liens étroits entre les deux) : les mécanismes de B ont principalement pour but de séparer les preuves à effectuer alors que, dans les diagrammes, le but est d'offrir à l'utilisateur une vue synthétique et fidèle du système à modéliser.

II.2.1. Entité

Une entité encapsulant à la fois les propriétés statiques et comportementales d'un ensemble d'objets, il semble naturel de faire correspondre, à chaque entité, une machine abstraite :

Règle S18
A chaque **entité *e* d'attributs $a_1, \ldots, a_n$** correspond
une machine abstraite *Me* avec les variables ***Ve*, $Va_1, \ldots, Va_n$**

Dans la machine entité *Me*, la clause*Sets* comprend, au minimum, l'ensemble des instances possibles sauf s'il s'agit d'une sous-entité (avec, éventuellement, d'autres ensembles nécessaires aux définitions d'attributs), la clause *variables* comprend l'ensemble *Ve* des instances existantes et la relation Va_i décrivant chaque attribut a_i, la clause *invariant* comprend le typage des variables ainsi que d'autres contraintes comme les contraintes d'intégrité ou d'entité associative. La clause *operations* comprend les opérations qui manipulent une instance ou un ensemble d'instances. Outre les *obligations de preuve* de base (assurant la cohérence entre invariant et opérations), nous associons, à chaque machine, des obligations de preuve assurant les propriétés dynamiques du modèle d'objets, par exemple pour les entités associatives.

II.2.2. Association

Dans les systèmes d'information orientés gestion de données, les associations jouent un rôle essentiel. De nombreuses opérations sont en effet des créations, suppressions, ou modifications de liens entre objets. Comme nous l'avons vu en II.1.2.1, la formalisation la plus directe du concept d'association (binaire) est la relation, c'est-à-dire un ensemble de paires. Le problème maintenant est de décider où mettre la variable représentant l'association : dans la machine d'une des entités associées ou dans une troisième machine, propre à l'association ?

Les solutions proposées dans la littérature ont consisté, soit à créer systématiquement une machine pour chaque association [NAG 94], soit inversement, à mettre systématiquement la variable association dans une machine entité [LAN 94b]. Aucune de ces solutions n'apparaît réellement satisfaisante : la première produit trop de machines, certaines inintéressantes car les invariants importants sont dans les machines incluantes. A l'inverse, la seconde solution peut inclure trop d'information dans la même machine : certaines associations peuvent présenter beaucoup d'opérations soit sur les liens eux-mêmes soit sur les attributs des liens, et sont donc plutôt des candidates naturelles à une machine indépendante.

En fait, une prise en compte plus fine des caractéristiques de l'association nous a conduit à la solution suivante : nous créons une machine propre à l'association si et seulement si elle est sujette à des opérations indépendantes des entités, c'est-à-dire à des opérations soit sur les attributs des liens soit sur les liens eux-mêmes. De manière évidente, il y a des opérations sur les attributs dès qu'il y a des attributs. Il y des opérations indépendantes sur les liens si ces liens ne sont fixes pour aucune des entités. Rappelons qu'une association *r* est fixe pour une entité *e* si l'ensemble des liens *r* concernant chaque objet de *e* est créé en même temps que l'objet et ne peut être modifié durant la vie de l'objet.

Nous obtenons alors :

Règle S19

Si une **association *r* entre *e* et *f*** est :

19-cas 1 : non fixe (ni pour *e* ni pour *f*) ou avec attributs : il lui correspond:

- **une machine abstraite *Mr,* distincte de *Me* et *Mf,*** qui :
  - utilise (lien "**uses**") ***Me*** et ***Mf,***
  - comprend ***Vr*** et les variables d'attributs de ***r*** (ainsi que les invariants et opérations associés).
- **une machine** incluant (lien "**includes**") ***Me, Mf*** et ***Mr.***

19-cas 2 : fixe pour *e*, non fixe pour *f* et sans attribut :

- **la variable *Vr* est dans la machine *Me*** (ainsi que les variables, invariants et opérations associés),
- ***Me*** utilise ("**uses**") ***Mf.***
- **une machine** incluant (lien "**includes**") ***Me*** et ***Mf.***

Exemple :

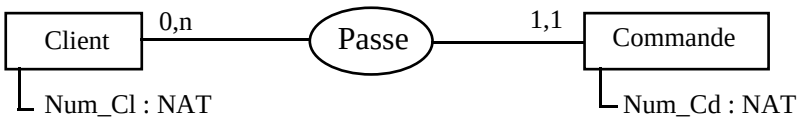


Figure 5.16. *Exemple d'association fixe*

Reprenons l'exemple de l'association passe entre l'entité Client et Commande. Notons que Passe n'a pas d'attribut, est fixe pour Commande mais pas pour Client, donc passe est dans la machine Commande :

MACHINE MA_Commande
USES MA_Client
SETS COMMANDES
VARIABLES Commandes, Num_Cd, Passe
INVARIANT Commandes $\subseteq$ COMMANDES $\wedge$
Num_Cd $\in$ Commandes $\rightarrowtail$ NAT $\wedge$
Passe^{-1} $\in$ Commandes $\rightarrow$ Client
INITIALISATION Commandes, Num_Cd, Passe := {}, {},{}
OPERATIONS
Ajout_Commande(o, n, c) =
PRE o $\in$ COMMANDES - Commandes $\wedge$ n $\in$ NAT $\wedge$ c $\in$ Clients
THEN Commandes := Commandes $\cup$ {o} ||
Num_Cd(o) := n ||
Passe := Passe $\cup$ {c $\mapsto$ o}
END ;
SuppCommande(c) =
PRE o $\in$ Commandes
THEN Commandes := Commandes - {o} ||
Num_Cd := {o} $\lhd$ Num_Cd ||
Passe := Passe $\rhd$ {o}
END
END

Remarques

- Le seul cas où la règle S19 ne peut s'appliquer est :

  r* est sans attribut, fixe pour *e* et pour *f

  dans ce cas, il n'y a pas de raison de choisir à priori entre *Me* et *Mf* pour contenir *Vr ;* nous laissons donc le choix au spécifieur ; une considération qui peut l'aider est : l'association

est-elle obligatoire pour *e* et pas pour *f* ? dans ce cas, il est préférable de mettre *Vr* dans *Me* (car la création d'instances de *e* doit être faite conjointement avec la création de liens *r*).

- Dans le cas d'une entité associative *c*, ses associations avec les entités déterminantes sont fixes pour *c*, on est donc soit dans le cas précédent soit dans le cas S19-2 .

- Pour éviter la circularité des liens *uses*, nous pouvons avoir à regrouper deux entités dans une même machine mais uniquement dans le cas de deux associations entre les deux entités, ce qui semble très rare.

- Si une association *r* est obligatoire pour *e* et que *Vr* n'est pas dans *Me*, la création d'instances de *e* seules est spécifiée dans la machine *Me* : la machine incluant *Me* et *Mr* utilise cette opération pour définir une opération créant en parallèle l'instance de *e* et ses liens. Cette machine incluante est la seule où on peut prouver que le caractère obligatoire de l'association est bien respecté.

  Exemple :
  Reprenons l'exemple précédent en supposant que chaque client doit obligatoirement avoir au moins une commande, c'est-à-dire que la cardinalité entre Client et Passe est égale à (1,n). Dans ce cas, l'association Passe est obligatoire pour Client, cependant elle est définie dans la machine Commande par le caractère fixe. Nous avons donc :

```
MACHINE          MA_Client
SETS             CLIENTS
VARIABLES        Clients, Num_Cl
INVARIANT        Clients ⊆ CLIENTS  ∧
                 Num_Cl ∈ Clients → NAT
INITIALISATION   Clients, Num_Cl := {}, {}

OPERATIONS
    Ajout_Client(c, n) =
    PRE  c ∈ CLIENTS - Clients ∧ n ∈ NAT
    THEN     Clients := Clients ∪ {c} ||
             Num_Cl(c) := n
    END
END
```

```
MACHINE          MA_Interface
INCLUDES  MA_Commande, MA_Client
OPERATIONS
      Ajout_Client_Commande(n_c, n_o) =
      PRE   n_c ∈ NAT  ∧  n_o ∈ NAT
      THEN

            ANY  c, o  WHERE        c ∈ CLIENTS - Clients  ∧
                                    o ∈  COMMANDES - Commandes
            THEN
                  Ajout_Client(c, n_c) ||
                  Ajout_Commande(o, n_o, c)
            END
      END
END
```

II.2.3. Héritage

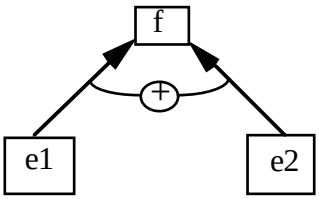
Dans la partie précédente, nous avons formalisé l'héritage très simplement par l'inclusion d'ensembles d'identités d'objet : si *e* hérite de *f* alors $Ve \subseteq Vf$. L'héritage d'opérations n'a pas d'équivalent aussi direct en B, cependant, il peut être simulé par une certaine utilisation des liens *uses* et *includes* entre machines abstraites. Il nous faut résoudre le même problème pour l'héritage que pour les associations : si une entité *e* hérite d'une entité *f*, nous avons $Ve \subseteq Vf$, mais où mettre *Ve* et *Vf* et donc les opérations associées ?

Comme pour les associations, certaines solutions ont déjà été proposées :

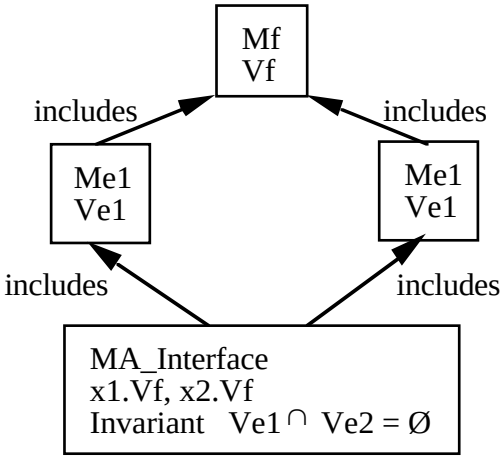
- mettre toutes les variables d'une même hiérarchie d'héritages dans la même machine [LAN 94b] : cette solution fait perdre le bénéfice de la spécification incrémentale apportée par l'héritage.

- mettre toutes les variables dans des machines différentes avec, si *e* hérite de *f*, un lien *Me includes Mf* [NAG 94] ; cette solution présente un inconvénient dans le cas où une

entité a plusieurs sous-entités directes et qu'il existe des contraintes d'intégrité entre ces sous-entités et (ou) la super-entité car ces contraintes ne peuvent être exprimées que dans une machine M qui inclut toutes les machines de la hiérarchie d'héritage.
Supposons que nous ayons le schéma suivant :



En appliquant cette solution, nous obtenons :



Comme le lien *includes* est transitif, la super-entité sera ainsi incluse plusieurs fois dans la machine MA_Interface. Ce qui entraîne une multiplication des ensembles d'instances existantes d'entités. Cette solution n'est donc pas satisfaisante.

Notre solution consiste également à conserver une machine différente par entité mais en utilisant des liens *uses* :

Règle S20
Si ***e*** **hérite de *f*:**
- **Me *uses* Mf**
- **Me, Mf** sont incluses (lien ***includes***) dans une machine **MA_Interface**
(MA_Interface inclut toutes les machines de la hiérarchie d'héritages)

Remarques

- En B, il n'y a pas de mécanisme d'héritage. Les solutions proposées permettent seulement un héritage d'attributs. Concernant l'héritage d'opérations, nous proposons, dans le cadre de ce travail, un moyen de le simuler :

  - Dans Me, nous pouvons créer des instances de e (mais pas en tant qu'instances de f) et spécialiser des instances de f comme instances de e ou déqualifier des instances de e (elles restent instances de f). Dans MA_Interface, nous pouvons utiliser ces opérations pour créer réellement des instances de e comme cas particulier d'instances de f (en appliquant en parallèle e-création et f-création).
  - C'est seulement dans MA_Interface que l'on peut prouver l'invariant (présent dans Me) Ve $\subseteq$ Vf.

  Exemple :
  Reprenons l'exemple de la MA_Personne donné dans II.1.1 et supposons qu'on ait les spécialisations : Secrétaire et Ingénieur sont des sous-entités de l'entité Personne.
  La règle S20 nous donne alors l'architecture :

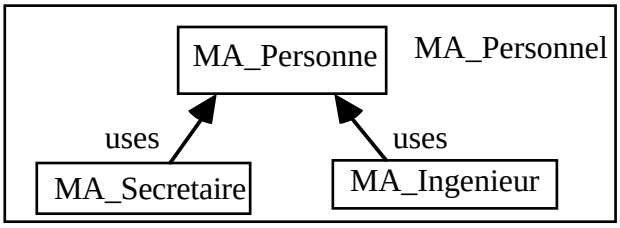


Figure 5.17. *Structure des machines associées à l'héritage entre Personne, Secrétaire et Ingénieur*

La MA_Personne (partie II.1.1) contient les caractéristiques communes aux entités Personne, Secrétaire et Ingénieur ainsi que les instances de Personne. La MA_Secrétaire contient les caractéristiques propres à l'entité Secrétaire et les instances de Secrétaire (idem pour la MA_Ingénieur). Dans la MA_Secrétaire, on ne peut ni créer ni supprimer de personne, par contre, on peut donner ou enlever à des personnes la qualification de secrétaire.

MACHINE	MA_Secrétaire
USES	MA_Personne
SETS	LANGUES = {anglais, français, ...}
VARIABLES	Secrétaires, Langues
INVARIANT	Secrétaires $\subseteq$ Personnes $\wedge$
	Langues $\in$ Secrétaires $\rightarrow \wp$ (LANGUES)

OPERATIONS
Ajout_Secrétaire(p, langues) =
PRE p ∈ PERSONNES − Secrétaires ∧ langues ⊆ LANGUES
THEN Secrétaires := Secrétaires ∪ {p} || Langues(p) := langues
END ;
Supp_Secrétaire(p) =
PRE p ∈ Secrétaires
THEN Secrétaires := Secrétaires − {p} || Langues := {p} ⩤ Langues
END
END

La MA_Personnel inclut les 3 autres MA, n'a pas de variable propre et contient les opérations qui forment l'interface de l'ensemble de la hiérarchie. Nous pouvons définir également dans cette machine la contrainte de disjonction entre Ingénieur et Secrétaire. C'est aussi dans cette machine que l'on trouvera des opérations comme la création d'une secrétaire en tant que nouvelle personne : une telle opération fait appel à la composition d'une opération propre à la MA_Personne (créer une personne) et d'une opération propre à la MA_Secrétaire (lui donner la qualification de Secrétaire). L'invariant Secrétaires ⊆ Personnes ne peut être prouvé que dans la machine incluante MA_Personnel (où Ajout_Secrétaire pour une personne nouvelle ne se fait qu'accompagné de Ajoute_Personne).

MACHINE MA_Personnel
INCLUDES MA_Personne, MA_Secrétaire, MA_Ingenieur
INVARIANT Ingenieurs ∩ Secrétaires = Ø
OPERATIONS
personne ← Arrivée(num_mat, date, nom, situation) =
PRE Personnes ⊂ PERSONNES ∧ num_mat ∈ NAT ∧ num_mat ∉ ran(Matricule) ∧
date ∈ DATE ∧ nom ∈ STRING ∧ situation ∈ STRING
THEN **ANY** p **WHERE** p ∈ PERSONNES − Personnes
THEN Ajout_Personne(p,num_mat, date, nom, situation) || personne := p
END
END ;
Qualif_Secretaire (p, langues) =
PRE p ∈ Personnes - Secrétaires ∧
langues ⊆ LANGUES
THEN Ajout_Secrétaire (p, langues)
END ;

```
        sec ← Arrivée_Secrétaire(num_mat, date, nom, situation,langues) =
        PRE   Personnes ⊂ PERSONNES ∧ num_mat ∈ NAT ∧ num_mat ∉ ran(Matricule)
∧
                date ∈ DATE ∧ nom ∈ STRING ∧ situation ∈ STRING
                ∧ langues ⊆ LANGUES
        THEN        ANY p WHERE p ∈ PERSONNES − Personnes
                    THEN        Ajout_Personne(p,num_mat, date, nom, situation) ||
                                Ajout_Secrétaire(p, langues) || sec := p
                    END
        END ;
        Départ_Personne(p) =
        PRE         p ∈ Personnes
        THEN        IF p ∈ Secrétaires THEN
                        Supp_Secretaire(p)
                    END ||
                    Supp_Personne(p)
        END
END
```

II.3. Génération des opérations de base

Dans un système d'information orienté gestion de données, on utilise très souvent des opérations appelées opérations de base, associées aux entités et associations, qui permettent l'ajout, la suppression d'instances d'entité ou d'association et la modification de valeur d'attribut. Nous avons étudié la possibilité de générer automatiquement ces opérations en fonction des contraintes de cardinalité exprimées dans le diagramme d'objets, contrairement à [HAD 96] où ce sont les contraintes inter-associations qui ont été prises en compte (uniquement pour l'ajout de lien d'association). Ces opérations pourront alors être utilisées dans les opérations "visibles aux utilisateurs", correspondant aux scénarios du modèle dynamique et définies dans la machine Interface (cf. partie III).

II.3.1. Principes généraux

Les opérations de base concernant une entité vont être définies dans la machine de cette entité car c'est là où les variables peuvent être modifiées directement. De même, les opérations concernant une association vont être spécifiées dans la machine où est définie l'association.

En B, chaque opération décrit son effet local sur les variables de la machine où elle est définie et non pas sur les variables représentant toutes les données du système. Concernant les obligations de preuve, les opérations de chaque machine doivent vérifier les invariants qui sont décrits en utilisant les variables de la machine elle-même. C'est pourquoi, les invariants sur les liens d'association ne peuvent être prouvés que dans la machine Interface qui inclut toutes les machines du système. En effet, en appliquant la règle S19, définie dans la partie II.2.2, si une association r est définie entre les entités e et f, sa variable Vr associée est soit spécifiée dans la machine Me ou Mf (figure 5.18.a), soit spécifiée dans une machine propre Mr (figure 5.18.b).

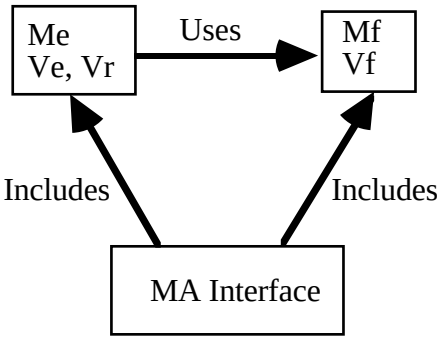


Figure 5.18.a. *Vr est définie dans Me*

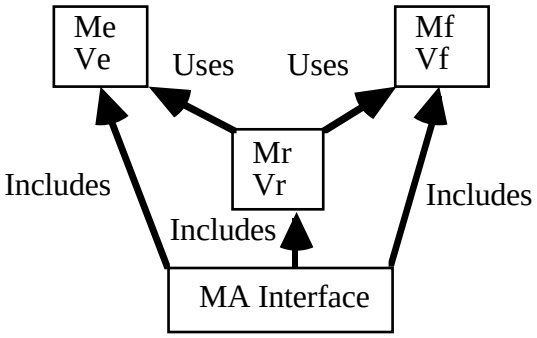


Figure 5.18.b. *Vr est définie dans Mr*

Dans les deux cas, l'invariant le plus général est :

$$Vr \in Ve \leftrightarrow Vf$$

Comme Ve est définie dans Me et Vf dans Mf, les obligations de preuve correspondant à cet invariant ne peuvent être vérifiées ni dans Me (cas a) ni dans Mr (cas b) mais seront démontrées seulement dans la machine Interface.

Notre but est de spécifier des machines entités et associations (appelées machines de base) avec les opérations de base qui respectent les invariants correspondant aux cardinalités du diagramme d'objets même si certaines preuves ne peuvent être faites que dans la machine Interface. Cela permet de :

- mieux séparer la vérification des contraintes concernant chaque modèle et
- aider à la spécification des opérations de la machine Interface.

Afin d'atteindre cet objectif, nous avons choisi de renforcer les pré-conditions des opérations de base des machines contenant des invariants de type $Vr \in Ve \leftrightarrow Vf$.

Examinons le cas où $Vr \in Ve \rightarrow Vf$. L'opération d'ajout d'un nouveau lien dans Vr, noté Ajout_Lien(ie, if), peut se traduire dans la machine Mr (cas b) des deux manières suivantes :

(i) Ajout_Lien(ie, if) =
BEGIN
Vr := Vr ∪ {ie ↦ if}
END

(ii) Ajout_Lien(ie, if) =
PRE ie ∈ Ve - dom(Vr) ∧
if ∈ Vf
THEN
Vr := Vr ∪ {ie ↦ if}
END

L'opération spécifiée dans le cas (i) est très générale et valable pour une relation quelconque. En revanche, dans le cas (ii), nous avons pris en compte le fait que Vr est une fonction totale en spécifiant dans la pré-condition que ie doit appartenir à l'ensemble Ve moins le domaine de Vr, et if à l'ensemble Vf.

Au niveau de la machine Interface, une opération OP faisant appel à l'opération Ajout_Lien de Mr doit être spécifiée comme suit pour que l'invariant $Vr \in Ve \rightarrow Vf$ soit vérifié :

OP(ie, if) =
PRE Cond ∧ /* Conditions sur les contraintes dynamiques */
ie ∈ Ve - dom(Vr) ∧
if ∈ Vf
THEN Ajout_Lien(ie, if) ||
. . .
END

Nous remarquons qu'il est plus facile de décrire les pré-conditions (concernant les contraintes du modèle d'objets) de l'opération OP à partir de l'opération Ajout_Lien du cas (ii) car ces pré-conditions sont déjà spécifiées dans l'opération de base. Dans le cas (i), c'est au concepteur de trouver lui-même la pré-condition.

Notons que les pré-conditions proposées dans les opérations de base sont les conditions minimales liées aux contraintes du modèle d'objets (contraintes de cardinalité). Si d'autres types de contraintes d'intégrité sont spécifiés, on peut ajouter d'autres pré-conditions.

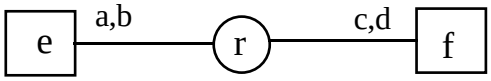
Nous allons préciser maintenant chaque type d'opération.

II.3.2. Opérations sur les entités

II.3.2.1. Opération d'ajout d'instances d'entité

Une telle opération consiste à ajouter un nouvel objet dans l'ensemble des objets existants de l'entité et à affecter une valeur pour chaque caractéristique de l'objet. Une caractéristique peut être un attribut ou un lien d'association.

Soient les deux entités e, f et l'association r :



Notons que Me est la machine où est définie l'entité e.

Nous avons deux cas :

a. L'association r n'est pas définie dans la machine Me

Machine	Me
Sets	Se
Variables	Ve, Vai (i=1 ..n)
Invariant	$Ve \subseteq Se \wedge Vai \in Ve \rightarrow Tai$

Dans ce cas, Vr n'est pas définie dans Me, aucune pré-condition supplémentaire n'est à envisager.

L'opération d'ajout d'une instance de *e*, notée Ajout_e, doit respecter les invariants de Me. Elle doit avoir comme paramètre un nouvel objet de l'entité à ajouter (noté *ie*) et les valeurs val_ai au moins des attributs obligatoires ai (de type Tai) de cette entité. Nous avons donc :

Ajout_e(ie, val_a1, ..., val_an) =
PRE (1) $ie \in Se - Ve \wedge$
(2) $val\_ai \in Tai \quad (i=1..n)$
THEN $Ve := Ve \cup ie \parallel$
$Vai := Vai \cup \{ie \mapsto val\_ai\} \quad (i=1..n)$

END

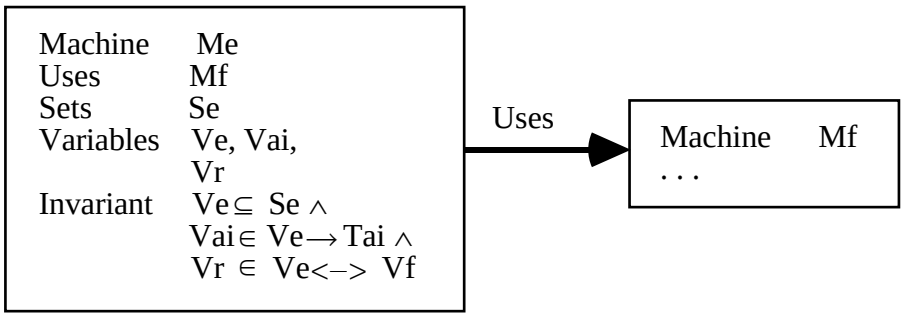
Notons que la pré-condition de cette opération vérifie (1) que l'objet ie est nouveau et que l'on peut l'ajouter et (2) le typage des valeurs des attributs.

Exemple :
L'opération Ajout_Personne de la machine MA_Personne (cf. partie II.1.1) est spécifiée comme suit :

Ajout_Personne(p, num_mat, nom, date, situation) =
PRE $p \in$ PERSONNES - Personnes $\wedge$
$num\_mat \in$ NAT $\wedge$ $num\_mat \notin ran(Matricule)$
$nom \in$ STRING $\wedge$
$date \in$ DATE $\wedge$
$situation \in$ STRING $\wedge$
THEN Personnes := Personnes $\cup$ {p} ||
Matricule := Matricule $\cup$ {p $\mapsto$ num_mat} ||
Nom := Nom $\cup$ {p $\mapsto$ nom} ||
Date_Naissance := Date_Naissance $\cup$ {p $\mapsto$ date} ||
Situation := Situation $\cup$ {p $\mapsto$ situation}
END

b. L'association r est définie dans la machine Me

Dans ce cas, l'association r est fixe pour e (cf. Règle S19, partie II.2.2).



Outre les paramètres, la pré-condition et la substitution qui concernent l'objet ie et la valeur val_ai des attributs obligatoires Vai (i=1 ..n), nous devons éventuellement, à la création de ie, établir le lien entre l'objet ie et autre(s) objet(s) de f selon la cardinalité de r car l'association r est définie dans la machine Me. Nous avons deux cas selon la cardinalité minimale a entre e et r :

- **a=1**

  À la création de ie, nous devons l'associer à un seul objet, noté if (b=1), ou à un ensemble d'objets (b=n), noté If, de l'entité f.

  L'opération Ajout_e va avoir comme paramètres :

  Ajout_e(ie, val_ai, if) (cardinalité (1,1))

  Ajout_e(ie, val_ai, If) (cardinalité (1,n))

- **a=0**

  À la création de ie, nous voulons éventuellement l'associer à un objet if, ou un ensemble d'objets If de l'entité f.

  Les paramètres de l'opération Ajout_e sont :

  Ajout_e(ie, val_ai [, if]) (cardinalité (0,1))

  Ajout_e(ie, val_ai [, If]) (cardinalité (0,n))

  Notons que la partie entre crochets est optionnelle.

Concernant la pré-condition de ces opérations, outre les vérifications associées au nouvel objet à ajouter et le typage des valeurs de ses attributs, nous devons vérifier les conditions concernant If (ou if).

Le tableau ci-dessous résume les pré-conditions et substitutions selon les différents cas de cardinalité :

<table>
<tr><th rowspan="2">Partie
Contrainte de cardinalité</th><th colspan="2">PRE</th><th colspan="2">THEN</th></tr>
<tr><th>Sur If ou if</th><th>Sur ie, val_ai</th><th>Sur Ve, Vai</th><th>Sur Vr</th></tr>
<tr><td>(a,n) (1,1)</td><td>If $\subseteq$ Sf - Vf</td><td rowspan="8">ie $\in$ Se - Ve $\wedge$
val_ai $\in$ Tai (i=1 ..n)</td><td rowspan="8">Ve := Ve $\cup$ {ie} $\|$
Vai := Vai $\cup$ {ie $\mapsto$ val_ai} (i=1..n)</td><td rowspan="4">Vr := Vr $\cup$ ie * If</td></tr>
<tr><td>(a,n) (0,1)</td><td>If $\not\subset$ ran(Vf)</td></tr>
<tr><td>(a,n) (1,n)</td><td rowspan="2">If $\subseteq$ Sf</td></tr>
<tr><td>(a,n) (0,n)</td></tr>
<tr><td>(a,1) (1,1)</td><td>if $\in$ Sf - Vf</td><td rowspan="4">Vr := Vr $\cup$ {ie $\mapsto$ if}</td></tr>
<tr><td>(a,1) (0,1)</td><td>if $\notin$ ran(Vr)</td></tr>
<tr><td>(a,1) (1,n)</td><td rowspan="2">if $\in$ Sf</td></tr>
<tr><td>(a,1) (0,n)</td></tr>
</table>

Nous allons seulement expliquer deux cas :

- *Cardinalités (1,n) entre e, r et (1,1) entre f et r*

  L'opération d'ajout d'une instance de e aura comme paramètre : Ajout_e(ie, val_ai, If)
  Entre f et r, nous avons la contrainte (1,1). Elle signifie qu'un objet de f doit être lié à un objet de e et cela doit être fait en même temps que la création de l'objet de f car les cardinalités minimale et maximale sont égales à 1. Donc, l'ensemble If ne doit pas déjà exister. La vérification sur If sera donc : If $\subseteq$ Sf - Vf.

- *Cardinalités (1,n) entre e, r et (0,1) entre f, r*

  Les paramètres de l'opération Ajout_e sont également : Ajout_e(ie, val_ai, If)
  La contrainte de cardinalité (0,1) entre f et r exprime qu'un objet de f peut être lié à un seul objet de e. Si l'association entre ie et If est faite à la création de If, nous avons :

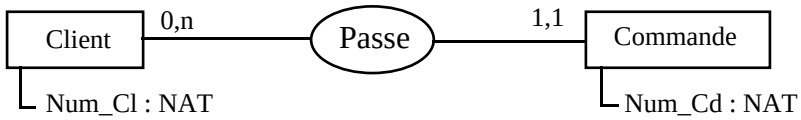
  (1) If $\subseteq$ Sf - Vf

  Dans le cas contraire : (2) If $\subseteq$ Vf - ran(Vf)

  (1), (2) => If $\subseteq$ Sf - ran(Vf)

  Nous avons donc : If $\not\subset$ ran(Vf)

Exemple :



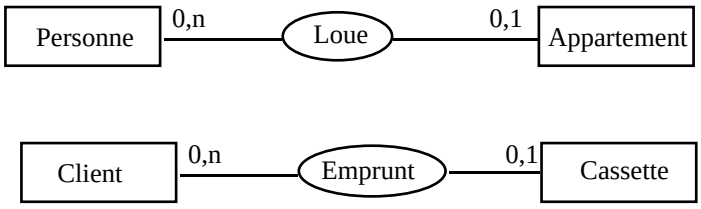
Dans cet exemple, l'association Passe va être définie dans la machine de Commande car elle est fixe pour cette entité (cf. Règle S19, partie II.2.2). L'opération Ajout_Commande est spécifiée de la manière suivante :

Ajout_Commande(cd, num_cd, cl) =
PRE cd ∈ COMMANDES - Commandes ∧
num_cd ∈ NAT ∧
cl ∈ CLIENTS /* car l'ajout d'une commande peut être faite en même temps que la création du client associé */
THEN Commandes := Commandes ∪ {cd} ||
Num_Cd := Num_Cd ∪ {cd ↦ num_cd} ||
Passe := Passe ∪ {cl ↦ cd}
END

II.3.2.2. Opérations de suppression d'instances d'entité

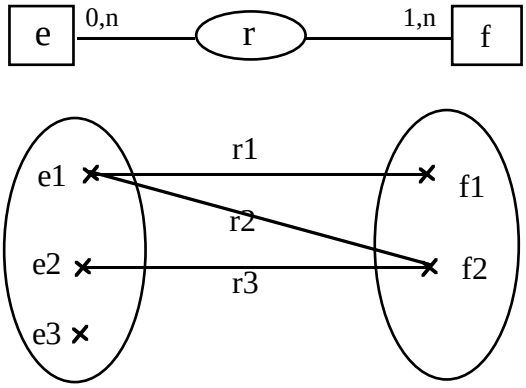
Contrairement aux opérations d'ajout qui peuvent être déduites automatiquement à partir du diagramme d'objets, les opérations de suppression posent le problème suivant. Considérons les deux cas ci-dessous.

Cas 1



Remarquons que, dans ces deux diagrammes, les cardinalités sont identiques. Cependant, leur sémantique est totalement différente. En effet, dans le premier diagramme, la suppression d'une personne peut entraîner la suppression du lien Loue mais ceci n'entraîne pas la suppression de son appartement ! Ce dernier peut rester sans locataire. À l'inverse, dans le deuxième diagramme, si on supprime un client qui garde une cassette (chez lui), on ne peut plus récupérer la cassette, il faut donc la supprimer.

Cas 2



Considérons le diagramme d'objets ci-dessus et le schéma représentant les liens entre e et f à un moment donné. Nous remarquons que la suppression de l'objet e2 n'entraîne pas d'incohérence dans le système puisque l'objet f2 est toujours lié à e1. Cependant, la suppression de e1 pose un problème car f1 ne sera plus lié à aucun objet de e.

Afin de résoudre ce problème, nous avons plusieurs possibilités :

- Soit interdire de supprimer e1,
- Soit supprimer e1 et aussi f1,
- Soit supprimer e1 et lier f1 à un autre objet de e.

Dans les cas 1 et 2, il n'y a pas d'information suffisante dans le diagramme d'objets pour choisir entre les diverses possibilités. Cette information ne sera donnée que dans les diagrammes E/T dont la traduction sera abordée dans la partie III de ce chapitre.

Les opérations de suppression ne peuvent donc pas être générées simplement à partir du diagramme d'objets car ce dernier ne comporte pas assez d'information sur le cycle de vie d'un objet. Dans ce travail, nous proposons les opérations de suppression de base qui respectent simplement l'invariant de la machine concernée. Ces opérations ne sont pas utilisables directement par l'utilisateur. Elles ne peuvent être appelées que par les opérations de la machine Interface, qui devront elles vérifier l'invariant global.

Dans cette partie, nous proposons les opérations de suppression d'instances d'entité qui respectent simplement l'invariant de la machine entité concernée (donc pas nécessairement l'invariant global).

Souvent dans les applications sur les systèmes d'information, il arrive que l'on veuille supprimer un ensemble d'objets ayant les mêmes valeurs de certaines caractéristiques : par exemple, la suppression de toutes les cassettes d'un film. Ce qui est rarement le cas pour les opérations de création qui ajoute en général un seul nouvel objet à la fois. Donc, contrairement aux opérations d'ajout qui agissent sur un seul objet, les opérations de suppression peuvent

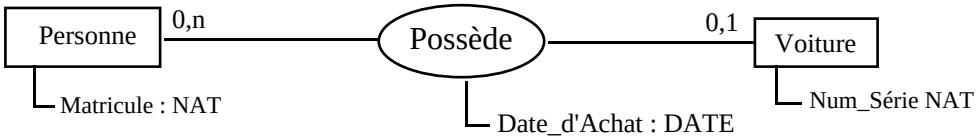
avoir comme paramètre un ensemble d'objets à supprimer. Nous avons également deux cas à envisager :

a. L'association r n'est pas définie dans la machine Me

L'opération de suppression consiste à enlever l'ensemble *Ens_e* (qui apparaît en tant que paramètre de l'opération) à l'ensemble *Ve* ainsi qu'à l'ensemble de tous les attributs. Nous avons :

```
Sup_e(Ens_e) =
PRE       Ens_e ⊆ Ve
THEN      Ve := Ve - Ens_e ||
          Vai := Ens_e ⩤ Vai    (i=1..n)
END
```

Exemple :



L'opération de suppression de toutes les voitures Ens_V d'une personne p (Ens_V= Possède[{p}]) est spécifiée comme suit :

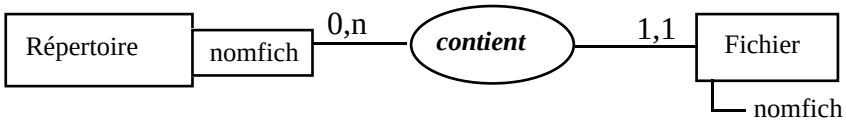
```
Sup_Voiture(Ens_V) =
PRE       Ens_V ⊆ Voitures
THEN      Voitures := Voitures - Ens_V ||
          Num_Série  := Ens_V ⩤ Num_Série
END
```

b. L'association r est définie dans la machine Me

Dans ce cas, l'opération de suppression de l'ensemble *Ens_e* doit supprimer également les liens d*e*r qui concernent les objets de *Ens_e*. Nous avons donc :

```
Sup_e(Ens_e) =
PRE       Ens_e ⊆ Ve
THEN      Ve := Ve - Ens_e ||
          Vai := Ens_e ⩤ Vai ||          (i=1..n)
          Vr := Ens_e ⩤ Vr
END
```

Exemple :



Rappelons que l'association Contient est définie dans la machine MA_Fichier :

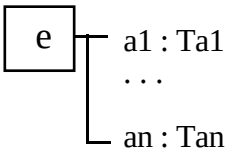
$$Contient^{-1} \in Fichiers \rightarrow Répertoires$$

L'opération de suppression de tous les fichiers Ens_F d'un répertoire r (Ens_F = Contient[{r}]) est spécifiée de la manière suivante :

Sup_Fichier(Ens_F) =
PRE Ens_F $\subseteq$ Fichiers
THEN Fichiers := Fichiers - Ens_F ||
Nomfich := Ens_F $\lhd$ Nomfich ||
Contient := Contient $\rhd$ Ens_F
END

II.3.2.3. Opérations de modification de valeurs d'attribut

Étant donné l'entité *e* :



Une opération de modification de la valeur d'un attribut *ai* de l'entité *e* consiste à changer l'ancienne valeur de l'attribut *ai* d'un objet *ie.* Elle est spécifiée comme suit :

Modif_ai(ie, val_ai) =
PRE ie $\in$ Ve $\wedge$
val_ai $\in$ Tai
THEN Vai(ie) := val_ai
END

Remarque

Une telle opération peut être générée systématiquement pour tous les attributs d'une entité. Cependant, cela semble inutile car, parmi les attributs, certains ne change jamais de valeur pendant l'exploitation de la base de données.

Exemple :
Considérons l'entité Employé du schéma suivant :

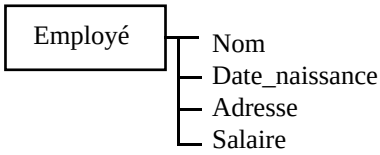
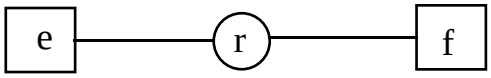


Figure 5.19. *Exemple d'entité Employé*

Remarquons qu'un employé ne change jamais son nom, sa date de naissance, il est inutile d'avoir les deux opérations de modification de ces deux attributs. Seuls les attributs Adresse et Salaire peuvent changer de valeur. Nous allons donc déduire les opérations uniquement pour les attributs modifiables.

II.3.3. Opérations sur les associations

Considérons le diagramme objet suivant :



Rappelons qu'une association r, entre les entités e et f, possède des opérations si elle est définie dans une machine propre. Autrement dit, r est non fixe pour aucune entité ou avec attribut(s) modifiable(s) (cf. chapitre 3, partie II.2.3). Donc, une association associée à une machine propre peut avoir deux types d'opération :

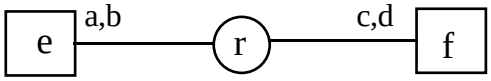
- les opérations sur les liens si elle est non-fixe pour e et pour f,
- les opérations de modification sur ses attributs s'ils sont modifiables.

II.3.3.1. Opérations sur les liens

a. Opération d'ajout d'un lien

Une telle opération, noté Ajout_Lien(ie, if), consiste à ajouter un nouveau lien (ie, if) pour l'association r. Les conditions sur ie, if dépendent fortement des contraintes de cardinalité car ces contraintes expriment la manière dont les objets des entités connectées sont liés.

Étant donné le diagramme suivant :



Notons que les deux entités e et f sont symétriques par rapport à l'association r. Nous allons donc examiner seulement les différents cas de contrainte de cardinalité (a,b) entre e et r.

(1) *(a,b) = (0,n)*

Nous avons ici la contrainte la plus large. L'ajout de nouveaux liens (ie, if) dans r peut être faite soit à la création de ie, nous avons alors la contrainte : $ie \in Se - Ve$. Soit elle peut agir sur un objet ie déjà existant (cardinalité maximale égale n) : $ie \in Ve$.

Nous avons donc : $ie \in Se - Ve \vee ie \in Ve$

=> **$ie \in Se$**

(2) *(a,b) = (0,1)*

Comme le cas précédent, l'ajout du lien (ie, if) peut être fait à la création de l'objet ie : $ie \in Se- Ve$. Ou si ie a déjà été créé, il faut vérifier qu'il n'est lié à aucun objet de l'entité f : $ie \in (Ve - dom(Vr))$ (car la cardinalité maximale est égale à un).

Nous avons : $ie \in Se- Ve \vee ie \in (Ve - dom(Vr))$

=> **$ie \in Se - dom(Vr)$**

(3) *(a,b) = (1,n)*

La cardinalité minimale égale à un exprime qu'à la création d'un objet ie de e, il faut l'associer à un objet de f. C'est à dire, l'opération d'ajout de nouveau lien (ie, if) peut être appelée à la création de ie, nous avons alors la contrainte : $ie \in Se - Ve$. Elle peut aussi être utilisée pour lier l'objet ie (déjà créé) à un autre objet if' de f. Dans ce cas, ie doit faire partie du domaine de Vr car il doit être déjà lié à un objet if : $ie \in dom(Vr)$.

Nous avons : $ie \in Se - Ve \vee ie \in dom(Vr)$

Rappelons que dom(Vr) = Ve car a=1

=> **$ie \in Se$**

(4) *(a,b) = (1,1)*

Dans ce cas, l'opération d'ajout de nouveau lien (ie, if) doit être faite en même temps que la création de ie. Nous avons donc :

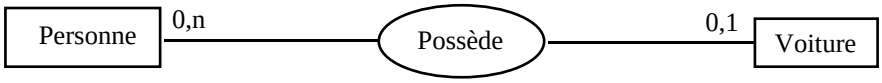
ie ∈ Se - Ve

Dans le tableau ci-dessous, nous résumons les pré-conditions et la substitution pour les quatre cas de contrainte de cardinalité :

Lien e_r		Lien f_r		THEN
(a,b)	PRE	(c,d)	PRE	
(0,n) (1,n)	ie ∈ Se	(0,n) (1,n)	if ∈ Sf	Vr := Vr ∪ {ie↦ if}
(0,1)	ie ∈ Se - dom(Vr)	(0,1)	if ∈ Sf - dom(Vr)	
(1,1)	ie∈ Se - Ve	(1,1)	if ∈ Sf - Vf	

Exemple :

Étant donné le diagramme ci-dessous :



L'opération Ajout_Possède(p, v) est spécifiée comme suivante :

Ajout_Possède(p,v) =
PRE p ∈ PERSONNES ∧
v ∉ ran((Possède)
THEN Possède := Possède ∪ {p ↦ v}
END

b. Opération de suppression de liens

Cette partie propose différents types d'opérations de suppression de liens. Rappellons que ces opérations ne sont pas utilisables directement par l'utilisateur mais vont être appelées dans les opérations de la machine Interface.

En fonction de la contrainte de cardinalité entre les deux entités e, f et l'association r, nous pouvons avoir différentes opérations de suppression de liens.

La première, noté Sup_Lien_e(Ens_e), consiste à supprimer tous les liens de r liés à l'ensemble Ens_e des objets de l'entité e. Elle est spécifiée comme ci-dessous :

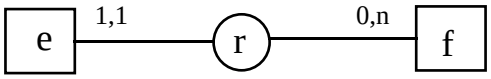
```
Sup_Lien_e(Ens_e) =
PRE     Ens_e ⊆ Ve
THEN    Vr := Ens_e ⩤ Vr
END
```

La deuxième, noté Sup_Lien_f(Ens_f), supprime tous les liens de r liés aux objets de Ens_f :

```
Sup_Lien_f(Ens_f) =
PRE     Ens_f ⊆ Vf
THEN    Vr := Ens_f ⩥ Vr
END
```

Ces deux types d'opération sont définis quelque soient les contraintes de cardinalité.

Pour les associations ayant la cardinalité maximale égale à 1 au moins d'un côté, on se ramène à une des deux opérations précédentes. En effet, supposons que nous ayons le diagramme suivant :



À partir d'un objet ie de e, nous pouvons déduire l'objet if de f lié à l'objet ie de manière suivante : if = Vr(ie). Donc, l'opération Sup_Lien(ie,Vr(ie)) est équivalente à l'opération Sup_Lien_e({ie}).

La troisième, noté Sup_Lien(ie, if), supprime un seul lien déterminé par deux objets (ie, if) dans les cas de contrainte de cardinalité multiple (cardinalité maximale égale n) des deux côtés e et f.

```
Sup_Lien_f(ie, if) =
```

PRE	(ie, if) ∈ Vr
THEN	Vr := Vr – {ie ↦ if}
END	

Exemple :

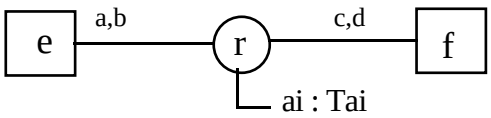
Reprenons l'exemple de l'association Possède entre l'entité Personne et Voiture . Pour cette association, nous spécifions deux opérations de suppression de lien suivantes :

(i) Sup_Lien_Personne(Ens_P) =
PRE Ens_P ⊆ Personnes
THEN Possède := Ens_P ⩤ Possède
END

(ii) Sup_Lien_Voiture(Ens_V) =
PRE Ens_V ⊆ Voitures
THEN Possède := Possède ⩥ Ens_V
END

II.3.4. Opérations sur les attributs des entités et des associations

Pour les attributs d'une association, les opérations de modification de valeurs d'attribut sont déduites de la même manière que pour ceux des entités. La seule différence vient du fait que l'on peut identifier l'instance de l'association à modifier de plusieurs manières, selon les cardinalités. Il faut donc passer en paramètre les instances de e et/ou f qui identifient cette isntance. Soit le diagramme suivant :



Exemple :
Dans le cas où la cardinalité maximale est égale à n des deux côtés, c'est-à-dire que b=n ∧ d=n, afin de déterminer un lien de l'association r, il nous faut connaître un objet ie de l'entité e et également celui de l'entité f, noté if.

L'opération de modification de valeur d'un attribut ai doit avoir comme paramètres la nouvelle valeur de ai, notée val_ai, ainsi que le couple (ie, if). Dans la pré-condition, il faut vérifier que le couple (ie, if) appartient à Vr. Cette opération est spécifiée comme suit :

```
Modif_ai(ie, if, val_ai) =
PRE     (ie, if) ∈ Vr ∧
        val_ai ∈ Tai
THEN    Vai(ie, if) := val_ai
END
```

III. Traduction du modèle dynamique

Le point de départ de la traduction du modèle dynamique est constitué non seulement des diagramme E/T, des scénarios mais aussi de la spécification B de la statique (obtenue du modèle d'objets) et de la spécification de certaines opérations de base qui sont déduites automatiquement à partir du modèle d'objets. La spécification B du modèle d'objets aide les utilisateurs à spécifier en B les expressions, sur les diagrammes E/T, décrites en langue naturelle et à formaliser les états. Quant à la spécification B des opérations de base, elle les aide à définir les actions. La traduction du modèle dynamique se réalise en deux phases.

La première phase consiste à générer des squelettes d'opérations B à partir des diagrammes du modèle dynamique.

Rappelons que chaque diagramme E/T concerne la dynamique d'une entité unique. Si on traduit chaque opération associée à une transition par une opération B, celle-ci ne décrira le changement sur les variables que d'une seule entité. Or en réalité, chaque demande d'utilisateur a un effet sur plusieurs entités. On ne pourra donc pas avoir un tout cohérent. Afin de respecter les invariants qui caractérisent les états cohérents du système, l'opération B doit correspondre aux opérations de l'utilisateur et non pas aux opérations locales sur les transitions. En fait, un scénario décrit une séquence d'événements qui débute par une demande extérieure (événement d'entrée). Il correspond donc bien à une opération de l'utilisateur.

L'idée principale est de modéliser *chaque état par un prédicat* et de traduire *chaque scénario par une opération B* qui décrit le changement d'état des objets courants de chaque

diagramme E/T concernés par le scénario. Plus précisément, cette opération spécifie formellement l'effet, sur les données du système, de l'événement d'entrée associé au scénario et porte donc le nom de cet événement. Cet effet est le changement de valeurs des variables d'une entité ou de plusieurs entités. Les diagrammes états/transitions nous aident à identifier les paramètres, les pré-conditions (condition de déclenchement et typage de l'objet courant) et la structuration du corps de ces opérations. Ils aident également à déterminer les obligations de preuves supplémentaires autour des opérations. Ces obligations de preuves sont des propriétés que doit vérifier le système afin de respecter la cohérence entre les divers éléments de la spécification formelle générée.

La deuxième phase concerne la répartition des opérations générées dans les machines obtenues à la première étape. Puis le concepteur complète les machines obtenues par le corps des opérations et les invariants décrivant les contraintes d'intégrité non exprimables graphiquement.

Les étapes générales sont donc :

- La traduction du modèle d'objets en spécification B,
- La génération des opérations de base à partir du diagramme d'objets,
- L'utilisation des notations B dans les diagrammes E/T,
- La formalisation des états,
- La génération de squelettes d'opérations B et d'obligations de preuve à partir des scénarios, diagrammes E/T,
- La répartition des opérations générées dans les machines,
- L'achèvement des machines obtenues par le corps des opérations et les invariants correspondant aux contraintes d'intégrité non exprimables graphiquement.

III.1. Utilisation des notations B dans les diagrammes états/transitions

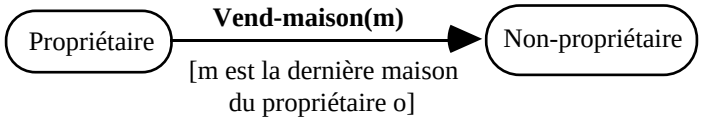
Si on se réfère à la présentation des diagrammes E/T (cf. chapitre 3), tout est exprimé en langue naturelle : les conditions de déclenchement sur les transitions, les critères de choix d'un ensemble d'objets dans les diagrammes associés à l'administrateur, les actions... Si on veut obtenir une spécification B complète, il faut remplacer ces notations en langue naturelle par des notations B en s'aidant des résultats des deux premières étapes. Ainsi on peut :

- préciser les paramètres des événements,

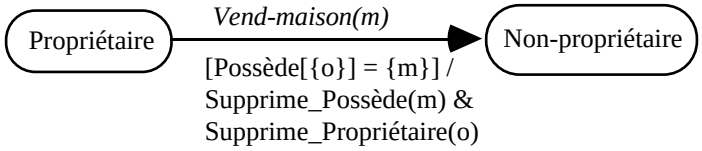
- donner une notation B aux différentes conditions,
- définir les actions comme des opérations de base des machines entités, associations.

Cette modification ne peut être faite que par le concepteur.

Exemple :
Reprenons l'exemple présenté dans le chapitre 3 :



Ce diagramme E/T sera complété par les notations B comme suit :



Nous supposons que dans les parties suivantes, les diagrammes E/T sont maintenant décrits par des notations formelles.

III.2. Formalisation d'un état

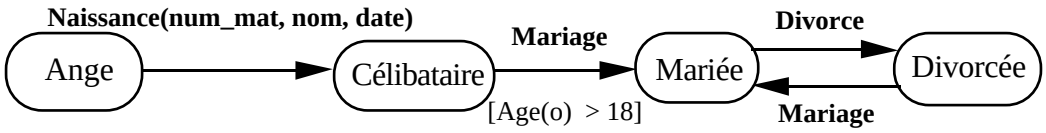
Règle D1

Chaque état État_i du diagramme E/T est spécifié en B par un prédicat Pred_État_i sur les variables et ensembles décrivant cet état. Le prédicat est paramètré par l'objet courant o. Il est défini dans la partie DEFINITION. Il existe deux cas :

- Si chaque état peut être exprimé par les attributs et/ou les liens d'association, le prédicat porte alors sur les variables correspondant à ces attributs et/ou ces liens.
- Sinon, une variable doit être ajoutée. Cette variable est une fonction totale de l'ensemble des instances existantes vers l'ensemble des états. Le prédicat sera alors décrit à l'aide de cette variable.

Exemple :

1.



Reprenons le diagramme E/T concernant l'entité Personne. Il s'agit en fait de définir les quatre états Ange, Célibataire, Mariée, Divorcée à partir de la variable formalisant l'attribut Situation (de type : Situation ∈ Personnes → {célibataire, mariée, divorcée}).

Nous obtenons donc les quatre prédicats suivants :

Ange(o) ≙ o ∈ PERSONNES - Personnes ;
Célibataire(o) ≙ o ∈ Situation^{-1}[{célibataire}] ;
Mariée(o) ≙ o ∈ Situation^{-1}[{mariée}] ;
Divorcée(o) ≙ o ∈ Situation^{-1}[{divorcée}]

2.

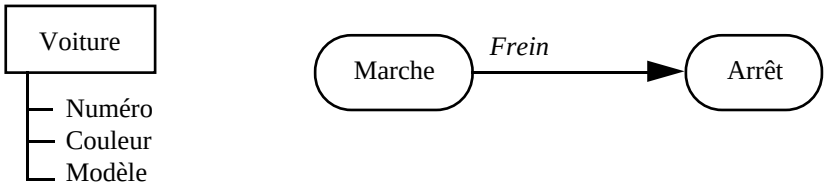


Figure 5.20. *Exemple de l'entité Voiture et son diagramme E/T*

Dans cet exemple, il n'y a pas d'attribut permettant de spécifier les états Marche et Arrêt. Une nouvelle variable État_Voiture est ajoutée qui respecte l'invariant :

État_Voiture ∈ Voitures → {Marche, Arrêt}

Dans la partie DÉFINITION, nous avons donc :

Marche(o) ≙ o ∈ État_Voiture^{-1}[{Marche}] ;
Arrêt(o) ≙ o ∈ État_Voiture^{-1}[{Arrêt}]

Remarques

- Nous distinguons ces deux cas car le premier permet de ne pas augmenter le nombre de variables. De plus, le changement d'état est fait automatiquement quand le prédicat de l'état n'est plus valide. Il n'est donc pas nécessaire d'associer à chaque transition une opération si celle-ci n'est pas décrite par le concepteur. La spécification finale est donc plus concise et les obligations de preuve sont moins nombreuses.

- Dans le cas où une entité ne contient qu'une seule instance, la notion d'objet courant du diagramme E/T n'existe plus. Chaque prédicat décrivant un état ne possède donc plus comme paramètre l'objet o.
  Prenons l'exemple "Level Crossing" présenté dans [DUP 98].

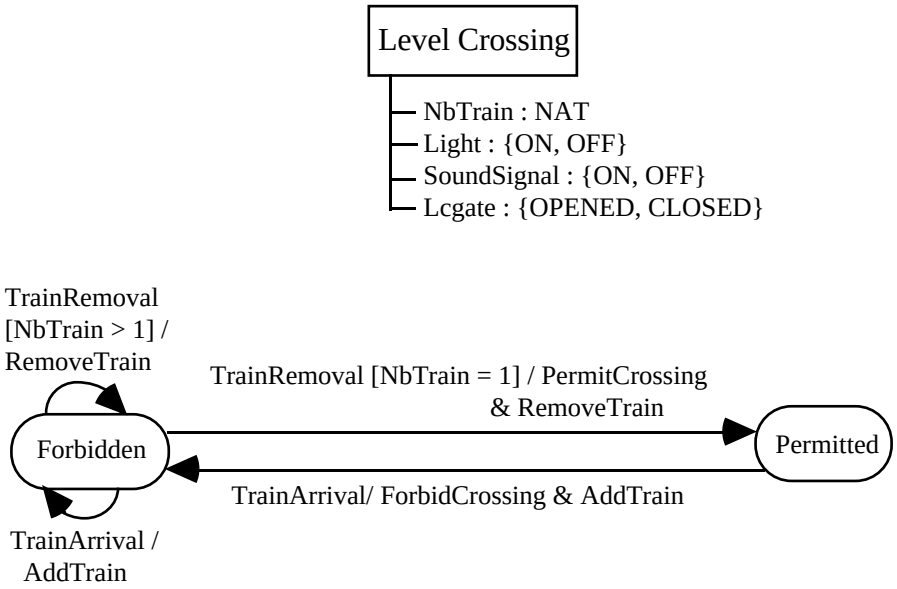


Figure 5.21. *Exemple d'entité avec une seule instance (tiré de [DUP 98])*

Les deux états Permitted et Forbidden sont spécifiés de la manière suivante :

Permitted $\widehat{=}$ Light = OFF $\wedge$ Lcgate = OPENDE $\wedge$ SoundSignal = OFF

Forbidden $\widehat{=}$ Light = ON $\vee$ Lcgate = CLOSED $\vee$ SoundSignal = ON

III.3. Génération de squelettes d'opérations B et d'obligations de preuve à partir des diagrammes E/T et scénarios

Nous allons nous intéresser maintenant à la génération de squelettes d'opérations B à partir de différents types de scénario. Chacun de ces types nous amène à une constitution différente du corps de l'opération ainsi qu'une obligation de preuve associés. Dans cette partie, nous allons expliciter ces constitutions en fonction des classements des scénarios. Pour chaque type de scénario, sont décrites deux règles qui concernent, pour la première, la génération de squelettes de l'opération (noté D + numéro + "_O"), pour la deuxième, l'obligation de preuve associée (noté D + numéro + "_P").

Dans la suite, nous adoptons la convention suivante pour la présentation des règles : nous mettons en italique les parties qui ne sont pas générées automatiquement.

III.3.1. Scénario mono-entité correspondant à une seule transition avec une seule action

Dans le diagramme E/T, chaque événement possède implicitement comme paramètre l'objet (dit courant) sur lequel va s'appliquer l'opération associée. Pour que l'opération soit applicable sur cet objet, les conditions de transition doivent être satisfaites. Ces conditions se composent du prédicat de l'état initial de la transition, de la vérification de typage, de la condition de déclenchement et d'autres conditions qui ne sont pas décrites sur le diagramme, comme par exemple les conditions de bord.

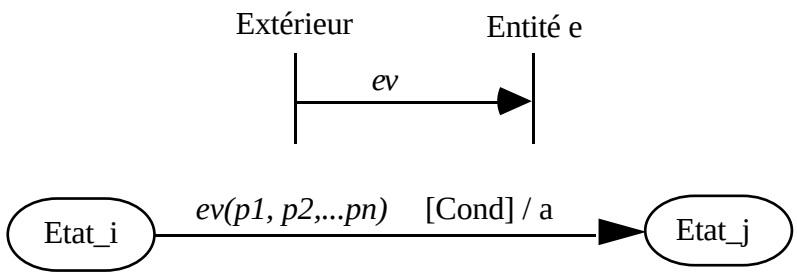
Comme nous avons supposé qu'un événement peut apparaître seulement si la condition de déclenchement est satisfaite, cette vérification ne doit pas être effectuée dans le corps de l'opération car il s'agit ici de l'approche "offensive". Donc les conditions de transition sont traduites en B dans la pré-condition.

Notons que dans [SEK 98], l'auteur propose de vérifier ces conditions au moyen de la structure "IF ... THEN ... ELSEIF... " ce qui correspond à l'approche "défensive".

Cependant, en B, la pré-condition d'une opération doit être levée pendant le raffinement de cette opération. Ceci implique que sa vérification devra être assurée avant l'appel de l'opération raffinée correspondante. Donc, si on considère le processus global de développement du système, tous les cas possibles seront bien pris en compte.

Nous obtenons donc :

Règle D2_O



Chaque scénario mono-entité correspondant à une seule transition avec une seule action

donne lieu à l'opération :

```
ev(o, p1, p2, ...pn) =      /* o est l'objet courant du diagramme */
PRE  Pred_État_i(o)         /* Prédicat décrivant l'État_i */
∧    pi ∈ Type_pi           /* i=1..n */
∧    Cond                   /* Condition de déclenchement de la transition */
∧    Cond_Sup               /* Autres pré-conditions possibles qui ne
                               sont pas décrites sur le diagramme */
THEN
     S                      /* Substitutions correspondant à l'effet de a */
END
```

Remarques

À ce niveau, il n'est pas possible de préciser le corps de l'opération, car cela dépend de la répartition des opérations dans les machines.

Génération de l'obligation de preuve :

Concernant les preuves, pour chaque opération, il faut prouver que, si elle est appelée sous sa pré-condition, avec invariant vrai, elle doit préserver l'invariant et établir le prédicat de l'état arrivée (de la transition portant l'opération).

Plus concrètement, étant donnés :

I : l'invariant de la machine,

P : la pré-condition.

Nous devons prouver que :

Règle D2_P

L'obligation de preuve associée au scénario mono-entité correspondant à une seule transition avec une seule action est :

$$\mathbf{(I \wedge P)} \quad \Rightarrow \quad [\mathbf{S}]\ \mathbf{(I \wedge Pred\_État\_j(o))}$$

avec $P = Pred\_État\_i(o) \wedge p_i \in Type\_p_i\ (i=1..n) \wedge Cond \wedge Cond\_Sup$.

Notons que $[S](I \wedge Pred\_État\_j(o))$ se lit par "S préserve l'invariant I et établit le prédicat de l'État_j". Autrement dit, c'est la condition la plus large telle que après avoir exécuté S, $[S](I \wedge Pred\_État\_j(o))$ soit vrai.

Exemple :
Étant donné les diagrammes de suivi d'événements concernant l'entité Personne et son diagramme E/T suivants :

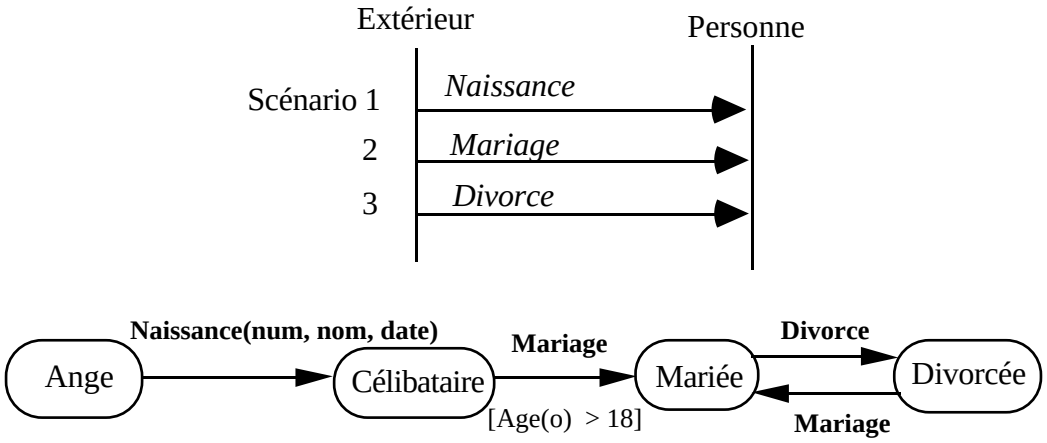


Figure 5.22. *Le diagramme de suivi d'événements et le diagramme E/T de l'entité* Personne

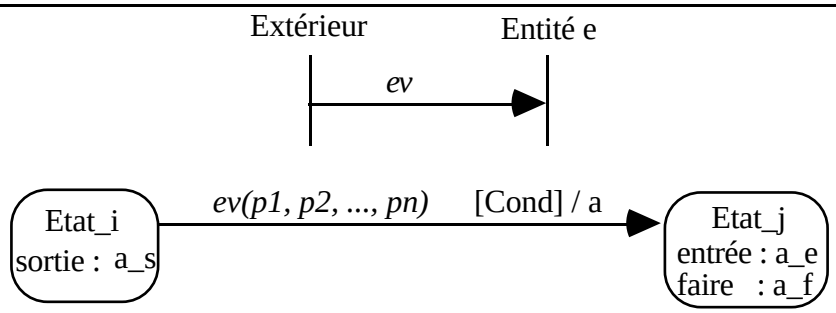
L'opération B correspondant au Scénario Divorce est générée comme ci-dessous :

Divorce(o) =
PRE Mariée(o)
THEN *S* /* Substitutions correspondant au changement de situation en Divorcée d'une personne */
END

III.3.2. Scénario mono-entité correspondant à une seule transition avec plusieurs actions

Dans ce type de scénario, OMT introduit un certain séquencement dans les actions. En B, au premier niveau de la spécification, on ne tient pas compte de l'enchaînement d'actions généré par l'événement et décrit dans le diagramme E/T. Seul l'effet global de ces actions, produit par un événement d'entrée, est spécifié dans l'opération B. Autrement dit, on fait une abstraction des actions pour ne retenir que leur effet conjugué sur le système. Le nombre d'actions que contient un scénario est alors sans importance.

Règle D3_O



Nous ne ferons donc pas de différence entre les scénarios à plusieurs actions et ceux composés d'une seule action. Nous obtenons cette fois-ci S qui décrit l'effet de a_s, suivi de a suivi de a_e et suivi de a_f.

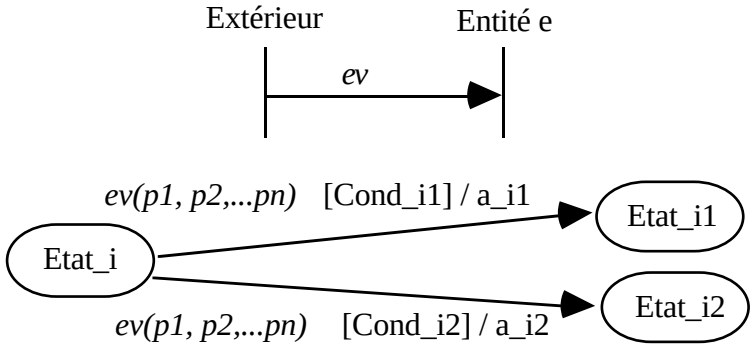
Génération de l'obligation de preuve :

Nous avons la même obligation de preuve que dans le cas précédent, c'est-à-dire : **(I ∧ P) ⇒ [S] (I ∧ Pred_État_j(o)).**

III.3.3. Scénario mono-entité correspondant à plusieurs transitions partant d'un même état avec conditions de déclenchement différentes

Ce type de scénario donne différents schémas d'exécution avec différents états de sortie. Rappelons que, si la condition Cond_i1 est vraie, l'état d'arrivée est l'État_i1. Par contre, dans le cas où Cond_i2 est vraie, l'état d'arrivée est l'État_i2. Ce choix alternatif se traduit par la structure "IF ...THEN ...ELSEIF ...". Nous obtenons donc :

Règle D4_O



Chaque scénario mono-entité correspondant à plusieurs transitions partant d'un même état avec conditions de déclenchement différentes donne lieu à l'opération suivante :

ev(o, p1, p2, ..., pn) =
PRE Pred_État_i(o)
∧ pi ∈ Type_pi
∧ (Cond_i1 ∨ Cond_i2)
∧ *Cond_Sup*
THEN IF Cond_i1 THEN
S1 /* Substitutions correspondant à l'effet de a_i1 */
ELSEIF Cond_i2 THEN
S2 /* Substitutions correspondant à l'effet de a_i2 */
END
END

Nous présentons les obligations de preuves concernant une opération B qui se compose de la substitution "IF C1 THEN S1 ELSEIF C2 S2 END". Nous avons pour cette substitution :

[IF C1 THEN S1 ELSEIF C2 THEN S2 END] Q => (C1 => [S1]Q) ∧ (C2 => [S2]Q)

Remplaçons C1 par Cond_i1, C2 par Cond_i2 et Q par l'invariant et le prédicat de l'état suivant pour chaque état de sortie, nous obtenons donc :

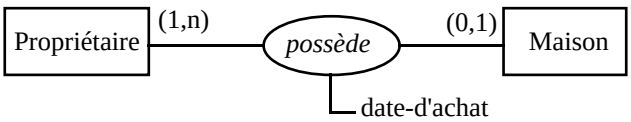
Règle D4_P

L'obligation de preuve associée au scénario mono-entité correspondant à plusieurs transitions partant d'un même état avec conditions de déclenchement différentes est :

(I ∧ Pred_État_i(o) ∧ pi ∈ Type_pi (i=1..n) ∧ (Cond_i1 ∨ Cond_i2) ∧ Cond_Sup)
=> **(Cond_i1 => [S1](I ∧ Pred_État_i1(o)))**
∧ **(Cond_i2 => [S2](I ∧ Pred_État_i2(o)))**

Exemple :

Reprenons l'exemple de l'entité Propriétaire liée à l'entité Maison par l'association Possède :



Supposons que nous ayons les diagrammes suivants :

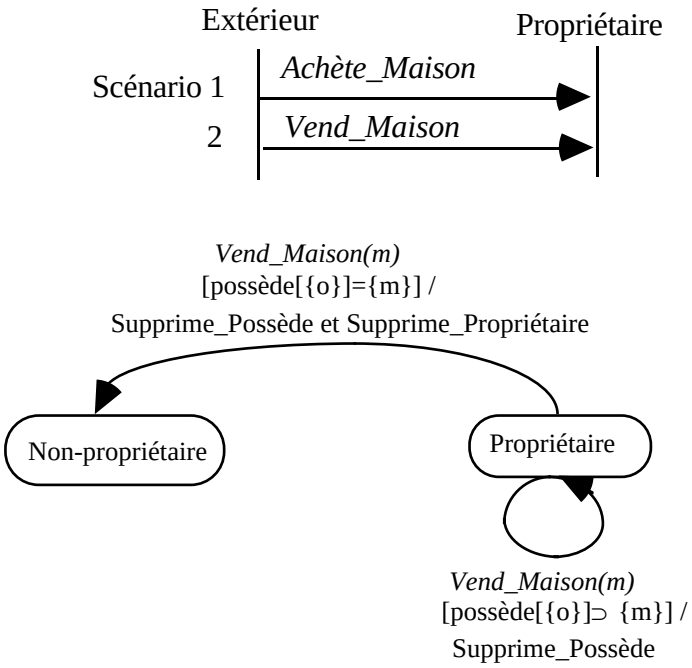


Figure 5.23. *Le diagramme de suivi d'événements et le diagramme E/T de l'entité Propriétaire*

Le scénario Vend_Maison sera traduit comme suit :

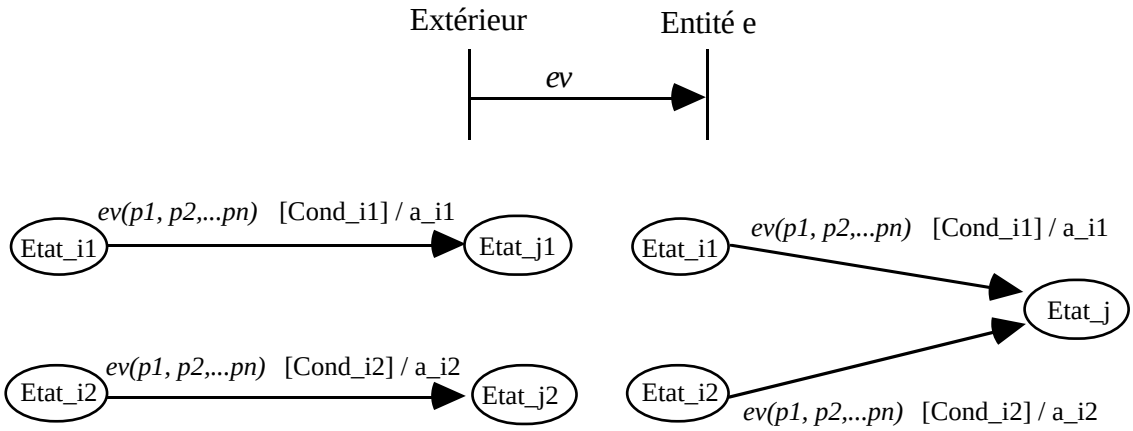
Vend_Maison(o, m) =
PRE Propriétaire(o) ∧
{m} ⊆ possède[{o}]
THEN IF {m} = possède[{o}] THEN
S1 /* Substitutions correspondant à la suppression
du lien possède et du propriétaire */
ELSEIF {m} ⊂ possède[{o}]
S2 /* Substitutions correspondant à la suppression
du lien possède */
END
END

III.3.4. Scénario mono-entité correspondant à plusieurs transitions partant d'états différents

Rappelons qu'à l'arrivée de l'événement ev, une seule transition sera déclenchée en fonction de l'état où se trouve l'objet courant o et des conditions sur les transitions. Si l'objet o se trouve dans l'État_i1 et la condition Cond_i1 est vraie, l'état d'arrivée est l'État_j1. Par

contre, dans le cas où o se trouve dans l'État_i2 et la condition Cond_i2 est vraie, l'état d'arrivée est l'État_j2. Ce choix se traduit par la structure "IF .. THEN ... ELSEIF ...".

Règle D5_O



Chaque scénario mono-entité correspondant à plusieurs transitions partant d'états différents donne lieu à l'opération suivante :

ev(o, p1, p2, ..., pn) =
PRE ((Pred_État_i1(o) ∧ Cond_i1)
∨ (Pred_État_i2(o) ∧ Cond_i2))
∧ pi ∈ Type_pi
∧ *Cond_Sup*
THEN IF Pred_État_i1(o) THEN
S1 /* Substitutions correspondant à l'effet de a_i1 */
ELSEIF Pred_État_i2(o) THEN
S2 /* Substitutions correspondant à l'effet de a_i2 */
END
END

Règle D5_P

L'obligation de preuve associée au scénario mono-entité correspondant à plusieurs transitions partant d'états différents est :

(I ∧ ((Pred_État_i1(o) ∧ Cond_i1) ∨
(Pred_État_i2(o) ∧ Cond_i1)
∧ pi ∈ Type_pi (i=1..n) ∧ Cond_Sup)

=>

(Pred_État_i1(o) => [**S1**]**(I** ∧ **Pred_État_j1(o)))**

∧ **(Pred_État_i2(o)** => [**S2**]**(I** ∧ **Pred_État_j2(o)))**

Remarque

- Dans le cas où nous avons un seul état d'arrivée, il suffit de remplacer dans la règle D5_P Pred_État_j1(o) et Pred_État_j2(o) par Pred_État_j(o).

- La substitution "IF...THEN...ELSEIF..." ne sera pas utilisée dans le cas où les actions ai_1 et ai_2 sont les mêmes.

Exemple :

Reprenons l'exemple de la figure 5.22. Nous nous intéressons cette fois-ci au scénario Mariage. On reconnaît un scénario mono-entité correspondant à plusieurs transitions partant d'états différents.

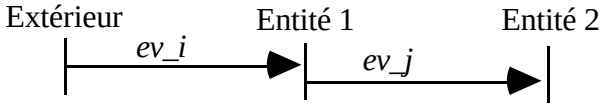
L'opération correspondant au scénario Mariage est spécifiée de la manière suivante :

Mariage(o) =
PRE (Célibataire(o) ∧ Age(o) > 18) ∨
Divorcée(o)
THEN *S* /* Substitutions correspondant au changement
de situation en Mariée d'une personne */
END

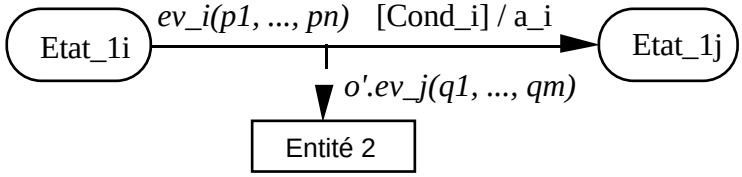
III.3.5. Scénario multi-entités

Si l'objet courant o du diagramme E/T de l'Entité 1 se trouve dans l'État_1i et la condition Cond_i est satisfaite, l'événement ev_i est transmis et l'opération a_i sera exécutée. La transition de o' ne peut être franchie que si o' est dans l'État_2i et Cond_j est vrai. Rappelons qu'à ce niveau, nous nous intéressons seulement à l'effet de ces actions et non pas à la séquence des actions. Il est nécessaire de distinguer le cas où o' se trouve dans l'État_2i et Cond_j est vraie du cas contraire. La structure "IF ... THEN ... ELSE ..." est alors utilisée afin d'exprimer le choix d'exécution. Nous avons donc :

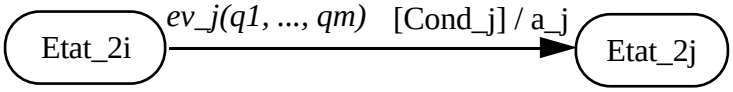
Règle D6_O



Partie du diagramme E/T de l'Entité 1 :



Partie du diagramme E/T de l'Entité 2 :



Rappelons que o' est soit calculé à partir de o (l'objet courant du diagramme E/T de l'Entité 1), soit reçu en tant que paramètre de l'événement ev_i. Et q1,...,qm sont des paramètres à transmettre à o' de l'Entité 2, qui dépendent du contexte.
Chaque scénario multi-entités donne lieu à l'opération suivante :

ev_i(o, p1, p2, ..., pn) =
PRE Pred_État_1i(o)
∧ pi ∈ Type_pi ∧ Cond_i
∧ *Cond_Sup*
THEN IF Pred_État_2i(o') ∧ Cond_j THEN
S1 /* Substitutions correspondant à l'effet de a_i ∧ a_j */
ELSE *S2* /* Substitutions correspondant à l'effet de a_i */
END
END

Génération de l'obligation de preuve :

Concernant les preuves, nous appliquons le même principe que dans le cas précédent. Notons que, dans le cas de l'exécution de S1, les états d'arrivée sont Pred_État_1j pour o et Pred_État_2j pour o'. Nous avons donc :

Règle D6_O

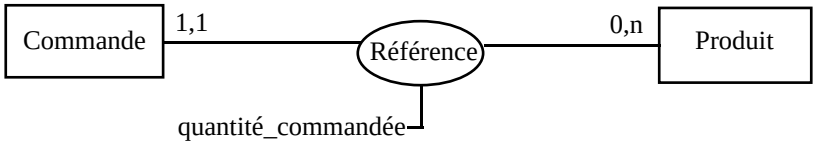
L'obligation de preuve associée au scénario multi-entités est :

(I ∧ Pred_État_1i(o) ∧ p_i e Type_p_i (i=1..n) ∧ Cond_i ∧ Cond_Sup)
=> (Pred_État_2i(o') ∧ Cond_j => [S1](I ∧ Pred_État_1j(o) ∧ Pred_État_2j(o')))
∧ (¬ (Pred_État_2i(o') ∧ Cond_j) => [S2](I ∧ Pred_État_1j(o)))

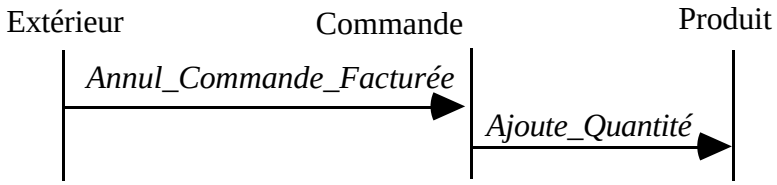
Remarque

La substitution "IF...THEN...ELSE..." ne sera pas utilisée si la condition Cond_j n'existe pas et si la contrainte Pred_État_2i(o') a été implicitement vérifiée dans la pré-condition de l'opération.
Exemple :
Étant donné les diagrammes suivants :

Diagramme d'objet :



Partie du diagramme de suivi d'événements :



Partie du diagramme E/T de l'entité Commande :

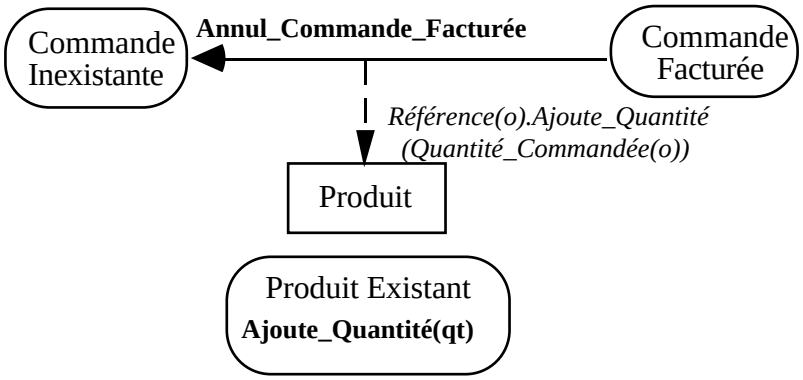


Figure 5.24. *Un exemple de scénario multi-entités*

Le scénario débutant par l'événement Annul_Commande_Facturée est spécifié comme suit :

Annul_Commande_Facturée(o) =
PRE Commande_Facturée(o) ∧

Quantité_Stock(Référence(o)) + Quantité_Commandée(o) ∈ NAT

THEN

S /* Substitutions correspondant à la suppression d'une commande et à l'ajout d'une quantité pour le produit correspondant */

END

- Remarquons que le corps de l'opération générée ne contient pas la structure "IF ... THEN ... ELSE" car dans le diagramme E/T de Produit, la condition de déclenchement n'existe pas et la contrainte sur le prédicat de l'état du produit associé à la commande a été vérifiée de manière indirecte dans la pré-condition. Nous allons montrer que Commande_Facturée(o) implique Produit_Existant(Référence(o)).
  Nous avons les fonctions totales suivantes :

  État_cd ∈ Commandes → {Facturée, En_Attente}

  et Référence ∈ Commandes → Produits

  Et les définitions : Commande_Facturée(o) ≙ o ∈ État_cd $^{-1}$ [{Facturée}] ;

  Produit_Existant(o) ≙ o ∈ Produits

  Donc, si nous avons :

  Commande_Facturée(o)
  => o ∈ État_cd $^{-1}$ [{Facturée}]
  => o ∈ Comandes
  => Référence(o) ∈ Produits
  => Produit_Existant(Référence(o))

- La condition Quantité_Stock(Référence(o)) + Quantité_Commandée(o) ∈ NAT assure que Quantité_Stock est toujours un entier quand on ajoute une nouvelle quantité Quantité_Commandée(o) pour le produit Référence(o).

III.3.6. Scénario dont l'événement d'entrée arrive à l'administrateur d'une entité

Référence figure 3.30.

Rappelons que, dans ce diagramme, l'événement ev_i ne contient pas d'objet courant car l'objet concerné n'est pas encore déterminé à ce stade. L'opération consiste, d'abord, à sélectionner l'objet ou un ensemble d'objets vérifiant le critère de sélection (Cond_Select), le prédicat de l'État_i du diagramme E/T de chaque objet sélectionné (Pred_État_i), et la condition de déclenchement (Cond_Dec). On peut avoir Cond_Select implique Pred_État_i et Cond_Dec. Après l'exécution des substitutions correspondant à l'action ev_j, chaque élément

concerné doit respecter le prédicat de l'État_j. Une telle sélection est traduite en B par la structure "ANY ... WHERE ...". Nous avons donc :

Règle D7_O

Chaque scénario dont l'événement d'entrée arrive à l'administrateur d'une entité donne lieu à l'opération suivante :

ev_i(p1, p2, ..., pn) =
PRE pi ∈ Type_pi
∧ Cond_Exist
∧ *Cond_Sup*
THEN ANY Ens /* ou e */ WHERE Cond_Ens THEN
S /* Substitutions correspondant à l'effet de a*/
END
END

NB :

- Ens (ou e) est l'ensemble d'objets (ou l'objet) sur lequel nous appliquons l'opération a.
- La condition Cond_Exist est la condition de déclenchement du diagramme administrateur. Elle décrit l'existence d'au moins un élément qui vérifie le critère de sélection.
- La condition Cond_Ens inclut :
  - La condition vérifiant que chaque instance de Ens vérifie Pred_État_i,
  - La condition de déclenchement Cond_Dec,
  - La condition exprimant le critère de sélection Cond_Select.

Génération de l'obligation de preuve :

Afin d'obtenir l'obligation de preuve associée au diagramme administrateur d'une entité, nous allons étudier l'obligation de preuve d'une opération qui se compose de la substitution ANY ... WHERE ... THEN ... END. Plus précisément, étant donnée I, l'invariant de la machine, l'opération op, et R l'ensemble des conditions que doivent vérifier op après son exécution : op

PRE P
THEN ANY e WHERE Q
THEN S END
END

Nous devons prouver que :

$I \wedge P$ => [ANY e WHERE Q THEN S] R

=> [@ e. (Q => S)] R

=> $\forall$ e. [Q => S] R

=> $\forall$ e. (Q(e) => [S] R) (1)

Remplaçons :

P	par	$p_i \in Type\_p_i$ (i=1..n) $\wedge$ Cond_Exist $\wedge$ Cond_Sup
e	par	Ens
Q	par	Cond_Ens
R	par	$I \wedge \forall$ o. (o $\in$ Ens => Pred_État_j(o))

Nous obtenons pour (1) :

$I \wedge p_i \in Type\_p_i$ (i=1..n) $\wedge$ Cond_Exist $\wedge$ Cond_Sup =>

$\forall$ Ens.(Cond_Ens(Ens)) => [S]($I \wedge \forall$ o.(o $\in$ Ens => Pred_État_j(o)))

<=> $I \wedge p_i \in Type\_p_i$ (i=1..n) $\wedge$ Cond_Exist $\wedge$ Cond_Sup

=> $\forall$ Ens.(Cond_Ens) => [S]($I \wedge \forall$ o.(o $\in$ Ens => Pred_État_j(o)))

Nous avons donc :

Règle D_7_P

L'obligation de preuve pour des scénarios associés au diagramme administrateur d'une entité est donc de la forme :

$I \wedge p_i \in Type\_p_i$ (i=1..n) $\wedge$ Cond_Sup $\wedge$ Cond_Exist

=> ($\forall$ Ens.(Cond_Ens) => [S]($I \wedge \forall$ o.(o $\in$ Ens => Pred_État_j(o))))

Exemple :

Examinons les diagrammes E/T suivants associés au scénario Création_Cd

Diagramme E/T de l'Administrateur de Commande

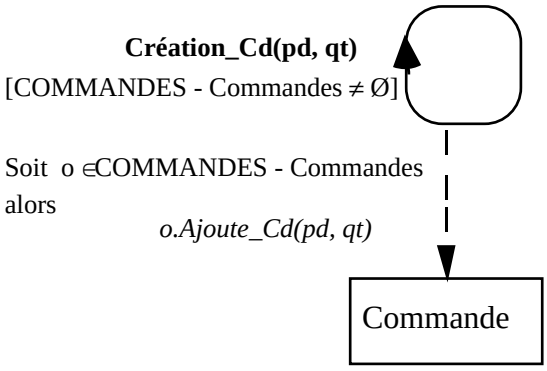


Diagramme E/T de l'entité Commande

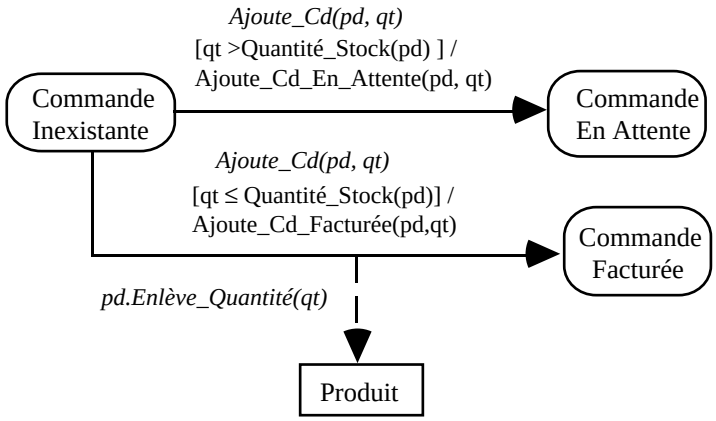


Diagramme E/T de l'entité Produit

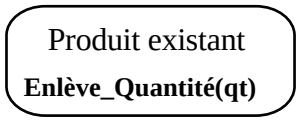


Figure 5.25. *Un exemple de scénario associé au diagramme administrateur de l'entité Commande*

L'événement Création_Cd est un événement externe, il ne contient pas d'objet courant car il s'agit de créer une nouvelle commande. L'opération de création consiste à choisir un élément quelconque parmi les instances libres puis à l'ajouter dans l'ensemble des instances existantes.

Création_Cd(pd, qt) =
PRE pd ∈ Produits ∧ qt ∈ NAT ∧
COMMANDES - Commandes ≠ Ø
THEN ANY o WHERE o ∈ COMMANDES - Commandes THEN
IF qt ≤ Quantité_Stock(pd) THEN
S1 /* Substitutions correspondant à l'ajout d'une commande,
à la facturation immédiate de cette commande et
à l'enlèvement d'une quantité en stock qt du produit pd */
ELSE
S2 /* Substitutions correspondant à l'ajout d'une commande */
END
END
END

III.4. Répartition des opérations dans les machines

Après ces deux premières étapes, en ce qui concerne les opérations utilisateurs, c'est-à-dire les opérations correspondant aux scénarios, nous avons une description des paramètres, des pré-conditions nécessaires et de la structure du corps. Le contenu du corps de chaque opération dépend fortement de la manière dont nous allons choisir la machine où elle sera spécifiée.

Dans la première partie de cette recherche, nous avons étudié la décomposition en machines de la spécification obtenue par la formalisation des concepts du modèle d'objets. Notre critère était de rester le plus fidèle possible à la structure du diagramme d'objets. Le schéma ci-dessous présente la structure des machines abstraites obtenues après la formalisation du diagramme d'objets.

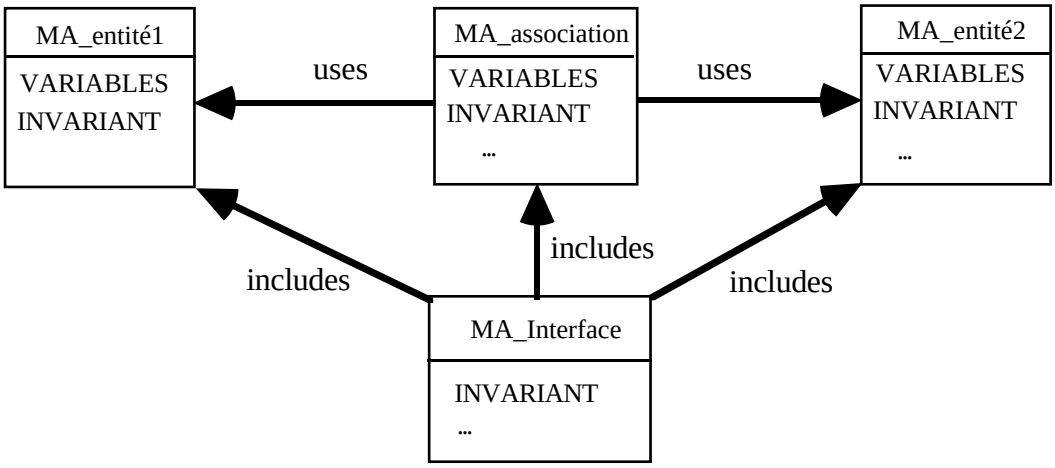


Figure 5.26. *Structure des machines obtenues après la formalisation du diagramme d'objets*

Cette partie est consacrée à la répartition des opérations, générées à partir des diagrammes E/T et scénarios, dans les machines obtenues à la première partie. Deux solutions sont possibles.

La première solution consiste à mettre les opérations dans les machines suivant les variables auxquelles accèdent les opérations. Puisque ces opérations correspondent aux événements d'entrée des scénarios, elles doivent être visibles aux utilisateurs. Pour cela, nous allons utiliser le lien *Promotes* : pour chaque opération qui n'est pas spécifiée dans la MA_Interface, nous ajoutons alors son nom dans la partie *Promotes* de cette machine.

Exemple :
Imaginons que nous ajoutons, pour l'entité Personne, l'attribut Sexe et nous développons le diagramme E/T de la figure 5.22. comme suit :

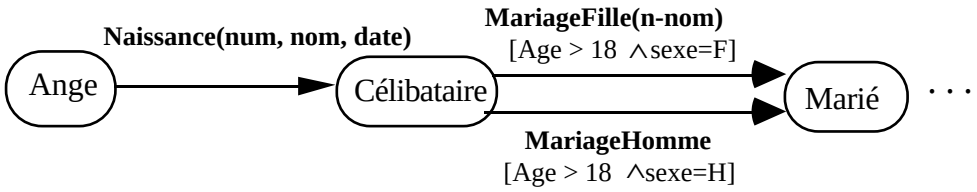


Figure 5.27. *Un exemple de diagramme E/T de l'entité Personne*

La naissance d'un ange le fait passer dans l'état Célibataire puis en situation Marié à son mariage. Souvent lorsqu'une fille se marie, elle change à la fois sa situation et son nom. L'opération Mariage_Fille modifiera non seulement la valeur de l'attribut Situation mais aussi l'attribut Nom.

En appliquant cette solution, l'opération Mariage_Fille correspondant au scénario Mariage_Fille sera définie dans la machine Personne avec dans la partie pré-condition, les conditions sur la transition du diagramme E/T. Puis nous l'ajoutons dans la partie PROMOTES de la machine Interface pour que cette opération soit visible aux utilisateurs.

Nous obtenons donc :

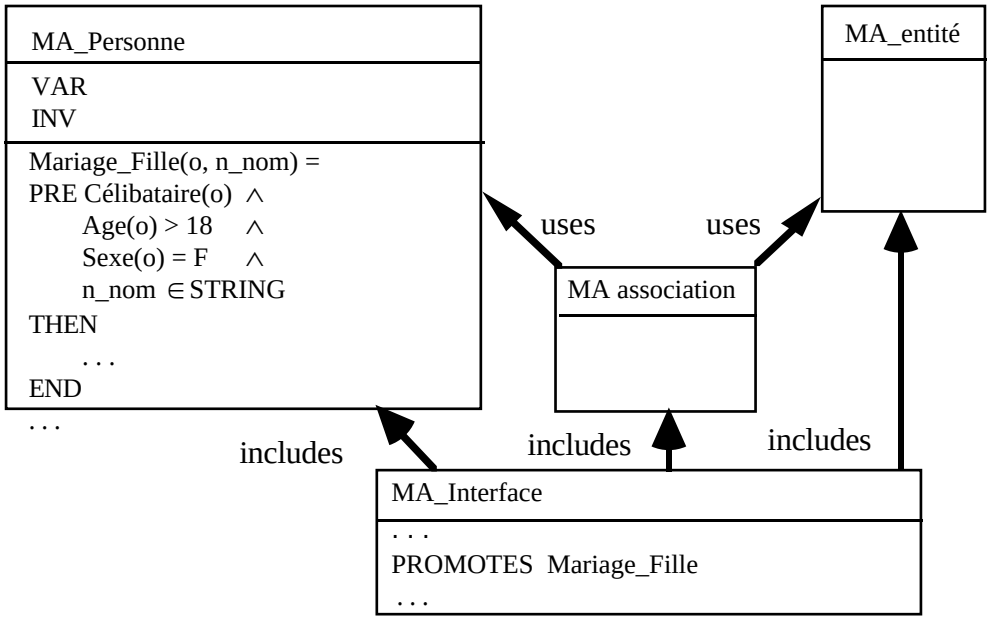


Figure 5.28. *Un exemple de la répartition des opérations*

L'avantage de cette solution est qu'elle fournit une spécification bien modulaire car les opérations sont décrites dans différentes machines abstraites. Cependant, elle complique la prise

en compte de l'évolution des diagrammes car à chaque maintenance, nous devons d'abord localiser la machine concernée, la modifer puis mesurer l'impact de la modification sur d'autres machines qui l'utilisent. C'est pour cela que nous préférons opter pour la solution présentée ci-dessous.

Puisque les opérations issues des scénarios servent d'interface aux utilisateurs, nous pouvons les mettre dans la machine MA_Interface.

Nous nous intéressons maintenant à une de ces machines de base. Par exemple, la machine entité. À partir du diagramme d'objets, nous pouvons obtenir non seulement les variables, les invariants décrivant la structure de cette entité mais aussi les opérations de base (c'est-à-dire les opérations de création, de suppression, de mise à jour). L'ensemble de ces opérations est d'abord spécifié séparément de la machine elle-même pour pouvoir effectuer certaines modifications liées à des contraintes techniques de B. En effet, l'utilisation du lien *includes* nous empêche d'appeler en parallèle deux opérations de la même machine. Ceci nous oblige à pré-composer ces opérations pour former une opération élémentaire avant son introduction dans la machine entité. Le corps des opérations de la machine Interface est composé d'appels des diverses opérations élémentaires.

Le principe présenté peut être résumé par le schéma suivant :

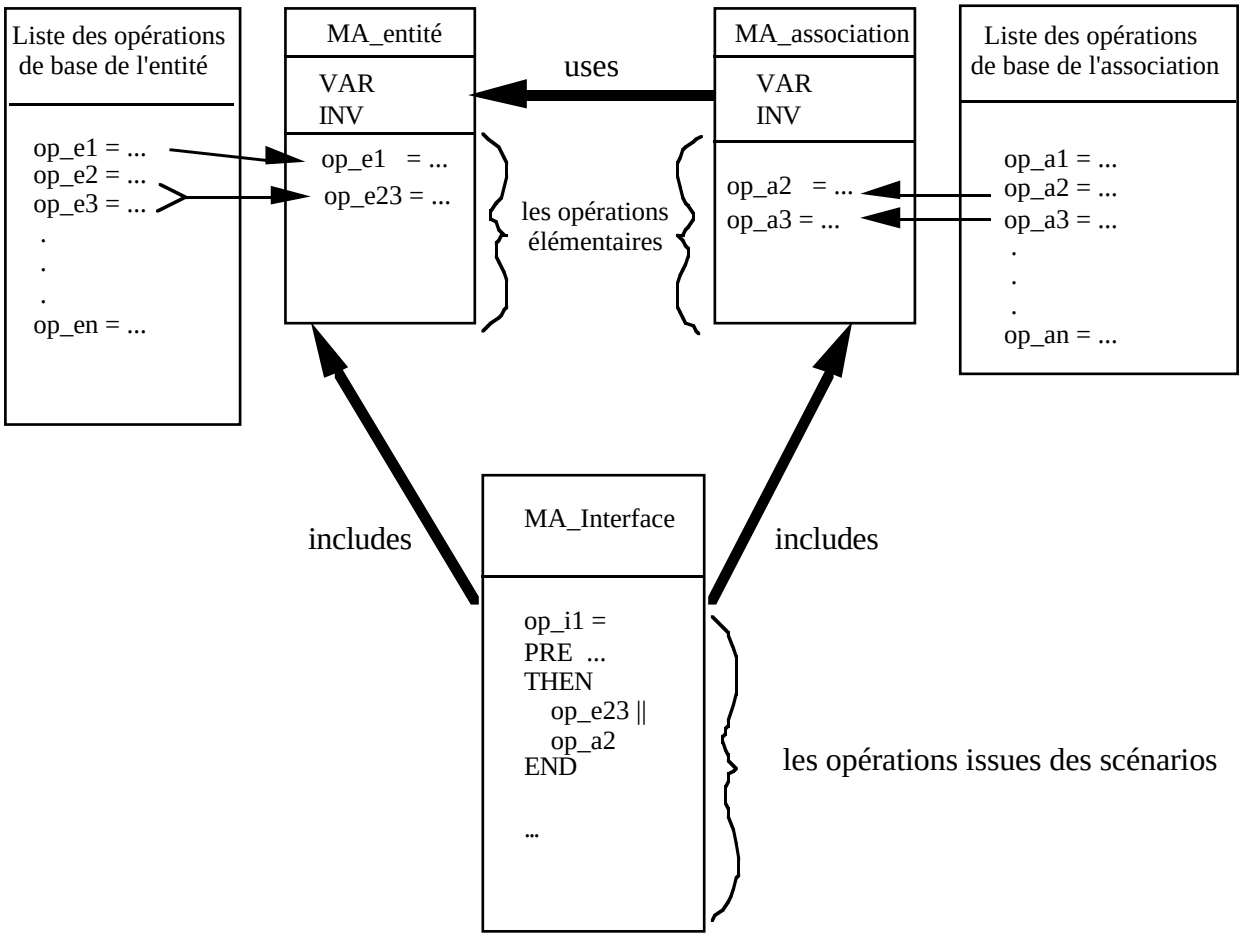


Figure 5.29. *Le schéma général de la répartition des opérations*

Nous avons donc la règle suivante :

Règle D8

- Chaque opération déduite à partir du scénario associé est spécifiée dans la MA_Interface ;
- Son corps se compose d'appels des opérations élémentaires.

Exemple :

Reprenons l'exemple du cas précédent. La deuxième solution consiste à composer les deux opérations de base Change_Situation et Change_Nom en une opération élémentaire qui s'appelle Change_Nom_Situation et de l'ajouter dans la machine Personne. Dans la machine Interface, l'opération Mariage_Fille va faire appel à cette opération.

Nous avons donc :

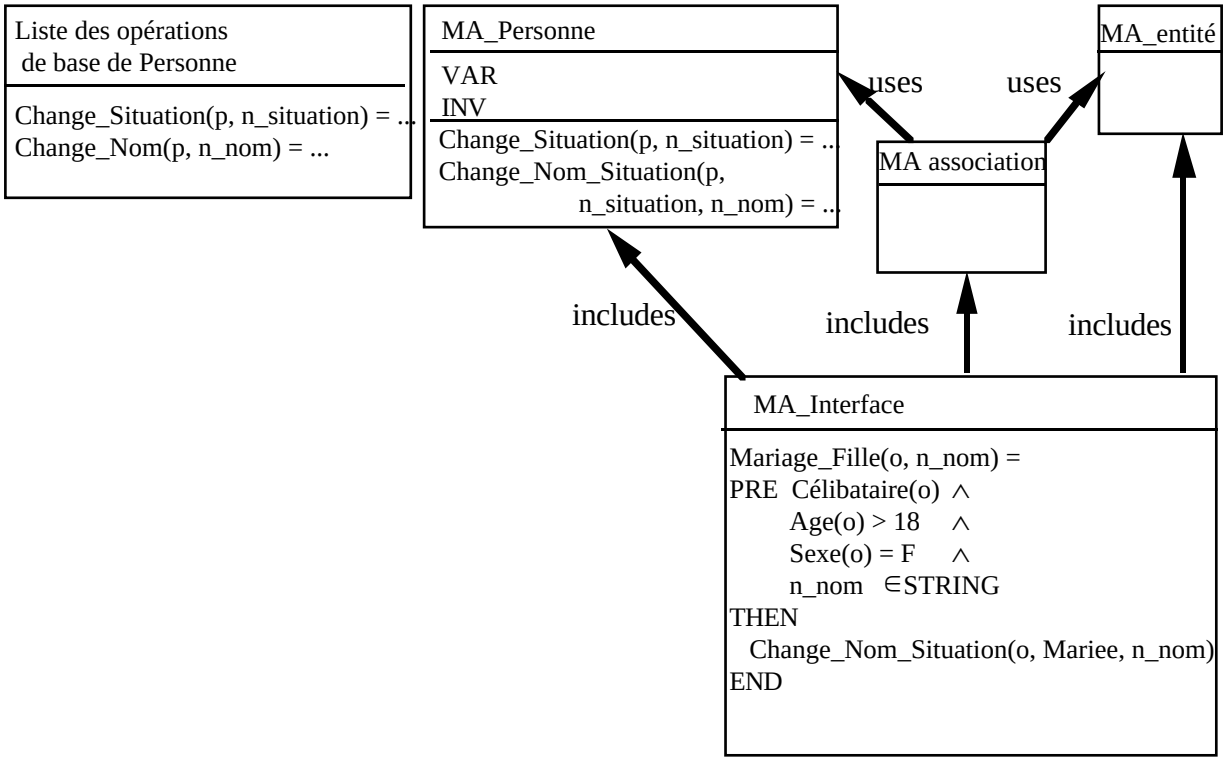


Figure 5.30. *Un exemple de la répartition des opérations en appliquant la règle D7*

III.5. Autre technique possible : utilisation du raffinement

Rappelons que, dans les diagrammes E/T, chaque opération fait passer un objet "courant" d'un état source à un état d'arrivée. Elle doit donc, d'une part, conserver l'invariant et d'autre part, assurer que l'objet "courant" se trouve bien dans l'état d'arrivée.

Avec la démarche choisie pour traduire les diagrammes E/T et les scénarios, le deuxième type d'obligation de preuve ne peut pas être généré automatiquement par le prouveur de l'Atelier B. De ce fait, nous proposons des règles permettant de générer les obligations de preuve assurant que, si chaque opération est appelée sous sa pré-condition, avec l'invariant vrai, non seulement elle doit préserver l'invariant mais *elle doit en plus établir le prédicat de l'état d'arrivée.*

Il est possible, par une démarche d'abstraction, de transformer ces obligations de preuve supplémentaires par des preuves de raffinement qui seront alors générées automatiquement par l'Atelier B. Cette démarche consiste à créer une machine MA_Système dont le raffinement est la machine MA_Interface. Les opérations de cette machine utilisent alors la substitution *devient tel que* qui permet de remplacer des variables par des valeurs qui satisfont à un prédicat donné.

Les étapes à suivre dans ce cas sont alors :

- Dans un premier temps, les règles à appliquer sont les mêmes que celles proposées dans notre étude (les règles D1, D2_O --> D6_0, D7). On obtient la MA_Interface.
- Puis on ajoute une machine MA_Système qui sera raffinée par la machine MA_Interface et dans laquelle on déclare toutes les variables modifiables par les opérations correspondant aux scénarios et on spécifie les invariants associés à ces variables.
- Dans la machine MA_Système, les opérations vont être spécifiées de la manière suivante :
  - Une opération de la machine MA_Interface donne lieu à une opération de la machine MA_Système.
  - À partir des opérations élémentaires appelées dans une opération de la machine MA_Interface, on récupère les variables modifiables par l'opération (v1, ..., vk).
  - L'opération correspondante dans la machine MA_Système devient alors :

    Nom_op(o, p1, ..., pn)
    PRE
    Pré-condition de la même opération dans la MA_Interface (qui inclut Préd_État_Source)
    THEN
    v1, ..., vk : Préd_État_Arrivée(o) $\wedge$ I
    /* I est l'invariant de la machine MA_Système */
    END

    La preuve de cette opération est alors :
    (I $\wedge$ Pré-condition) => [v1, ..., vk : Préd_État_Arrivée(o) $\wedge$ I] I

Lors du raffinement, l'obligation de preuve assurant que chaque opération de la machine MA_Interface établit le prédicat de l'état d'arrivée sera alors générée automatiquement comme obligation de preuve de cohérence entre le raffinement (MA_Interface) et la spécification initiale (MA_Système).

Nous avons le schéma général suivant :

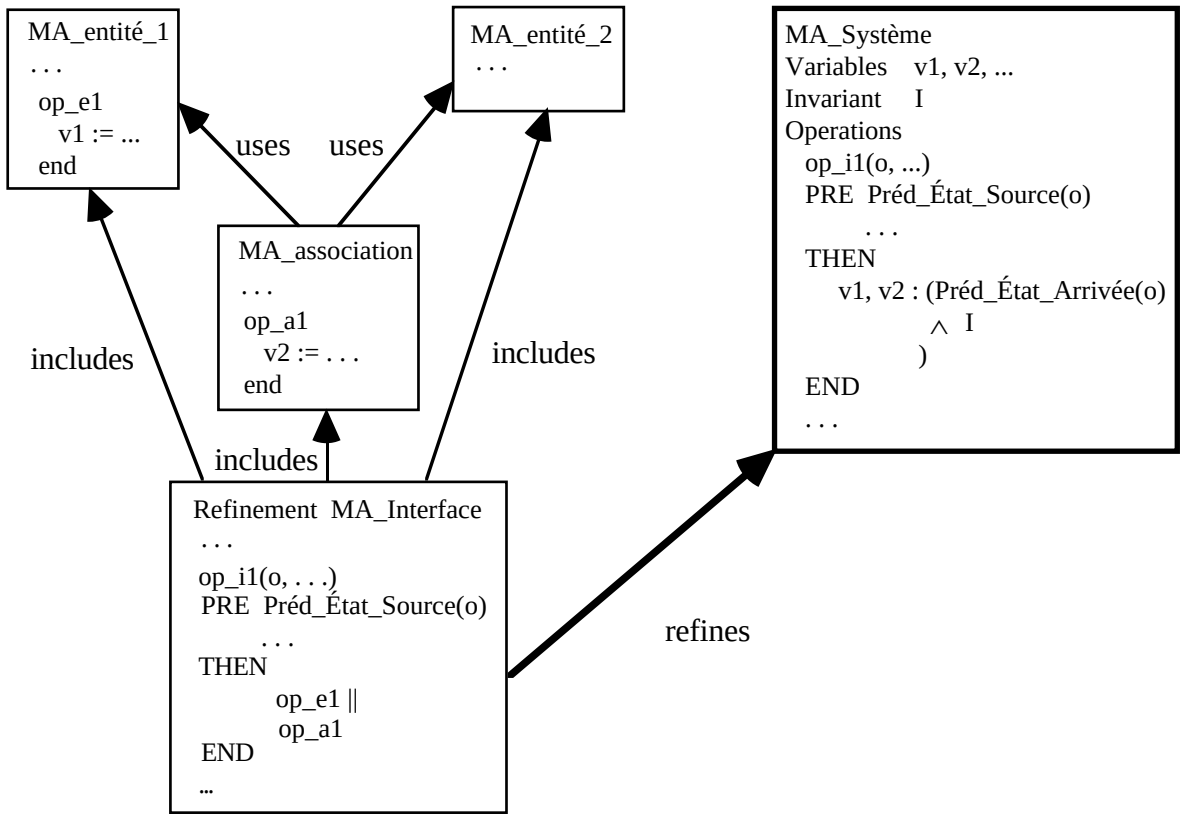
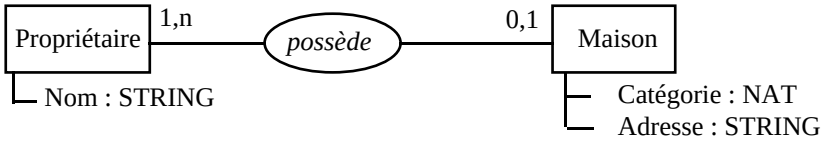
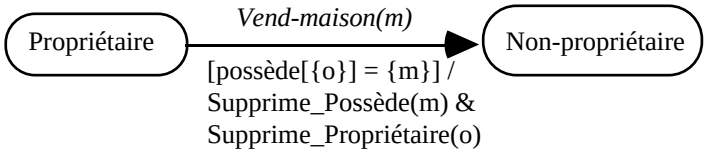


Figure 5.31. *Le schéma général de la technique utilisant le raffinement*

Exemple :

Reprenons le diagramme d'objets entre les entités Propriétaire, Maison en ajoutant les attributs pour ces deux entités, ainsi que le diagramme E/T de Propriétaire.





En suivant les étapes ci-dessus, nous obtenons les machines suivantes :

MACHINE	MA_Propriétaire

SETS PROPRIÉTAIRES
VARIABLES Propriétaires, Nom
INVARIANT Propriétaires $\subseteq$ PROPRIÉTAIRES $\wedge$
Nom $\in$ Propriétaires $\rightarrow$ STRING
INITIALISATION Propriétaires, Nom := {}, {}
OPERATIONS *Supprime_Propriétaire(p)* =
PRE p $\in$ Propriétaires
THEN Propriétaires := Propriétaires – {p} ||
Nom := {p} $\lhd$ Nom(o)
END
...
END

MACHINE MA_Maison
SETS MAISONS
VARIABLES Maisons, Catégorie, Adresse
INVARIANT Maisons $\subseteq$ MAISONS $\wedge$ Catégorie $\in$ Maisons $\rightarrow$ NAT $\wedge$
Adresse $\in$ Maisons $\rightarrow$ STRING
INITIALISATION Maisons, Catégorie, Adresse := {}, {}, {}
OPERATIONS **...**
END

MACHINE MA_Possède
USES MA_Propriétaire, MA_Maison
VARIABLES Possède
INVARIANT Possède^{-1} $\in$ Maisons $\nrightarrow$ Proprétaires
INITIALISATION Possède := {}
OPERATIONS *Supprime_Possède(m)* =
PRE m $\in$ ran(Possède)
THEN Possède := Possède $\rhd$ {m}
END
...
END

REFINEMENT MA_Interface

INCLUDES MA_Propriétaire, MA_Maison, MA_Possède
REFINES MA_Système
DEFINITIONS Propriétaire(o) ≙ o ∈ Propriétaires ;
Propriétaire(o) ≙ o ∈ PROPRIÉTAIRES - Propriétaires
OPERATIONS *Vend_Maison(o,m)* =
PRE Propriétaire(o) ∧
Possède[{o}] = {m}
THEN Supprime_Possède(m) ||
Supprime_Propriétaire(o)
END
...
END

MACHINE MA_Système
SETS PROPRIÉTAIRES, MAISONS
VARIABLES Propriétaires, Nom, Possède, Maisons
INVARIANT Propriétaires ⊆ PROPRIÉTAIRES ∧
Maisons ⊆ MAISONS ∧
Nom ∈ Propriétaires → STRING ∧
$Possède^{-1}$ ∈ Maisons ⇸ Proprétaires
DEFINITIONS Propriétaire(o) ≙ o ∈ Propriétaires ;
Non_Propriétaire(o) ≙ o ∈ PROPRIÉTAIRES - Propriétaires
OPERATIONS *Vend_Maison(o,m)* =
PRE Propriétaire(o) ∧ Possède[{o}] = {m}
THEN
Possède, Propriétaires, Nom : (Non_Propriétaire(o) ∧
Propriétaires ⊆ PROPRIÉTAIRES ∧
Maisons ⊆ MAISONS ∧
Nom ∈ Propriétaires → STRING ∧
$Possède^{-1}$ ∈ Maisons ⇸ Proprétaires)
END
END

Cette technique est également intéressante. Les opérations de la MA_Système sont en fait les plus abstraites et indéterministes possibles : elles précisent uniquement quelles sont les

variables modifiées, et ce que leurs modifications doivent respecter sans dire comment elles sont faites.

IV. Réalisation d'un prototype

Cette partie décrit le processus de construction d'un prototype d'aide à la dérivation de spécifications formelles B à partir de spécifications semi-formelles qui est en cours de développement.

IV.1. Les fonctionnalités du prototype

Le prototype offre dans un premier temps un moyen de définir les diagrammes d'objets et diagrammes états/transitions. Ces deux types de diagrammes sont ensuite traduits en B selon les règles présentées dans ce chapitre. Le prototype contient donc les modules suivants :

IV.1.1. Module de saisie des diagrammes

La saisie de la description des diagrammes d'objets, diagrammes états/transitions se fait par une interface graphique.

IV.1.2. Module de traduction du modèle d'objets en B

La traduction du modèle d'objets se fait en deux phases : la première concerne la traduction du diagramme d'objets, et la deuxième la génération des opérations de base.

IV.1.3. Module de traduction du modèle dynamique en B.

Le modèle dynamique est traduit en plusieurs étapes. Dans la première étape, il faut intégrer les notations B dans les diagrammes E/T à l'aide de la spécification B du diagramme d'objets et de l'utilisateur. Il s'agit ensuite de formaliser les états à partir de variables et d'invariants. La troisième étape concerne la génération de squelettes d'opérations B et les obligations de preuve à partir des scénarios et diagrammes E/T. L'étape suivante consiste à

déduire les opérations élémentaires à partir des opérations de base et des squelettes d'opérations de la machine Interface. La dernière étape complète la machine Interface par l'appel des opérations élémentaires dans le corps des opérations générées, ainsi que par les invariants correspondant aux contraintes d'intégrité non exprimables graphiquement.

Le schéma ci-dessus désigne l'architecture fonctionnelle du système :

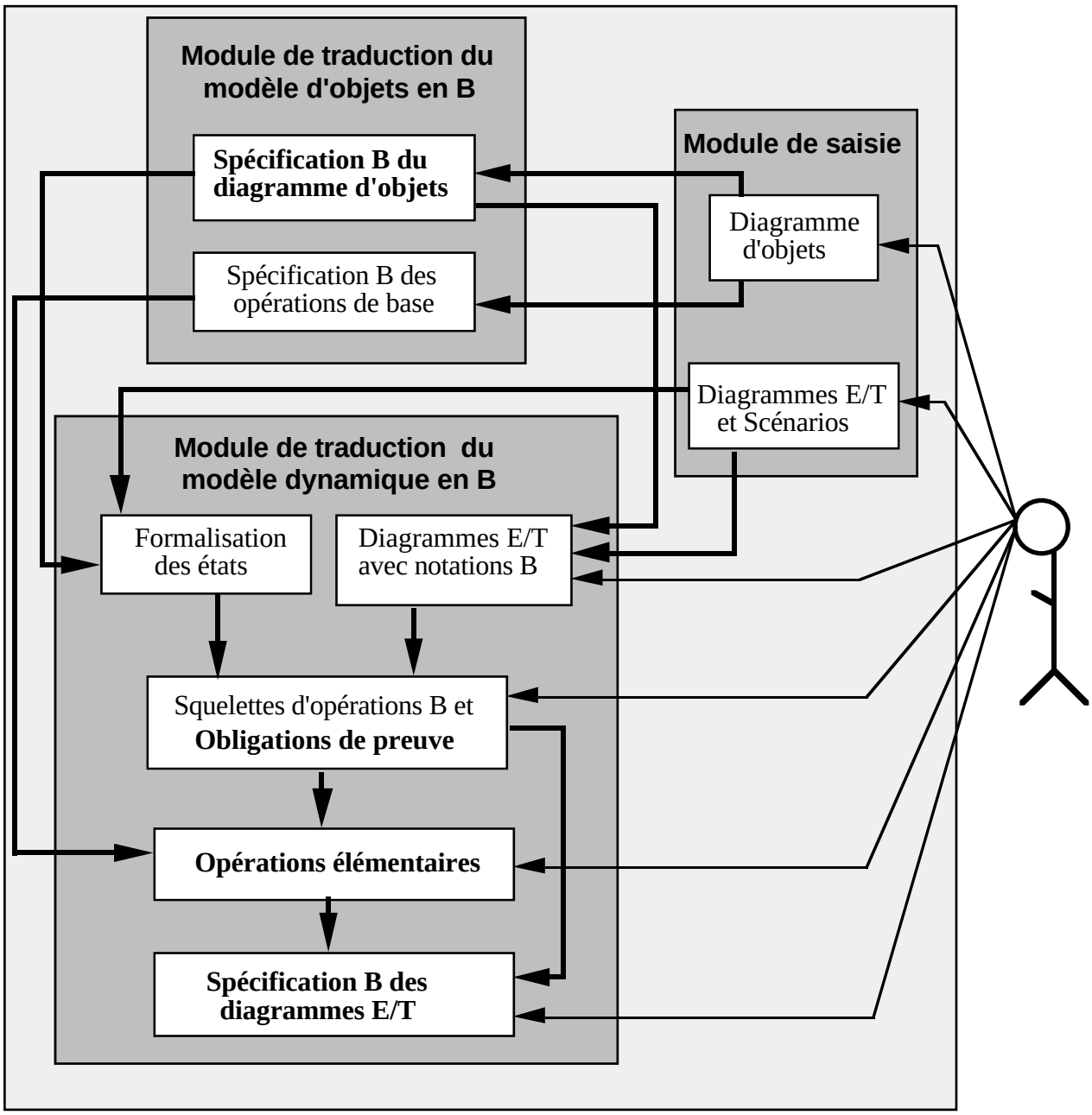


Figure 5.32. *Architecture fonctionnelle du système*

IV.2. La réalisation du prototype

Nous présentons ici les deux parties principales : la saisie des diagrammes et leur traduction en B.

IV.2.1. Le module de saisie des diagrammes

Ce module propose au concepteur une interface graphique qui lui permet de spécifier ses diagrammes. Cette interface graphique est construite autour de l'atelier Graphtalk-Métamodélisation [GRA 94]. Cet atelier permet la réalisation d'ateliers de conception de systèmes d'information. Ces derniers sont ensuite utilisés pour dessiner, manipuler les modèles, tout en assurant la cohérence des formalismes, du vocabulaire et de la grammaire employés.

La conception grâce à Graphtalk-Métamodélisation s'effectue par une programmation graphique, au travers de quatre modèles :

- Spécification sémantique : ce type de graphe permet de décrire les différents modèles qui peuvent exister, par exemple le modèle d'objets, le modèle dynamique. Il permet aussi de représenter les objets complexes, les hiérarchies des classes et le comportement dynamique des objets.

- Spécification des propriétés : elle contient les formes graphiques des classes de graphes, d'objets et de liens.

- Spécification des formes : elle permet de définir et de paramètrer uniquement les propriétés pour chacune des classes du modèle.

- Spécification des fenêtres : elle décrit les fenêtres de dialogues avec l'utilisateurs.

En utilisant le langage graphique offert par Graphtalk, nous avons spécifié les formalismes, le vocabulaire et la grammaire du modèle d'objets et du modèle dynamique définis dans le chapitre 3. L'atelier Graphtalk-Métamodélisation a ensuite généré un outil qui permet de saisir les diagrammes du modèle d'objets et du modèle dynamique et d'en extraire les informations (entité, association, attributs, ...).

À titre d'illustration, la figure 5.33 fournit un exemple de diagramme d'objets. À partir de Graphtalk-Métamodélisation, nous avons créé l'atelier Graphtalk-OMT utilisé ici pour saisir un exemple concret : deux entités Voiture et Personne sont reliées par l'association Possède. La partie droite de la figure (fenêtres "Recherche") montre les informations que l'on peut extraire du diagramme.

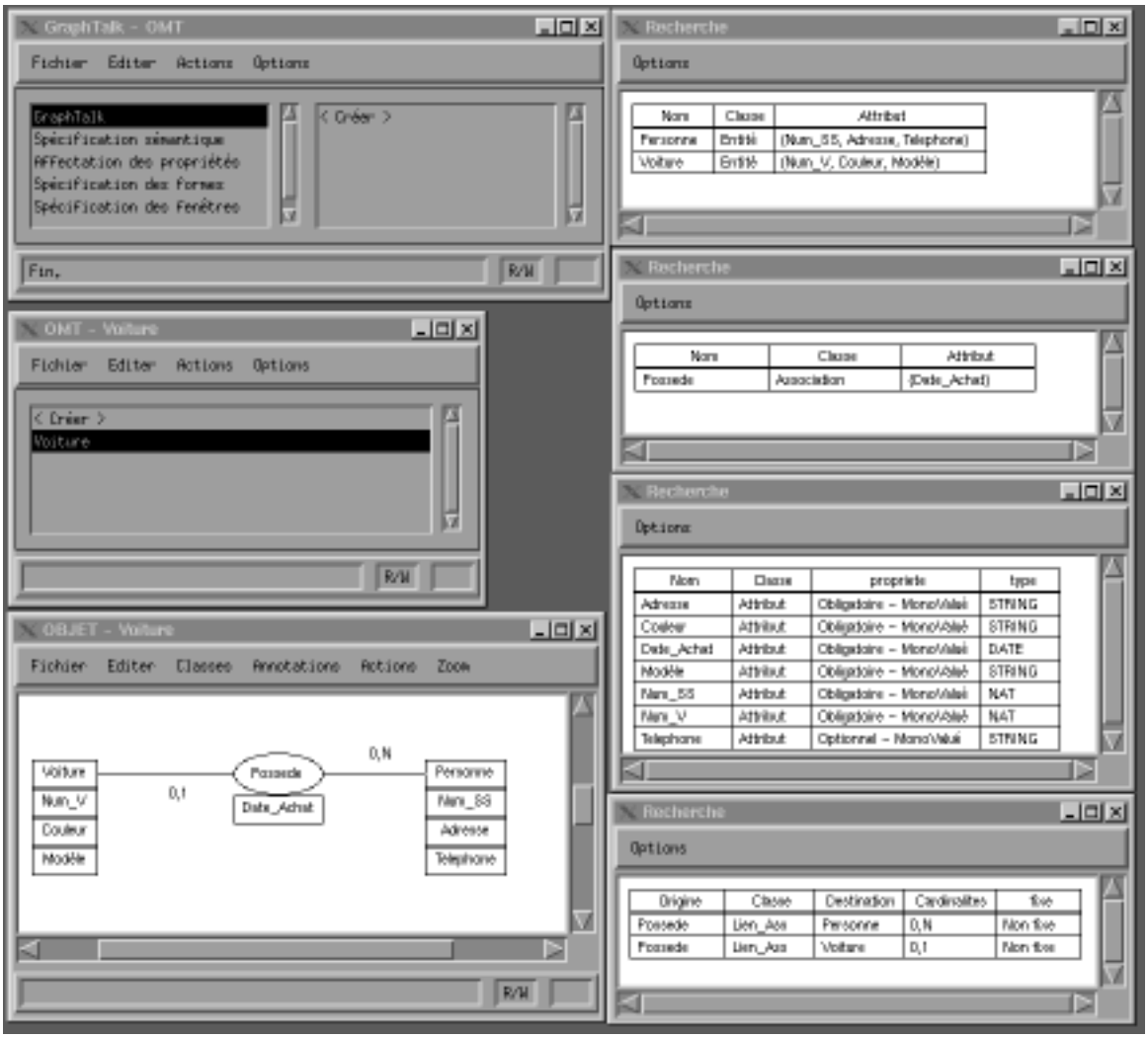


Figure 5.33. *Un exemple de diagramme d'objets saisi avec l'atelier Graphtalk-OMT*

IV.2.2. Le module de traduction des diagrammes

La traduction en B des diagrammes d'objets et des diagrammes états/transitions consiste à passer d'un formalisme à un autre. Le formalisme choisi pour représenter les deux modèles est la syntaxe abstraite. Pour cela, nous avons définit la syntaxe abstraite du modèle semi-formelle et de B. Le passage d'une syntaxe abstraite à l'autre consiste à appliquer les règles proposées dans ce chapitre. Cette partie de l'outil est en cours de développement.

V. Conclusion

Dans ce chapitre, nous avons présenté un ensemble de règles de dérivation d'une spécification formelle B à partir d'une spécification semi-formelle. Concernant la traduction du modèle d'objets, trois aspects ont été examinés : la formalisation des concepts, la modularisation de la spécification et la génération des opérations de base. En ce qui concerne la dynamique, nous avons étudié la formalisation d'un état, la génération des opérations ainsi que la répartition des opérations dans les machines.

Pour la génération des opérations de base, nous avons pris en compte la cardinalité des liens directs. Une extension possible serait de prendre en compte d'autres contraintes (par exemple les contraintes entre ensembles de liens, ...) pour générer des opérations de modification en cascade.

Remarquons que la traduction du modèle dynamique est plus difficile que celle du modèle d'objets. En effet, dans les méthodes semi-formelles, le modèle d'objets est mieux défini, avec une sémantique assez claire, que le modèle dynamique :

- Les diagrammes E/T de OMT sont supposés être utilisés à tous les niveaux, de la spécification à la conception détaillée. Or, par leur aspect opératoire, ils relèvent plus de la phase de conception que de celle de spécification. Il n'est évidemment pas très satisfaisant de partir d'un formalisme relevant de la conception pour en dériver une spécification. Nous avons donc utilisé les scénarios. Bien qu'ils aient une formulation assez vague en OMT et qu'ils soient considérés comme secondaires par rapport aux autres modèles, ils relèvent cependant plus de l'étape de spécification.

- Les concepts ne sont pas clairement définis. Un premier travail a été de les clarifier. Cependant, pour certains d'entre eux, nous avons dû les étendre afin de préciser leur

utilisation (événement avec paramètre, condition de déclenchement, échange d'événements, etc..., cf. chapitre 3).

- La sémantique des diagrammes E/T n'est pas toujours claire. Par exemple, que faire d'un événement survenant alors qu'aucune condition sur la transition n'est satisfaite ? Plusieurs solutions sont envisageables :

  - Celle développée par cette étude fait l'hypothèse qu'un événement ne peut apparaître que si la condition de la transition est vraie. Cela suppose qu'il existe un mécanisme de vérification des conditions de déclenchement.
  - Une deuxième solution possible est de prévoir dans le diagramme E/T tous les cas (y compris les cas d'erreur) et d'assurer pour chaque événement qu'il y a toujours une et une seule transition dont la condition est vraie. L'inconvénient est que les diagrammes deviennent vite illisibles.

- Certaines informations ne sont pas exprimables avec les notations définies d'où l'introduction de nouveaux concepts (diagramme associé à l'administrateur, cf. chapitre 3).

Ces points ayant été clarifiés, des règles précises permettant le passage en B des concepts présentés au chapitre 3 ont été données.

De plus, deux solutions pour passer du modèle dynamique à une spécification B ont été proposées : par génération d'obligations de preuve supplémentaire ou par le raffinement.

À titre d'illustration de l'application des règles de traduction, trois études de cas complètes sont données en annexe.

Chapitre 6

CONCLUSION

Pour conclure le travail présenté, nous nous proposons, dans ce chapitre, de résumer notre contribution et de la positionner par rapport aux travaux de recherche étudiés dans l'état de l'art (cf. chapitre 2). Nous terminons en évoquant les principales perspectives de ce travail de recherche.

I. Bilan de la contribution

Comme mentionné dans l'introduction, l'objectif de cette thèse était de proposer une solution pour l'intégration des méthodes semi-formelles et formelles.

Nous avons d'abord étudié de manière approfondie les différents concepts du modèle d'objets et du modèle dynamique existants dans les différentes méthodes (cf. chapitre 3).

Pour l'aspect statique, on retrouve souvent parmi les différentes méthodes les mêmes concepts liés au diagramme d'objets sous différents noms. Nous avons donc sélectionné un ensemble de concepts assez riche afin de modéliser de façon fidèle le monde réel. Au cours de cette sélection, nous avons éclairci les points restés vagues afin de préciser le cadre d'utilisation des concepts. Pour cela, nous avons examiné en détail la sémantique de ces concepts avec éventuellement des choix d'interprétation ce qui nous a amené parfois à les compléter. À titre

d'exemple, les concepts d'association fixe, d'entité associative, d'opérations associées aux entités et associations ont été précisés et étendus.

Pour l'aspect dynamique, devant la diversité des modèles et du fait qu'à l'intérieur de chaque modèle les concepts utilisés diffèrent du point de vue sémantique, une démarche d'extraction de concepts communs n'était pas envisageable. Aussi les diagrammes états/transitions et les scénarios du modèle dynamique de la méthode OMT ont-ils été choisis en raison de la large diffusion de cette méthode. Nous avons alors appliqué aux concepts liés au modèle dynamique de OMT la même démarche d'éclaircissement, de précision et d'extension (par exemple pour les concepts d'événement, d'état, de condition, ...). Notons qu'il a été plus facile de préciser la sémantique des concepts du diagramme d'objets que celle du diagramme états/transitions car la sémantique des concepts du diagramme d'objets est déjà assez bien définie dans les méthodes semi-formelles. Cependant, dans les deux cas, nous avons dû introduire de nouvelles notions afin d'être en mesure de représenter certaines situations non exprimables auparavant. Ainsi, nous avons introduit la notion de diagramme associé à l'administrateur d'une entité pour spécifier les opérations d'entité. Toujours dans cette démarche, nous avons établi la relation entre les diagrammes de suivi d'événements et les diagrammes états/transitions. Nous avons également introduit différents types de scénarios permettant de classifier les constructions possibles des diagrammes états/transitions.

Puis (chapitre 5), nous avons proposé des règles de traduction du modèle d'objets et du modèle dynamique en spécification B. Notre principale motivation n'était pas uniquement d'obtenir une traduction du modèle semi-formel de départ mais aussi d'aboutir à une spécification formelle à la fois fidèle au modèle semi-formel de départ, modulaire et traçable. La traduction du modèle d'objets a consisté en la formalisation des concepts et en la génération des opérations de base. Des règles de modularisation de la spécification ont ensuite été proposées. Concernant la traduction du modèle dynamique, deux étapes sont apparues nécessaires : la génération des opérations à partir des scénarios et des diagrammes états/transitions et la répartition des opérations générées dans les machines obtenues à partir de la traduction du modèle d'objets.

Remarquons que l'étude des concepts des modèles semi-formels et leur traduction en spécification B ne s'est pas effectuée en deux étapes complètement séparées. En effet, la sémantique des concepts n'a été complètement définie qu'au cours de l'élaboration des règles de passage en spécification formelle B. Cette formalisation présente donc aussi un intérêt pour

l'étude des méthodes semi-formelles, indépendamment de la dérivation en spécifications formelles.

En conclusion, le résultat de ce travail peut d'une part être utilisé comme guide méthodologique pour passer d'une spécification semi-formelle à une spécification formelle bien structurée. Dans cette optique, il peut aussi aider à l'apprentissage des méthodes formelles pour les personnes familières avec les méthodes semi-formelles et réciproquement. Il nous a permis d'autre part de clarifier les concepts du modèle d'objets et du modèle dynamique. Enfin, il donne les bases pour le développement d'un outil d'aide à la spécification formelle, dont nous avons conçu un prototype.

II. Positionnement de la contribution

Dans le chapitre 2 de cette thèse, nous avons proposé différents critères, à la fois quantitatifs et qualitatifs, pour l'étude des différents travaux existants dans le domaine. Afin de positionner précisément cette thèse, nous allons lui appliquer ces critères.

II.1. Application des critères quantitatifs

II.1.1. L'aspect statique

Nous reprenons le tableau de synthèse de l'aspect statique (figure 2.1) que nous complétons en indiquant les concepts pris en compte par ce travail.

Concepts / Articles	Entité (Classe)	Attribut d'entité	Association (Relation)	Attribut d'association	Association n-aires	Agrégation	Héritage	Contraintes graphiques	Opérations de base
[NAG 94]	T	T	T	T	P	T	T⁻		A
[LAN 94b]	T	T	T				T⁻		
[HAD 96]	T		T					T⁻	T⁻
[MAL 98]	T	T	T						
[DUP 97]	T	T	T	T	T	T	T⁻		A
[FRA 97]	T	T	T			T	T⁻		A
[SHR 97]	T	T	T	T		T	T⁻		
[NGU 98]	**T**	**T**	**T**	**T**	**T**	**T⁻**	**T⁻**	**T**	**T⁻**

Figure 6.1. *Tableau de synthèse de l'aspect statique*

Rappel de la légende : :

T	: le concept est traité complètement
T -	: le concept n'est que partiellement traité
A	: le concept est simplement abordé
P	: le concept n'est pas traité mais pourrait l'être dans la démarche adoptée

Ce tableau appelle plusieurs remarques. Tout d'abord, nous nous sommes attachée à étudier le plus grand nombre de concepts possibles. Concernant le traitement des associations n-aires, nous rappelons que, dans notre approche, une association est définie seulement entre deux entités : nous faisons correspondre au concept d'association n-aires celui d'entité associative (cf. chapitre 3). Les solutions proposées dans la littérature pour la formalisation du concept d'héritage permettent seulement un héritage d'attributs. Concernant l'héritage d'opérations, nous avons proposé un moyen de le simuler. Ce point nécessiterait une étude plus approfondie. Pour ce qui concerne les opérations de base, seul l'effet local (contrainte de cardinalité, association fixe, ...) a été pris en compte pour la génération de ces opérations. Le traitement de l'effet global (contrainte entre plusieurs associations, ...) n'a pas été traité. Le concept d'agrégation, qui a été considéré comme étant un cas particulier d'association, mériterait une étude plus approfondie.

II.1.2. L'aspect dynamique

Pour les raisons données au chapitre 5, notre choix d'un modèle semi-formel pour l'aspect dynamique s'est porté sur OMT. Deux travaux parmi ceux étudiés dans le chapitre 2 [LAN 94b] et [DUP 98], s'intéressent également au modèle dynamique de OMT. Nous allons donc situer notre travail par rapport à ces derniers. Le tableau suivant indique les différents concepts examinés pour chaque travail.

Concepts / Articles	Événement	État	Condition	Opération	Généralisation diagrammes E/T	Concurrence d'objets	Échange d'événements
[DUP 98]	T	T	T	T	T	T	T
[LAN 94b]		T	T	T			
[NGU 98]	**T**	**T**	**T**	**T**		**T**	**T**

Figure 6.2. *Tableau de synthèse de l'aspect dynamique*

Nous avons essayé de traiter la plupart des concepts du modèle dynamique définis dans la méthode OMT. Notons que les événements dans [DUP 98] et [NGU 98] ne sont pas traités de la même manière. [DUP 98] gère explicitement les événements au moyen de classes d'événements, alors que notre approche consiste à spécifier l'effet de l'événement sur les données du système à l'aide d'opérations.

II.1.3. La modularisation

Comme la plupart des travaux étudiés, nous avons pris en compte la modularisation. Cependant, nous avons adopté des critères différenciant notre approche des autres :

- La spécification obtenue doit rester le plus fidèle possible au modèle semi-formel de départ afin de rendre possible le passage dans les deux sens (semi-formel vers formel et réciproquement), ce qui facilite certainement l'apprentissage des méthodes formelles aux personnes familières avec le domaine semi-formel et vice-versa, ainsi que la maintenance.
- L'unité de conception en B étant la machine abstraite, nous avons veillé à trouver un équilibre entre la quantité d'information contenue dans chaque machine et le nombre de machines créées.

II.2. Application des critères qualitatifs

Un certain nombre de problèmes ont été soulevés lors de l'étude des différents travaux dans le chapitre 2. Certains d'entre eux ont été résolus facilement grâce au choix de B :

- gérer l'ensemble des objets existants et l'ensembles des objets possibles dans une même structure (i.e. *machine abstraite*).
- définir des identités d'objet indépendamment de leur valeur et donc du concept de clé.
- considérer le concept d'association à un haut niveau d'abstraction (sans le réduire à un produit cartésien) en utilisant toute la puissance des opérateurs relationnels binaires de B.
- prendre en compte les preuves de cohérence de la spécification grâce au générateur d'obligations de preuve et au démonstrateur.

D'autres problèmes ont été résolus lors de l'élaboration de nos règles :

- La manière de spécifier les états par un prédicat portant sur les variables représentant les attributs entraîne un changement d'état automatique lors de chaque modification des valeurs des attributs.
- La manière de modulariser la spécification dans notre approche ne produit ni trop de machines comme, à notre avis [NAG 94], ni de machines pouvant inclure trop d'information comme, à notre avis [LAN 94].
- Nous avons proposé des règles permettant de générer les obligations de preuve portant sur la vérification de l'état d'arrivée de chaque opération. En outre, une autre technique permettant d'obtenir ces obligations automatiquement lors des preuves de raffinement a été également proposée.

III. Perspectives

Outre les points qui n'ont pas été traités ou qui ne l'ont été que partielement, partant de ce travail de recherche, plusieurs développements sont possibles :

- Bien entendu, la réalisation de l'outil de traduction d'une spécification semi-formelle vers une spécification formelle doit être poursuivie (seul un prototype ayant été réalisé).

- Il serait envisageable d'étudier le raffinement de la spécification obtenue en une implémentation. Ce travail pourrait déboucher sur l'intégration avec un système de gestion de bases de données (à objets ou relationnelles). Notons également que le raffinement pourrait être davantage utilisé dans la spécification elle-même (cf. remarque du chapitre 5, partie III.5)

- Il serait également intéressant d'intégrer à ce travail la définition des opérations de modification en cascade prenant ainsi en compte leur effet global sur le système puis de les traduire dans un langage type SQL.

- Ce travail peut aussi aider au développement de l'approche "interprétée" [FAC 95] qui utilise une formalisation explicite du méta-modèle semi-formel.

Nous espérons ainsi avoir contribué à l'intégration des méthodes semi-formelles et formelles de conception des systèmes d'information.

BIBLIOGRAPHIE

[ABR 80] ABRIAL J.R.,
The Specification Language Z : Basic Library,
Oxford University, Programming Research Group, 1980.

[ABR 96] ABRIAL J.R.,
The B Book : Assigning Programs to Meanings,
Cambridge University Press, 1996.

[BOO 94] BOOCH G.,
Object-Oriented Analysis and Design with Applications,
Benjamin/Cummings, Menlo Park, CA, Second edition, 1994.

[BUG 95] *The BUG Book*,
Publication privée du "B User Group", Octobre 1995.

[CAS 94] CASTELLANI X.,
Les administrateurs des classes d'objets,
Ingénierie des Systèmes d'Information, vol.2-n°5, 1994.

[CHE 76] CHEN P.P.,
The Entity-Relationship Model : Toward a Unified View of Data,
ACM Transactions on Database Systems, 1(1), Mars 1976.

[CLA 97] CLARK T., EVANS A.,
Foundations of the Unified Modeling Language
In Proceedings of The Second Northern Formal Methods Workshop. Spring-Verlag, 1997.

[COD 71] CODASYL,
Feature Analysis of Generalized Data Base Management Systems,
Sys. Commitee Tech. Report, ACM, New York, 1971.

[COL 94] COLEMAN D., ARNOLD P., BODOFF S., DOLLIN C., GILCHRIST H., HAYES F., JEREMAES P.,
Object-Oriented Development : The Fusion Method,
Prentice Hall, Englewood Cliffs, NJ, Object-Oriented Series edition, 1994.

[COR 96] B-CORE,
B-Toolkit Release 3.2. Manual,
Oxford, U.K., 1996.

[DAY 88] DAYAL U.,
The HIPAC Project : Combining Active Databases and Timing Constraints,
SIGMOD Record, Vol. 17, n° 1, Mars 1988.

[DIC 91] DICK J., LOUBERSAC J.,
Integrating Structured and Formal Methods : a visual approach to VDM,
3nd International Conference ESEC'91, pages 37-59. Milan, Octobre 1991, Springer-Verlag.

[DIG 96] DIGILOG groupe STERIA,
Atelier B - Manuel de référence,
DIGILOG, BP 16000, 13791 Aix-en-Provence Cedex 3 France, 1996.

[DUK 91] DUKE R., KIN P., ROSE G., SMITH G.,
The object-Z Specification Language : Version 1,
Rapport de recherche 91-1. Depertement of Computer Science. University of Queensland, Austria, Software Verification Research Centre, Avril 1991.

[DUP 97] DUPUY S., LEDRU Y., CHABRE-PECCOUD M.,
Integrating OMT and Object-Z,
In K. Lano (eds) A. Evans, editor, Proceedings of BCS FACS/EROS ROOM Workshop, tecnical report GR/K67311-2, Dept. of Computing, Imperial College, 180 Queens Gqte, London, UK, Juin, 1997.

[DUP 98] DUPUY S., LEDRU Y., CHABRE-PECCOUD M.,
Translating the OMT dynamique model into Object-Z,
Proceeding of 11th International Conference of Z Users, Berlin, Germany, 24-26 Septembre 1998.

[EVA 97] EVANS A., FRANCE R., LANO K., RUMPE B.,
The UML as a Formal Modeling Notation,
OOPSLA 1997.

[FAC 95] FACON P., LALEAU R.,
Des spécifications informelles aux spécifications formelles : compilation ou interprétation?,
Actes du 13ème congrès INFORSID, Grenoble, Juin 1995.

[FAC 96a] FACON P., LALEAU R., NGUYEN H. P.,
Mapping Object Conceptual Diagrams into B Specifications,
Methods Integration Workshop, Leeds, RU, Mars 1996 (in coll. eWiCS, Springer Verlag, 96).

[FAC 96b] FACON P., LALEAU R.,
Des modèles d'objets aux spécifications formelles ensemblistes,
Ingénierie des Systèmes d'Information, vol.4-n°2, 1996.

[FAC 96c] FACON P., LALEAU R., NGUYEN H. P.,
Dérivation de spécifications formelles B à partir de spécifications semi-formelles de systèmes d'information,
Proceedings of the "First B Conference", Nantes, France, Novembre 1996.

[FAC 98] FACON P., LALEAU R., NGUYEN H. P.,
The Invoicing System Problem : From OMT Diagrams to B Specifications,

International Workshop on Comparing Systems Specification Techniques "What questions are prompted by ones particular method of specification ?", Nantes, France, Mars 1998.

[FRA 97] FRANCE R. B., BRUEL J. M., LARRONDO-PETRIE M. M., GRANT E., *Rigorous Object-Oriented Modeling : Integrating Formal and Informal Notations,*
Proceedings of the 6th International AMAST Conference, Décembre 1997.

[FRE 96] FREIRE JUNIOR J.,
Pouvoir d'expression de Modèles Orientés Objet,
Ingénierie des systèmes d'information, Volume 4, n° 2/1996, pages 219-237.

[GOG 83] GOGUEN J.A., MESEGUER J., PLAISTED D.,
Programming with parameterized abstract objects in OBJ,
Theory and Practice of Software Technology, North-Holland ed., 1983.

[GRA 94] *Graphtalk - Méta Modélisation,*
Interface de programmation. Parallax, 1994.

[GUT 85] GUTTAG J.V., HORNING J.J., WING J.M.,
The Larch Family of Specification Languages,
IEEE Software, 2(5), 1985.

[HAB 93] HABRIAS H.,
Introduction à la spécification,
Masson, 1993.

[HAB 95] HABRIAS H.,
Les spécifications formelles pour les systèmes d'information : Quoi ? Pourquoi ? Comment ?,
Ingénierie des systèmes d'information. Volume 3 - n° 2, pages 205 - 253, Mars 1995.

[HAB 97] HABRIAS H.,
Dictionnaire encyclopédique du Génie Logiciel,
Masson, 1997.

[HAD 96] HADJ-RABIA N., HABRIAS H.,
Formal specification from NIAM model : A Bottom Up Approach,
ISCIS XI (The Eleventh International Symposium on Computer and Information Sciences), 6-8 Novembre 1996, Antalya, Turkey.

[HAR 87] HAREL D.,
Statecharts : A visual formalism for complex systems,
Science of Computer Programming, vol 8, n° 3, pages 231-274, 1987.

[INT 98] *International Workshop on Comparing Systems Specification Techniques "What questions are prompted by ones particular method of specification ?",*
Nantes, France, Mars 1998.

[JAC 92] JACOBSON I.,
Object oriented software engineering,
Addison-Wesley, 1992.

[JON 90] JONES C.B.,
Systematic Software Development using VDM,
2nd edition, Prentice Hall Int., 1990.

[LAN 94a] LANO K., HAUGTHON H.,
Object-Oriented Specification Case Studies,
Prentice Hall Int., 1994.

[LAN 94b] LANO K., HAUGTHON H.,
Improving the Process of System Specification and Refinement in B,
6th Refinement Workshop, D. TILL Editor, Springer Verlag, Workshops in Computing, 1994.

[LAN 96] LANO K., GOLDSACK S.,
Integrated Formal and Object-Oriented Methods : The VDM++ Approach,
A. Bryant & L.Seemmens Editor, Springer Verlag, Method Intergration Workshop, Leeds, RU, Mars 1996.

[LAR 94] LARSEN P. G., PLAT N., TOETENEL H.,
A Formal Semantics of Data Flow Diagrams,

Formal aspects of computing, vol.6-n°6, December, 1994.

[LED 95] LEDRU Y., CHIARAMELLA Y.,
Integrating and Teaching ZSP and Z,
Proceeding of "Z twenty years on - what is its future ?", Nantes, France, 1995.

[LED 96] LEDRU Y.,
Complementing semi-formal spcification with Z,
Proceedings of the 11th Knowledge-Based Software Engineering, IEEE Computer Society Press, September, 1996.

[MAL 98] MALIOUKOV A.,
An Object-Based Approach to the B Formal Method,
B'98 : Recent Advances in the Development and Use of the B Method, 2nd International B Conference, Montpelier, France, April 1998, LNCS n° 1393.

[MAM 98] MAMMAR A.,
Utilisation de graphes pour la spécification de formules relationnelles,
Rapport de stage du DEA IIE-INT-EVE “Conception des Systèmes Informatiques Avancés”, Evry, Septembre 1998.

[MEY 88] MEYER B.,
Object-Oriented Software Construction,
Prentice Hall, Hemel Hempstead, UK, 1988.

[MET 96] *Methods Integration Workshop,*
Leeds, RU, Mars 1996.

[MIS 97] MISIÉ V. B., MOSER S.,
Formal Approach to Metamodeling : A Generic Object-Oriented Perspective,
16th International Conference on Conceptual Modeling (E/R 97), Los Angeles, California, USA, Novembre 1997, LNCS n° 1331.

[MON 97] MONGE F.,
Formalisation du méta modèle des méthodes graphiques d'analyse et conception orientées objet,

Rapport de stage du DEA IIE-INT-EVE “Conception des Systèmes Informatiques Avancés”, Evry, Septembre 1997.

[NAN 92] NANCI D., ESPINASSE B., COHEN B., HECKENROTH H.,
Ingénierie des systèmes d'information avec Merise,
Sybex Int., 1992.

[NAG 94] NAGUI-RAISS N.,
A Formal Software Specification Tool using the Entity-Relationship Model,
Proceeding of the 13th International Conference on the Entity-Relationship Approach, LNCS, Springer-Verlag, Manchester, RU, Décembre 1994.

[NGU 95] NGUYEN H. P.,
Formalisation des modèles sémantiques des systèmes d'information,
Rapport de stage du DEA, IIE-INT-EVE "Conception des Systèmes Informatiques Avancés", Evry, Juin, 1995.

[POL 93] POLACK F., WHISTON M., MANDER K.,
The SAZ Project: Integrating SSADM and Z,
Proceeding of the International Symposium Formal Methods Europe, Odense, Danemark, Avril 1993.

[RUM 91] RUMBAUGH J., BLAHA M., PREMERLANI W., EDDY F., LORENSEN W.,
Object-Oriented Modelling and Design,
Prentice Hall : New Jersey, 1991.

[SCH 91] SCHEWE K.D., SCHMIDT J.W., WETZEL I.,
Specification and refinement in an integrated database application environment,
Proceeding of the 4th International Conference VDM'91, Noordwijkerhout Holland, LNCS, Springer-Verlag, Octobre 1991.

[SHL 92] SHLAER S., MELLOR S. J.,
Object lifecycles : Modeling the World in States,
Yourdon, Prentice-Hall, 1992.

[SHR 97] SHROFF M., FRANCE R. B.,
Towards a Formalization of UML Class Structures in Z,
COMPSAC'97, 1997.

[SMI 77] SMITH J.M., SMITH D.C.P.,
Database Abstractions : Aggregation and Generalization,
ACM TODS, 2(2), Juin 1977.

[SPI 92] SPIVEY J. M.,
The Z Notation : A Reference Manual,
Prentice Hall, Englewood Cliffs, NJ, Second edition, 1992.

[TAR 83] TARDIEU H., ROCHFELD A., COLLETTI R.,
La méthode Merise - Tome 1 : Principe et outils,
Les Éditions d'Organisation, 1983.

[UML 97] The UML Group, www.rational.com/uml,
Unified Modeling Language. Version 1.1,
Rational Sofware Corporation, Santa Clara, USA, Juillet 1997.

[WIR 90] WIRFS-BROCK R. and WILKERSON B.,
Designing object oriented sofware,
Prentice Hall, 1990.

ANNEXES

Les annexes ci-après présentent trois études de cas. Elles ont pour but de montrer l'applicabilité de nos règles de dérivation de spécifications formelles à partir de spécifications semi-formelles. Ces études de cas ont été validées par l'Atelier B. Cette phase de validation nous semble importante pour plusieurs raisons : tout d'abord elle permet de vérifier la cohérence de la spécification finale ; ensuite elle fait partie intégrante du processus de dérivation. En effet, c'est dans la phase de validation qu'apparaissent les manques d'information, le caractère contradictoire de certaines contraintes. L'apparition de ces défauts de spécification oblige à modifier les spécifications formelles.

La première étude de cas intitulée "Système de Commandes" a été proposée à la conférence [CCS 98]*. Le but de cette conférence était de comparer les différentes techniques de spécification. À cette ocasion, différentes approches semi-formelles et / ou formelles ont été adoptées pour traiter le cas. Nous présentons ici le résultat de l'application de notre approche. Nous comparons notre spécification B à celle de [DIA 98]** du point de vue de la lisibilité et de la validation.

Le sujet de la deuxième étude de cas, Gestion d'un Passage à Niveau, provient de [DUP 98] où les auteurs aboutissent à une spécification en Object-Z. Le traitement de ce cas nous a semblé intéressant pour deux raisons. D'une part, il nous a permis de comparer deux spécifications, celle de [DUP 98] en Object-Z et la nôtre en B, obtenues à partir de la même spécification semi-formelle de départ, en OMT. D'autre part, dans la mesure où il ne s'agit pas d'une application de gestion de base de données, il nous a permis de mieux cerner le domaine d'applicabilité de nos règles.

Enfin, nous présentons une dernière étude de cas, Gestion d'un Club Vidéo. Il s'agit d'un cas réel à la fois complet et complexe qui nous a permis d'appliquer la quasi totalité des règles proposées.

Pour chacune de ces trois études, nous adoptons la convention typographique suivante : dans les machines B obtenues, nous mettons en italique les parties qui ne sont pas générées automatiquement.

Annexe 1

SYSTÈME DE COMMANDES

Dans cette partie, nous reprenons l'étude de cas proposée par H. Habrias à l'occasion de la conférence [CCS 98]* dont l'intitulé est : "Comparing Systems Specification Techniques - What questions are prompted by ones particular method of specification ?". Deux cas ont été proposés. Le but de la conférence était de savoir quelles questions sont posées par la méthode utilisée pour l'étude.

Nous nous intéressons uniquement à l'un des deux cas qui nous apparaît le plus complet.

I. Présentation du cas

Le cas soumis à étude s'énonce en ces termes :

« Il s'agit de facturer des commandes. Facturer est changer l'état d'une commande (le faire passer de "En attente" à "Facturée").

Sur une commande, on a une et une seule référence à un produit commandé en une certaine quantité commandée. La quantité peut être différente d'une commande à l'autre. La même référence peut être commandée sur plusieurs commandes. Une commande a son état changé en "Facturé" si la quantité commandée est égale ou inférieure à la quantité en stock pour la référence du produit commandé.

---

*[CCS 98], International Workshop on : Comparing Systems Specification Techniques *"What questions are prompted by ones particular method of specification ?"* March 26-27, 1998, Nantes (France)

Vous avez à prendre en compte les entrées des :

- nouvelles commandes,
- des annulations de commandes,
- des entrées de quantités dans le stock.

Peut-être considérerez-vous que ce texte est incomplet. Le but de cette étude de cas est de savoir quelles questions sont déclenchées par votre (vos) méthode(s) favorite(s). Vous proposerez différentes "solutions" (qui expriment des besoins cohérents) et vous expliciterez comment votre (vos) méthode(s) vous ont amené à proposer ces "solutions".

Attention !
N'élargissez pas le domaine. Par exemple, ne faites pas de la gestion de stock (quand, selon quelle quantité économique d'achat, etc.), n'ajoutez pas des informations comme "catégorie de produits", "catégories de clients", "modes de règlement", "domiciliation bancaires", etc. »

II. Spécification semi-formelle

II.1. Le modèle d'objets

Afin de montrer comment le diagramme d'objets a été conçu, nous allons prendre chaque phrase de l'énoncé.

- "Sur une commande, on a une et une seule référence à un produit commandé en une certaine quantité commandée. La quantité peut être différente d'une commande à l'autre". Nous définissons donc :
  - deux entités Commande et Produit,
  - une association Référence entre ces deux entités, avec la contrainte de cardinalité (1,1) pour Commande,
  - un attribut Quantité_Commandée de l'association Référence.
- "La même référence peut être commandée sur plusieurs commandes" : la contrainte de cardinalité (0,n) entre Référence et Produit.

- " ... si la quantité commandée est inférieure ou égale à la quantité en stock pour la référence du produit commandé" : nous définissons l'attribut Quantité_Stock pour l'entité Produit qui détermine la quantité en stock pour chaque produit.

Nous obtenons donc le diagramme d'objet suivant :

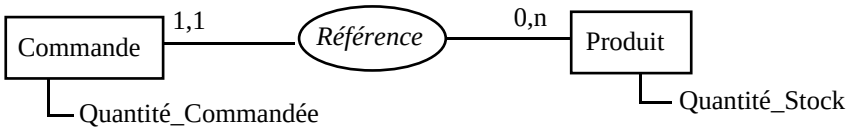


Figure 1. *Le diagramme d'objets du système de commandes*

II.2. Le modèle dynamique

Afin de compléter la description de la dynamique (diagrammes E/T et scénarios), l'analyse du sujet nous a amené à poser les questions suivantes :

Questions :

1. Quand l'opération de facturation est-elle déclenchée ?
   (autrement dit, quel type d'événement va déclencher la transition ?)
   Trois solutions sont envisageables :
   1.1. périodiquement. Ex : tous les soirs, tous les deux jours, ...
   1.2. aléatoirement (à la demande de l'utilisateur du système)
   1.3. à chaque fois qu'il y a des entrées de quantité dans le stock, des nouvelles commandes, des annulations de commande facturée.
2. Dans quel état une commande peut-elle être supprimée (En_Attente, Facturée) ?
3. A quel moment est fait la diminution du stock pour une commande donnée ?

Propositions :

1. Nous avons choisi de traiter le troisième type d'événement (1.3), plus intéressant à spécifier car permettant d'optimiser la facturation.
2. En ce qui concerne la suppression d'une commande, nous supposons que c'est possible quel que soit l'état de la commande : "Commande En_Attente" ou "Commande Facturée".
3. Nous proposons que la diminution du stock soit faite quand la commande est facturée.

Nous avons les événements externes suivants :

(1) "Nouvelle commande" pour un produit donné en une certaine quantité, noté *Création_Cd.*
(2) "Annulation d'une commande". Selon que la commande est dans l'état "En_Attente", ou dans l'état "Facturée", l'événement est de type *Annul_Cd_En_Attente,* ou *Annul_Cd_Facturée.*
(3) "Entrée de stock" pour un produit donné en une certaine quantité, noté *Ajout_Quantité.*

De manière informelle, chaque événement (i) (i $\in$ (1..3)) entraîne l'opération (i) suivante :

(1) Si la quantité en stock est suffisante alors on facture la commande. Ce qui entraîne une diminution du stock.
(2) Si c'est une commande dans l'état "Facturée" alors le stock du produit concerné est augmenté, ce qui peut entraîner la facturation d'un ensemble maximal de commandes en attente.
(3) L'augmentation de stock peut entraîner la facturation d'un ensemble maximal de commandes en attente.

Précisons l'opération "Facturation d'un ensemble maximal de commandes en attente concernant un produit donné" :
Maximal signifie que si on ajoute une autre commande dans cet ensemble, on ne peut plus satisfaire l'ensemble. Notons que plusieurs ensembles maximaux sont possibles. À ce stade de conception, on n'est pas obligé de préciser lequel est choisi. L'opération "Facturation" décrite est une opération d'entité car l'ensemble des commandes à facturer n'est pas encore déterminé.
Dans ce cas, il est nécessaire de définir un diagramme associé à l'administrateur de l'entité "Commande" qui va choisir un ensemble de commandes vérifiant les 3 critères de sélection : les commandes sont en état "En attente" ; la somme des quantités commandées de ces commandes est inférieure ou égale à la quantité en stock du produit ; c'est un ensemble maximal.
L'opération consiste alors à facturer chaque commande de cet ensemble : changer son état et diminuer le stock du produit.

En utilisant les notations de OMT, nous obtenons le diagramme de suivi d'événements suivant :

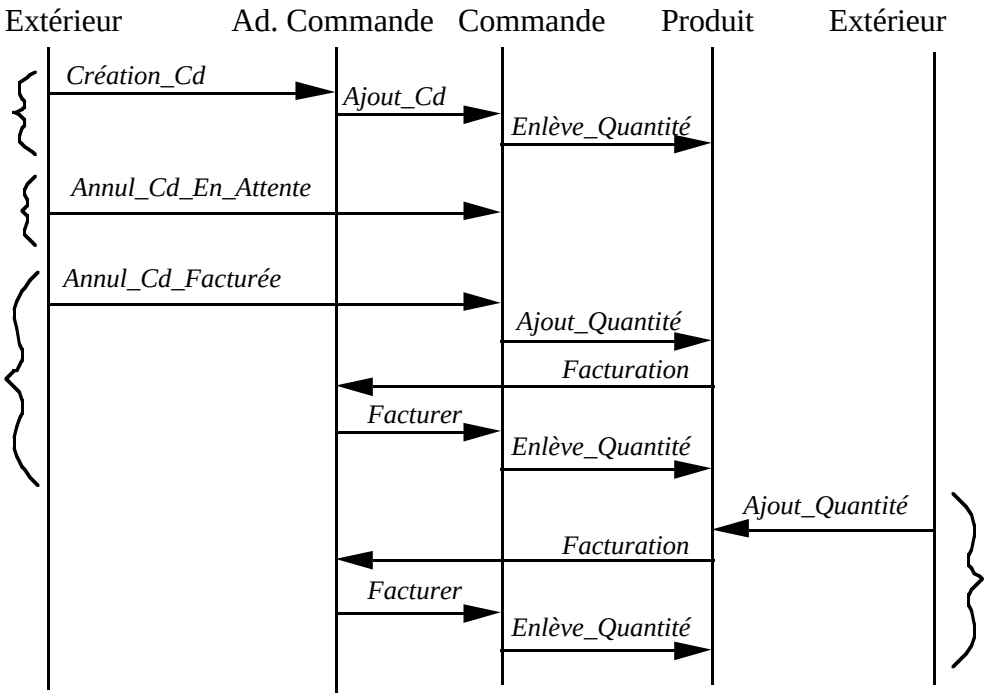
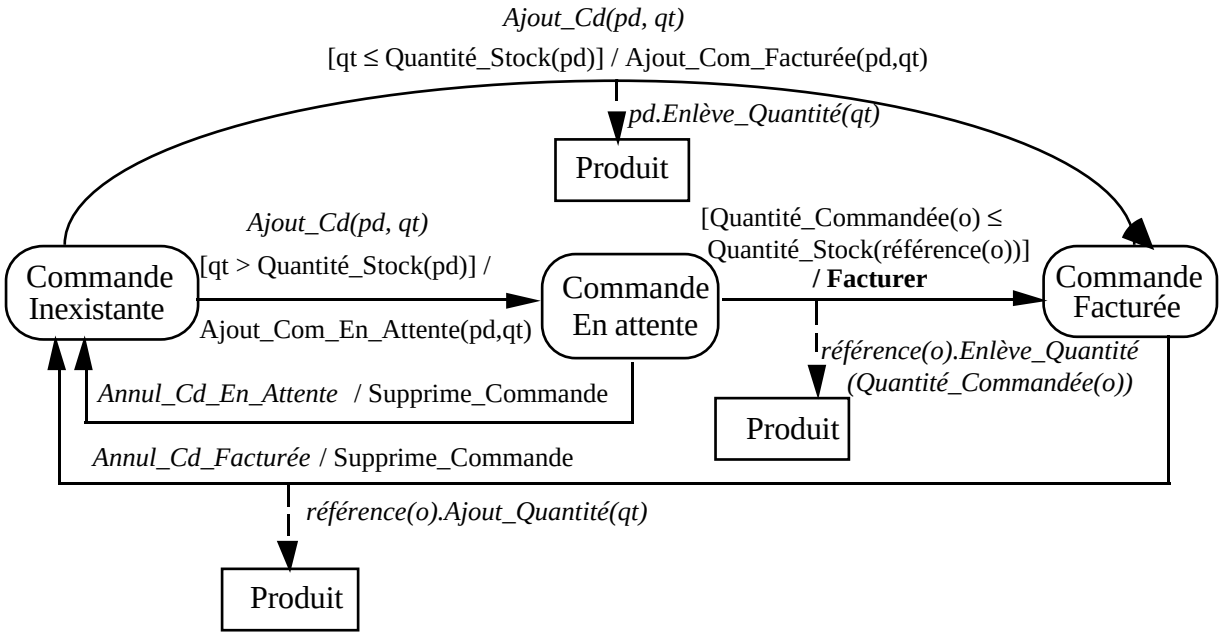


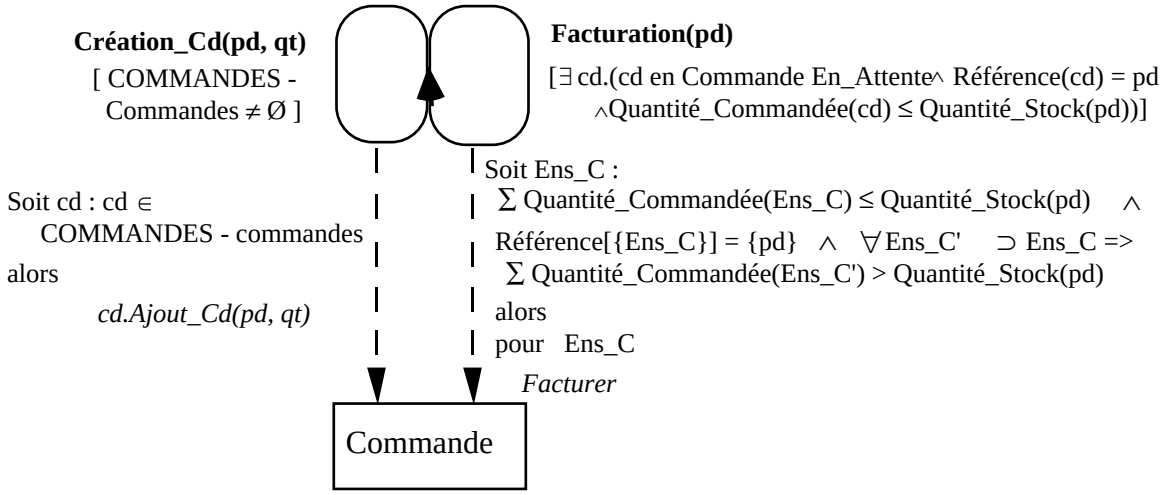
Figure 2. *Le diagramme de suivi d'événements*

Et les diagrammes E/T suivants :

a. Diagramme E/T de Commande :



b. Diagramme E/T de l'administrateur de Commande :



c. Diagramme E/T de Produit :

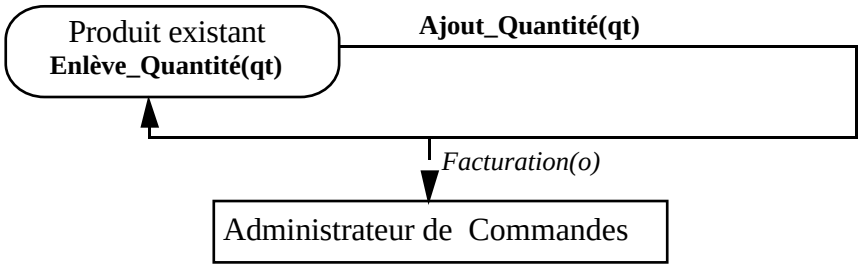


Figure 3. *Les diagrammes E/T*

Remarque

Notons que la stratégie choisie (facturation automatique d'un ensemble maximal de commandes à chaque augmentation de stock) impose qu'il faut définir la contrainte d'intégrité suivante (qui ne peut pas s'exprimer graphiquement) :

CI1: il n'existe pas de commande en attente pouvant être facturée (si une commande est en attente cela implique que sa quantité commandée est supérieure à la quantité en stock du produit concerné).

II.2.1. Scénario pour la création d'une commande

Création_Cd est un événement externe vers le diagramme associé à l'administrateur de Commande (figure 3b) qui doit d'abord choisir une nouvelle commande et ensuite l'ajouter dans l'ensemble des commandes. Nous avons l'événement interne Ajout_Cd du diagramme de l'administrateur de Commande vers le diagramme de Commande (figure 3a).

Si la quantité en stock est suffisante, la nouvelle commande sera facturée. Ensuite, nous devons diminuer la quantité en stock du produit commandé. Cela est représenté par l'événement interne Enlève_Quantité du diagramme E/T de l'entité Commande vers Produit (figure 3b, 3c).

Si la quantité en stock est insuffisante, la commande sera dans l'état Commande_En_Attente.

II.2.2. Scénario pour l'annulation d'une commande en attente

Annul_Cd_En_Attente est un événement externe vers l'entité Commande (figure 3a). La commande à annuler est un paramètre implicite de cet événement. Cet événement interne n'entraîne pas d'autre événement.

II.2.3. Scénario pour l'annulation d'une commande facturée

Annul_Cd_Facturée est un événement externe vers l'entité Commande (figure 3a). La quantité en stock du produit concerné est donc augmentée. Nous avons l'événement interne Ajout_Quantité de l'entité Commande vers l'entité Produit (figure 3c) ce qui peut entraîner la Facturation d'un ensemble maximal de commandes en attente. L'administrateur de Commande doit d'abord choisir un ensemble possible selon trois critères exprimés (figure 3b) et ensuite facturer chaque commande de cet ensemble. Avec la facturation d'un ensemble de commandes en attente, nous devons diminuer la quantité en stock du produit concerné. Cela est représenté par l'envoi de l'événement Enlève_Quantité vers le diagramme de Produit (figure 3c).

II.2.4. Scénario pour l'ajout d'une quantité en stock

Ajout_Quantité est un événement externe vers l'entité Produit (figure 3c). Cet événement entraîne l'événement interne Facturation vers le diagramme de l'administrateur de Commande (figure 3b). Comme le cas précédent, l'administrateur de Commande doit choisir

un ensemble de commandes en attente selon les critères exprimés (figure 3b) et ensuite les facturer. Cette facturation entraîne la diminution de la quantité en stock du produit concerné.

III. Spécification formelle

En appliquant les règles présentées dans le chapitre 5, nous obtenons :

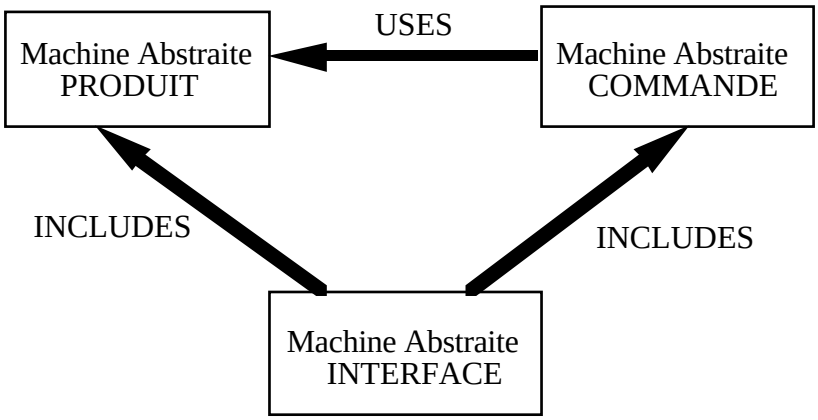
- Une machine abstraite pour l'entité Produit qui se compose de l'ensemble des produits existants et de la variable représentant l'attribut Quantité_Stock. Seule l'opération de base Change_Quantité qui consiste à changer la quantité en stock est à considérer, les opérations de création ou de suppression d'un produit ne sont pas demandées dans le sujet.

- Une machine abstraite pour l'entité Commande qui se compose de l'ensemble des commandes existantes et des variables représentant l'attribut Etat_cd, l'association Référence, l'attribut Quantité_Commandée. Notons que l'association Référence est décrite dans cette machine car aucune modification de la valeur des attributs de l'association n'est autorisée dans l'énoncé. De plus, cette association est obligatoire et fixe pour une commande (c'est-à-dire que la création d'une commande entraîne la création d'une occurrence de l'association et que celle-ci ne peut jamais être associée à une autre commande). Afin de pouvoir utiliser les variables de la machine abstraite Produit dans la machine abstraite Commande, il est naturel d'utiliser le lien *Uses* de la machine Commande vers la machine Produit.
  La machine Commande contient également les opérations de base qui permettent d'ajouter, supprimer et facturer une commande, ainsi que des opérations pré-composées d'opérations de base. En effet, considérons l'opération de la machine Interface qui correspond à l'effet du scénario associé à l'événement externe déclencheur Annul_Cd_Facturée. Cette opération consiste à supprimer une commande déjà facturée. Ceci entraîne le changement de quantité en stock du produit concerné et éventuellement la facturation de commandes en attente. Nous remarquons alors que deux opérations de base de la machine Commande (Supprime_Com et Facture_Com) seraient appelées dans l'opération Annul_Cd_Facturée, ce qui est interdit en B. Afin de résoudre ce problème, nous avons pré-composé ces deux opérations de base en une opération élémentaire : Supprime_Facture_Com.

C'est aussi le cas pour l'opération élémentaire Ajoute_Com_Facturée, de l'entité Commande, qui est le résultat de la pré-composition des opérations de base Ajoute_Com_En_Attente et Facture_Com.

- Une machine Interface qui inclut les deux machines précédentes et qui contient les quatre opérations associées aux quatre événements déclencheurs du diagramme de suivi d'événements (cf. figure 2). Notons également que la contrainte d'intégrité CI1 a été traduite par un invariant.

Voici le schéma de la structuration des machines :



Et les machines obtenues sont les suivantes (avec la convention de mettre en italique les parties qui ne sont pas générées automatiquement).

MACHINE Produit
SETS PRODUITS
VARIABLES Produits, Quantite_Stock
INVARIANT Produits $\subseteq$ PRODUITS $\wedge$
Quantite_Stock $\in$ Produits $\rightarrow$ NAT
INITIALISATION Produits, Quantite_Stock := {}, {}
OPERATIONS
Change_Quantite(pd, qt) = /* opération pré-composée */
/ Modifie de qt la quantite en stock du produit pd */*
PRE *pd $\in$ Produits $\wedge$*
qt $\in$ INTEGER $\wedge$
Quantite_Stock(pd) + qt $\in$ NAT

THEN *Quantite_Stock(pd) := Quantite_Stock(pd) + qt*
END
END

MACHINE Commande
USES Produit
SETS COMMANDES ;
ETAT = {En_Attente, Facturee}
VARIABLES Commandes, Etat_cd, Reference, Quantite_Commandee
INVARIANT Commandes $\subseteq$ COMMANDES $\wedge$
Etat_cd $\in$ Commandes $\rightarrow$ ETAT $\wedge$
Reference $\in$ Commandes $\rightarrow$ Produits $\wedge$
Quantite_Commandee $\in$ Commandes $\rightarrow$ NAT
INITIALISATION Commandes, Etat_cd := {}, {} ||
Reference, Quantite_Commandee := {}, {}

OPERATIONS
Ajoute_Commande_Facturee(cd, pd, qt) = /* opération pré-composée */
PRE cd $\in$ COMMANDES - Commandes $\wedge$
pd $\in$ Produits $\wedge$ qt $\in$ NAT
THEN Commandes := Commandes $\cup$ {cd} ||
Reference := Reference $\cup$ {cd $\mapsto$ pd} ||
Quantite_Commandee := Quantite_Commandee $\cup$ {cd $\mapsto$ qt} ||
Etat_cd := Etat_cd $\cup$ {cd $\mapsto$ Facturee}
END ;
Ajoute_Commande_En_Attente(cd, pd, qt) = /* opération de base*/
PRE cd $\in$ COMMANDES - Commandes $\wedge$
pd $\in$ Produits $\wedge$ qt $\in$ NAT
THEN Commandes := Commandes $\cup$ {cd} ||
Reference := Reference $\cup$ {cd $\mapsto$ pd} ||
Quantite_Commandee := Quantite_Commandee $\cup$ {cd $\mapsto$ qt} ||
Etat_cd(cd) := Etat_cd $\cup$ {cd $\mapsto$ En_Attente}
END ;
Supprime_Commande(cd) = /* opération de base*/
PRE cd $\in$ Commandes

```
        THEN        Commandes := Commandes - {cd} ||
                    Etat_cd  := {cd} ⩤ Etat_cd ||  Reference := {cd} ⩤ Reference ||
                    Quantite_Commandee := {cd} ⩤ Quantite_Commandee
        END ;
        Facture_Commande(Cd) =                          /* opération de base*/
        PRE         Cd ⊆ Commandes
        THEN        Etat_cd :=  Etat_cd <⊢ Cd * {Facturee}
        END ;
        Supprime_Facture_Commande(cd, Cd) =             /* opération pré-composée */
        PRE         cd ∈ Commandes ∧
                    Cd ⊆ Commandes ∧
                    cd ∉ Cd
        THEN        Etat_cd  :=  ({cd} ⩤ Etat_cd) <⊢ Cd * {Facturee} ||
                    Commandes := Commandes - {cd} ||
                    Reference := {cd} ⩤ Reference ||
                    Quantite_Commandee := {cd} ⩤ Quantite_Commandee
        END
END
```

```
MACHINE         Interface
INCLUDES        Produit, Commande
INVARIANT       ∀ ce. (ce ∈ Ens_Commande_En_Attente
                => Quantite_Commandee(ce) > Quantite_Stock(Reference(ce)))
                /* correspond à la contrainte d'intégrité CI1 :
                il n'existe pas de commande en attente pouvant être facturée */
DEFINITIONS
        Ens_Commande_Inexistante        ≙ COMMANDES - Commandes  ;
        Ens_Commande_En_Attente         ≙ Etat_cd^-1[{En_Attente}] ;
                /* Ensemble des commandes en état en attente */
        Ens_Commande_Facturee           ≙ Etat_cd^-1[{Facturee}] ;
                /* Ensemble des commandes en état facturée */
        Ens_Commande_Produit(pd)        ≙ Reference^-1[{pd}] ;
                /* Ensemble des commandes concernant le produit pd */
        Facturable(pd, qt)              ≙
```

∃ cc. (cc ∈ (Ens_Commande_En_Attente ∩
Ens_Commande_Produit(pd)) ∧
Quantite_Commandee(cc) ≤ qt) ;

/* le produit pd est facturable pour une quantité en stock supposée être qt */

Cond_Facturable(Cd, pd, qt) ≙

Cd ⊆ Ens_Commande_En_Attente ∧
Cd ⊆ Ens_Commande_Produit(pd) ∧ Cd ≠ {} ∧
SIGMA(xx).(xx ∈ Cd | Quantite_Commandee(xx)) ≤ qt ;

/* Cd est un ensemble facturable */

Max_A_Facturer(Cd, pd, qt) ≙ *Cond_Facturable(Cd, pd, qt) ∧*
∀ Cd2. (Cd2 ⊆ Ens_Commande_En_Attente ∧
Cd ⊂ Cd2 => not(Cond_Facturable(Cd2,pd, qt)))

/* Cd est un ensemble maximal satisfaisant la condition facturable */

OPERATIONS

Creation_Cd(pd, qt) =
PRE Ens_Commande_Inexistante ≠ {} ∧
pd ∈ Produits ∧
qt ∈ NAT
THEN **ANY** cd **WHERE** cd ∈ Ens_Commande_Inexistante **THEN**
IF qt ≤ Quantite_Stock(pd) **THEN**
Ajoute_Commande_Facturee(cd, pd, qt) ||
Change_Quantite(pd, -qt)
ELSE
Ajoute_Commande_En_Attente(cd, pd, qt)
END
/* au moment de l'ajout de la commande si la quantité
commandée est <= à la quantité en stock, on facture
la commande, sinon la commande reste dans l'état
en attente */
END
END ;

Annul_Cd_En_Attente(cd) =
PRE cd ∈ Ens_Commande_En_Attente
THEN *Supprime_Commande(cd)*

```
                /* L'annulation d'une commande en état en attente n'entraîne
                   pas l'entrée d'une quantité pour le produit concerné */
END ;

Annul_Cd_Facturee(cd) =
PRE             cd ∈ Ens_Commande_Facturee ∧
                Quantite_Stock(Reference(cd)) + Quantite_Commandee(cd) ∈ NAT
THEN
        IF Facturable(Reference(cd), Quantite_Stock(Reference(cd)) +
                                        Quantite_Commandee(cd))
        THEN
                ANY Cd WHERE Max_A_Facturer(Cd,Reference(cd),
                        Quantite_Stock(Reference(cd) ) + Quantite_Commandee(cd))
                THEN
                        Supprime_Facture_Commande(cd, Cd) ||
                        Change_Quantite(Reference(cd),Quantite_Commandee(cd) -
                                SIGMA(xx).(xx ∈ Cd | Quantite_Commandee(xx)))
                END
        ELSE    Supprime_Commande(cd) ||
                Change_Quantite(Reference(cd), Quantite_Commandee(cd))
        END
END ;
Ajoute_Quantite(pd, qt) =
PRE             pd ∈ Produits ∧
                qt ∈ NAT ∧
                Quantite_Stock(pd) + qt ≤ MAXINT
THEN            /* L'ajout d'une quantité pour un produit entraîne éventuellement
                   la facturation d'un ensemble maximal de commandes en attente
                   pour le produit concerné */
        IF Facturable(pd, Quantite_Stock(pd) + qt) THEN
                ANY Cd WHERE Max_A_Facturer(Cd, pd, Quantite_Stock(pd) + qt)
                THEN    Facture_Com(Cd) ||
                        Change_Quantite(pd,qt - SIGMA(xx).
                                (xx ∈ Cd | Quantite_Commandee(xx)))
                END
```

```
            ELSE        Change_Quantité(pd, qt)
            END
        END
END
```

IV. Validation de la spécification

Comme nous avons présenté dans le chapitre 4, la méthode B définit les obligations de preuve pour chaque opération et chaque initialisation. Ces obligations de preuve permettent d'assurer la validité de la spécification. À titre d'exemple, étant donnée l'opération de création d'une commande comme suit (version simplifiée) :

Création_Cd =
PRE Commandes ≠ COMMANDES
THEN **ANY** cc **WHERE** cc ∈ COMMANDES - Commandes **THEN**
Commandes := Commandes ∪ {cc}
END
END

Nous obtenons l'obligation de preuve suivante :

(1) Commandes ⊆ COMMANDES ∧
(2) Commandes ≠ COMMANDES ∧
(3) cc ∈ COMMANDES - Commandes
=>
(4) Commandes ∪ {cc} ⊆ COMMANDES

Cette obligation de preuve exprime que sous l'invariant (1) et les conditions des clauses PRE (2) et ANY (3), la substitution doit préserver l'invariant (4). Autrement dit, après avoir ajouté cc dans l'ensemble Commandes, cet ensemble doit être toujours un sous-ensemble de COMMANDES.

Dans le cadre de ce travail, nous avons utilisé l'Atelier B pour valider la spécification B obtenue. Pour chaque machine spécifiée, le tableau suivant présente :

- le nombre d'obligations de preuve générées,
- le nombre de preuves prouvées automatiquement par l'Atelier B,
- le nombre de preuves prouvées en mode interactif,

- le nombre de preuves restant,
- le pourcentage de preuves démontrées.

Machine	Obligations de preuve	Preuves automatiques	Preuves Interactives	Preuves Restant	% Prouvé
Produit	5	5 (soit 100 %)	0 (0 %)	0 (0 %)	100 %
Commande	24	21 (soit 87 %)	3 (13 %)	0 (0 %)	100 %
Interface	38	28 (soit 74 %)	10 (26 %)	0 (0 %)	100 %
Total	67	54 (soit 81 %)	13 (19 %)	0 (0 %)	100 %

Nous avons remarqué que les preuves interactives peuvent souvent être regroupées en catégories à l'intérieur desquelles les démonstrations sont similaires.

Exemple :

(i) Quelques preuves de la machine Commande
Ces preuves concernent l'opération Supprime_Facture_Commande(cd, Cd)

- $\{cd\} \lhd Etat\_cd \lhd\!\!+ Cd * \{Facturee\} \in Commandes - \{cd\} \rightarrow\!\!\!\!\!\!\rightarrow ETAT$
- $dom(\{cd\} \lhd Etat\_cd \lhd\!\!+ Cd * \{Facturee\}) = Commandes - \{cd\}$

Ces preuves concernent le caractère total ou partiel d'une fonction. Par exemple, une condition suffisante pour prouver le deuxième point est de prouver que la fonction du premier point est en fait totale.

(ii) Quelques preuves de la machine Interface
Ces preuves concernent l'opération Annul_Commande_Facturée(cd)

- Quantite_Commandee(cd) - SIGMA(xx).(xx ∈ Cd | Quantite_Commandee(xx)) ∈ INTEGER
- Quantite_Stock(Reference(cd)) + (Quantite_Commandee(cd) - SIGMA(xx).(xx ∈ Cd | Quantite_Commandee(xx))) ⩽ MAXINT

Ces preuves se rapportent au manque de règles de base dans l'Atelier B concernant "SIGMA". Par exemple, il manque au prouveur de savoir qu'une somme d'entiers naturels est positive.

En conclusion, nous pouvons remarquer que l'application des règles proposées au chapitre 5 permet l'obtention d'une spécification correcte puisque 100% des preuves ont pu être démontrées. Soulignons en particulier que toutes les preuves que l'Atelier B n'a pas pu démontrer automatiquement ont été menées à bien en mode interactif.

Par ailleurs, la solution choisie dans [DIA 98]* ne consiste pas à partir d'une spécification semi-formelle mais de proposer directement une spécification formelle. Deux machines sont obtenues. L'une correspond à notre machine Produit et l'autre au regroupement de nos deux machines Commande et Interface. Il est à noter qu'en adoptant le choix de modularisation proposé dans [DIA 98], nous obtenons le même nombre d'obligation de preuves et de preuves démontrées automatiquement qu'avec notre choix de modularisation. Notre approche apporte cependant un gain de lisibilité sans pour autant générer de preuves supplémentaires.

* [DIA 98], H. Diab and M. Frappier, *"The specification of the Invoicing Case Study in B"* , International Workshop on : Comparing Systems Specification Techniques "What questions are prompted by ones particular method of specification ?" March 26-27, 1998, Nantes (France)

Annexe 2

GESTION D'UN PASSAGE À NIVEAU

Nous reprenons l'étude de cas proposée dans [DUP 98]. Elle a été spécifiée en Object-Z. Nous présentons à titre indicatif la spécification B obtenue en appliquant les règles proposées dans notre travail.

I. Présentation du cas

Nous présentons ici la traduction de l'énoncé de l'étude de cas.

Un passage à niveau est constitué de plusieurs composants qui avertissent les conducteurs et les empêchent de traverser la voie ferrée quand un train approche (voir figure 1). Ces composants sont : un signal sonore, un signal lumineux clignotant et une barrière. En fait, ces composants apparaissent de chaque côté de la voie mais on suppose que les composants de chaque côté ont le même comportement.

Quand un train rentre dans la zone de détection des trains venant d'une direction quelconque, les signaux lumineux et le signal sonnore sont déclenchés pendant que la barrière se baisse. Quand tous les trains sont sortis de la zone de détection des trains, les barrières s'ouvrent et quand elles ont atteint la position verticale, les signaux lumineux et le signal sonore cessent.

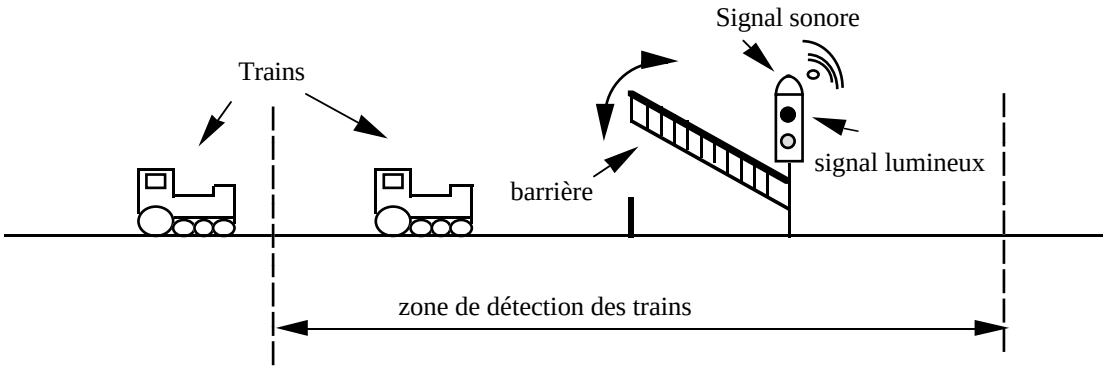


Figure 1. *Un passage à niveau (tiré de [DUP 98])*

Légende : cette figure a été proposée dans [DUP 98]. Nous la reprenons dans le but de faciliter la compréhension du sujet.

On suppose que lorsqu'un train entre dans la zone de détection des trains un événement TrainArrival (Arrivée de train) est déclenché et de façon similaire un événement TrainRemoval (Retrait de train) signale la sortie d'un train.

II. Spécification semi-formelle

Nous reprenons le diagramme d'objets et le diagramme états/transitions proposés dans [DUP 98] qui décrivent l'aspect statique et dynamique de l'application afin de pouvoir comparer les deux méthodes.

II.1. Le modèle d'objets

[DUP 98] propose de représenter dans le modèle OMT le passage à niveau décrit ci-dessus par une entité LevelCrossing (Passage à niveau) (figure 2). Cette entité possède les attributs : nbTrain qui représente le nombre de trains présents dans la zone de détection, light (signal lumineux), soundsignal (signal sonore) et lcgate (barrière). Quatre opérations de base :

PermitCrossing (Autoriser le passage), ForbidCrossing (Interdire le passage), AddTrain (Ajouter un train) et RemoveTrain (Supprimer un train) sont prédéfinies.

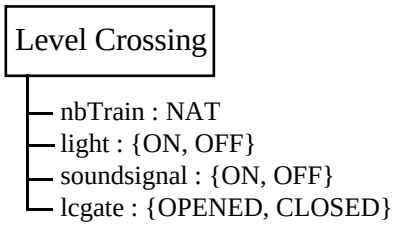


Figure 2. *Le diagramme d'objets du cas "Gestion d'un passage à niveau"*

II.2. Le modèle dynamique

Le diagramme E/T proposé dans [DUP 98] exprime le fait qu'un passage à niveau peut avoir deux états : Permitted (Autorisé) et Forbidden (Interdit) qui représentent le droit de traverser la voie. Si un train arrive dans la zone de détection l'état devient Forbidden, l'événement TrainArrival déclenche les actions ForbidCrossing (qui commute le signal lumineux, le signal sonore et la barrière) et AddTrain (qui incrémente le nombre de trains présents dans la zone de détection). Quand le dernier train quitte la zone de détection (événement TrainRemoval et condition nbTrain = 1), le passage à niveau devient Permitted. L'état "Permitted" signifie que les signaux lumineux et sonore sont OFF (éteints), la barrière est OPENED (ouvert) et le nombre de trains vaut 0. L'état "Forbidden" signifie que les signaux lumineux et sonore sont ON (allumés), la barrière CLOSED (fermée) et le nombre de trains strictement positif.

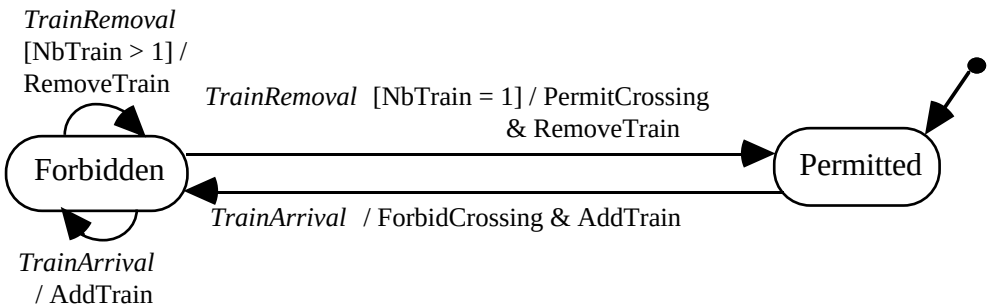


Figure 3. *Le diagramme E/T (tiré de [DUP 98])*

Nous présentons de plus le diagramme de suivi d'événements qui décrit les scénarios du système. Dans notre approche, ce diagramme sert pour la traduction des opérations.

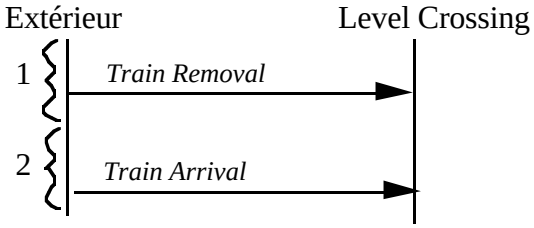


Figure 4. *Le diagramme de suivi d'événements*

III. Spécification formelle

Rappelons que nos règles ont été définies principalement pour des applications dans lesquelles on veut gérer un ensemble d'instances (type base de données). Dans ce contexte, à chaque entité sont associés un ensemble des instances possibles, une variable représentant l'ensemble des instances existantes. À chaque attribut de l'entité, est associée une variable qui est une relation entre l'ensemble des instances existantes et le type de l'attribut. Dans cette étude de cas, l'entité n'a qu'une seule instance car on gère un seul passage à niveau. Cela ne remet pas en cause les règles proposées. On pourrait facilement prendre en compte ce genre d'applications en transformant une variable de type relation dans laquelle l'ensemble de départ est réduit à un singleton en une variable dont le type est celui du domaine d'arrivée. C'est ce que nous choisissons de faire ici : par exemple, la variable nbTrain est de type NAT au lieu d'être une relation entre l'ensemble des passages à niveau et l'ensemble des entiers naturels.

Nous avons défini la machine Level_Crossing (associée à l'entité portant le même nom) dans laquelle nous spécifions les variables correspondant à ses attributs ainsi que les opérations élémentaires qui vont être appelées dans les opérations relatives aux scénarios. Rappelons qu'une opération élémentaire peut être une opération de base (AddTrain, RemoveTrain) ou une opération pré-composée (PermitCrossing_RemoveTrain, ForbidCrossing_AddTrain) de plusieurs opérations de base.

La machine Interface qui inclut la machine Level_Crossing se compose de la description des états et des opérations correspondant aux scénarios du système. Remarquons que dans cette étude de cas, nous aurions pu ne spécifier qu'une seule machine contenant les variables de l'entité Level_Crossing et les opérations associées aux scénarios dans lesquelles

nous modifions directement les variables. La séparation en deux machines permet une spécification incrémentale et facilite la compréhension.

Voici les machines obtenues :

MACHINE Level_Crossing
SETS SIGNAL = {ON, OFF} ;
GATE = {OPENED, CLOSED}
VARIABLES
nbTrain, light, soundsignal, lcgate
INVARIANT
nbTrain ∈ NAT ∧
light ∈ SIGNAL ∧
soundsignal ∈ SIGNAL ∧
lcgate ∈ GATE
INITIALISATION
light := OFF || lcgate := OPENDED || soundsignal := OFF || nbTrain := 0
OPERATIONS
ForbidCrossing_AddTrain = /* opération pré-composée */
PRE *nbTrain + 1 ∈ NAT*
THEN *light := OFF ||*
lcgate := OPENDED ||
soundsignal := OFF ||
nbTrain := nbTrain + 1
END ;
PermitCrossing_RemoveTrain = /* opération pré-composée */
PRE *nbTrain > 0*
THEN *light := OFF ||*
lcgate := OPENDED ||
soundsignal := OFF ||
nbTrain := nbTrain - 1
END ;
AddTrain = /* opération de base */
PRE *nbTRain + 1 ∈ NAT*
THEN *nbTrain := nbTrain + 1*
END ;

```
        RemoveTrain =                          /* opération de base */
        PRE         nbTrain > 0
        THEN        nbTrain := nbTrain - 1
        END
END
```

```
MACHINE         Interface
INCLUDES        Level_Crossing
DEFINITIONS     Permitted   ≙   light = OFF ∧
                                lcgate = OPENDED ∧
                                soundsignal = OFF ∧
                                nbTrain = 0
                Forbidden   ≙   light = ON ∧
                                lcgate = CLOSED ∧
                                soundsignal = ON ∧
                                nbTrain > 0
OPERATIONS
        TrainRemoval =
        PRE         Forbidden
        THEN        IF nbTrain = 1 THEN PermitCrossing_RemoveTrain
                    ELSE        RemoveTrain
                    END
        END ;
        TrainArrival =
        PRE         (Permitted ∨ Forbidden) ∧
                    nbTrain + 1 ∈ NAT
        THEN        IF Permitted THEN ForbidCrossing_AddTrain
                    ELSE        AddTrain
                    END
        END
END
```

IV. Validation de la spécification

Nous présentons ci-dessous le tableau qui indique pour chaque machine le nombre d'obligations de preuve générées, le nombre de preuves démontrées automatiquement et le pourcentage de preuves démontrées.

Machine	Obligations de preuve	Preuves automatiques	% Prouvé
Level_Crossing	6	6 (soit 100%)	100 %
Interface	Aucune OP n'a été retenue par le générateur d'OP (preuves triviales)		
Total	6	6 (soit 100%)	100 %

La spécification obtenue a été prouvée totalement. Il est à noter que les conditions de bord $nbTrain > 0$ et $nbTrain + 1 \in Nat$ n'apparaîssaient pas dans la spécification obtenue initialement en appliquant les règles de dérivation puisqu'elles n'ont pas été décrites dans la spécification semi-formelle. Bien évidemment, dans cette première spécification, les obligations de preuve qui consistent à démontrer que l'incrémentation et la décrémentation du nombre de trains respectent les extrémités du type NAT n'ont pas pu être démontrées. Nous avons donc ajouté les pré-conditions concernant ces vérifications dans chaque opération concernée. Dans la spécification en Object Z proposée par [DUP 98] on ne trouve pas ces conditions. Cela est peut-être dû à l'absence d'un outil de preuves.

Avec notre approche, on obtient une spécification avec moins de variables et moins d'opérations. En effet, nous n'avons pas ajouté de variables pour représenter les états du digramme E/T, de ce fait nous n'avons pas besoin d'associer à chaque transition une opération qui change l'état. Cependant, en Object-Z on peut appeler plusieurs opérations d'un même schéma dans une autre opération, cet avantage permet de ne pas pré-composer les opérations en une opération unique comme en B. Par ailleurs, contrairement à [DUP 98], nous ne nous intéressons pas à la gestion des événements. En effet, notre travail se situant au premier niveau de la spécification, nous préférons spécifier l'effet des événements sur les données du système dans le but d'en assurer la cohérence plutôt que de gérer l'arrivée des événements eux-mêmes.

Annexe 3

GESTION D'UN CLUB VIDÉO

I. Présentation du cas

Dix boutiques de location de cassettes vidéo se sont regroupées pour mettre en commun les cassettes dont elles disposent et ont fondé un club de location de cassettes. Chaque boutique dispose d'un terminal écran-clavier relié à un mini-ordinateur central.

Le club contient plusieurs films. Chaque film et chaque épisode (si un film en a plusieurs) peuvent avoir plusieurs cassettes qui les concerne. Chaque film a un numéro, un titre (de même pour chaque épisode) et un genre.

Chaque cassette possède un numéro qui lui est propre et concerne un film ou un épisode. Le club ne possède pas de cassettes vierges. Une cassette peut être dans un des trois états suivants :

- *Disponible* : si la cassette est présente dans une des boutiques,
- *Empruntée* : si la cassette est empruntée par un client. Les cassettes empruntées doivent être retournées (dans n'importe quelle boutique du club) dans un délai de 8 jours,
- *Réservée* : si elle est réservée par un client. La réservation des cassettes sera supprimée automatiquement après un délai d'un jour si le client ne vient pas les chercher.

I.1. La gestion des inscriptions

Un client, désirant emprunter des cassettes au club, doit s'inscrire et verser une caution. Le club lui associera une carte. Sur chaque carte est noté le numéro qui lui est propre, le nom, l'adresse et le nombre maximum de cassettes empruntées et réservées selon la caution payée. La caution sera rendue au client en cas de résiliation de l'abonnement, elle sera, par contre, encaissée si ce dernier garde plus de trois mois une cassette et le client sera alors classé dans la catégorie des clients douteux.

I.2. La gestion des emprunts de film (ou d'épisode)

Pour emprunter des films (ou épisodes), le client doit se présenter avec sa carte au guichet d'une boutique. Le guichetier vérifie si c'est un client douteux. Si c'est le cas, l'emprunt est refusé.

Un client peut emprunter des films s'il est en règle (c'est-à-dire qu'il n'a pas de cassette en retard) et si le nombre maximum de cassettes empruntées et réservées n'est pas encore atteint. S'il existe une cassette disponible du film (ou épisode) qu'il désire emprunter dans la boutique, on lui prête la cassette. Dans le cas où il existe une cassette disponible du film (ou épisode) dans une autre boutique, le client peut la réserver pour un délai d'un jour.

I.3. La gestion des remises de cassettes

Les cassettes empruntées doivent être retournées dans un délai de huit jours dans n'importe quelle boutique. Quand un client rend une cassette, si le délai d'emprunt est dépassé, le client doit payer une amende. On ne gère pas les amendes.

I.4. La gestion des résiliations

Un client peut voir sa carte retirée s'il garde au moins une cassette pendant plus de trois mois. Dans ce cas, toutes les cassettes empruntées par ce client doivent être supprimées. Une personne peut aussi ne plus être client en demandant de retirer son nom de la liste clientèle. Cela est possible seulement si le client a rendu toutes ses cassettes.

II. Spécification semi-formelle

II.1. Le modèle d'objets

À partir du texte précédent, nous allons utiliser les concepts du modèle d'objets pour obtenir une réécriture un peu plus formalisée : la spécification semi-formelle pour les données du système.

II.1.1. Les entités

II.1.1.1. Client

On veut garder les informations suivantes concernant le client :

- Numéro de carte (Num_Carte: NAT),
- Nom, Adresse : pour pouvoir communiquer avec le client en cas de besoin (par exemple l'envoi de lettre pour les cassettes en retard) (Nom_Cl: STRING, Adresse_Cl: STRING),
- Nombre de cassettes maximal que le client peut emprunter et réserver en même temps : cette information est utile pour la gestion des emprunts et des réservations (Nb_Max: NAT),
- Carte retirée : afin d'enregistrer les clients résiliés par le club (Carte_Retirée: BOOLEAN).

II.1.1.2. Boutique

Le club possède dix boutiques, il est donc nécessaire de conserver les informations concernant chacune d'entre elles afin de les communiquer au client en cas de besoin. Ces informations sont : le nom et l'adresse de la boutique (Nom_B: STRING, Adresse_B: STRING).

II.1.1.3. Film, Épisode

Pour chaque film, on veut stocker le numéro, le titre du film (de même pour épisode) et son genre (Num_Fl: NAT, Titre_Fl: STRING, Genre: STRING, Num_Ep: NAT, Titre_Ep: STRING).

II.1.1.4. Cassette

Les cassettes se composent chacune d'un numéro déterminant son identité (Num_K7: NAT).

II.1. 2. Les associations

II.1.2.1. Emprunt

Un client est lié à une cassette si la cassette lui est prêtée. Nous introduisons cette relation entre Client et Cassette par l'association Emprunt.

La cardinalité (0,p) entre Client et Emprunt exprime le fait que le client peut emprunter de 0 à p cassettes. Rappelons que $0 \leq p \leq$ Nb_Max pour chaque client car le client ne peut emprunter plus d'un certain nombre de cassettes en fonction du montant de caution qu'il a versé au début.

La cardinalité (0,1) entre Cassette et Emprunt exprime le fait qu'une cassette ne peut pas être empruntée par plusieurs clients à la fois.

L'attribut Date_E (de type NAT) enregistre la date d'emprunt de la cassette afin de pouvoir gérer les cassettes en retard.

II.1.2.2. Réservation

Quand la cassette n'est pas disponible dans la boutique mais dans une autre, le client peut la réserver. Ce lien est représenté par l'association Réservation entre Client et Cassette.

Notons qu'une cassette ne peut pas être réservée par plus d'un client à la fois, nous avons donc la cardinalité (0,1) entre Cassette et Réservation. Cependant, un client peut réserver au maximum Nb_Max cassettes, nous avons donc la contrainte de cardinalité (0,q) entre Client et Réservation ($0 \leq q \leq$ Nb_Max).

L'attribut Date_Res (de type NAT) est ajouté afin de pouvoir supprimer les réservations de la veille.

II.1.2.3. Appartient

Cette association décrit les liens entre Cassette et Boutique. Puisqu'une cassette ne se trouve que dans une seule boutique, la contrainte de cardinalité entre Cassette et Appartient est égale à (1,1).

II.1.2.4. Contient1

Contient1 se compose des liens entre Cassette et Film. Puisqu'une cassette ne concerne qu'un seul film, nous définissons la contrainte de cardinalité (0,1) entre Cassette et Contient1.

II.1.2.5. Contient2

Cette association se compose des liens entre Cassette et Épisode. Notons qu'une cassette ne peut être liée qu'à un seul épisode, ce qui entraîne la contrainte de cardinalité (0,1) entre Cassette et Contient2.

II.1.2.6. Possède

Cette association est réservée aux films qui ont plusieurs épisodes. La contrainte de cardinalité entre Film et Possède est donc égale à (0,n). Cependant, un épisode est forcement lié à un et un seul film. Nous avons la contrainte de cardinalité (1,1) entre Épisode et Possède.

II.1.3. Les contraintes d'intégrité

II.1.3.1. Les contraintes exprimables graphiquement

a. Notons qu'une cassette non-disponible est soit empruntée, soit réservée par un client. Nous devons donc définir une contrainte d'exclusion entre Emprunt et Réservation.

b. Afin d'exprimer le fait que le contenu d'une cassette peut être soit un film, soit un épisode mais pas les deux à la fois, nous définissons la contrainte d'exclusion entre les deux associations Contient1 et Contient2. "Le club ne possède pas de cassettes vierges" est décrit par la contrainte de totalité entre ces deux associations.

c. Si un film est contenu entièrement dans une cassette, il n'a pas d'épisode. Il faut définir alors la contrainte d'exclusion entre Contient1 et Possède.

II.1.3.2. Les contraintes non-exprimables graphiquement

Nous avons la contrainte suivante : le nombre de cassettes empruntées et réservées par un client ne doit pas dépasser le nombre maximum autorisé, c'est-à-dire : $p+q \leq Nb\_Max$. Il est défini à l'inscription du client selon la caution payée.

Voici le diagramme d'objets obtenu :

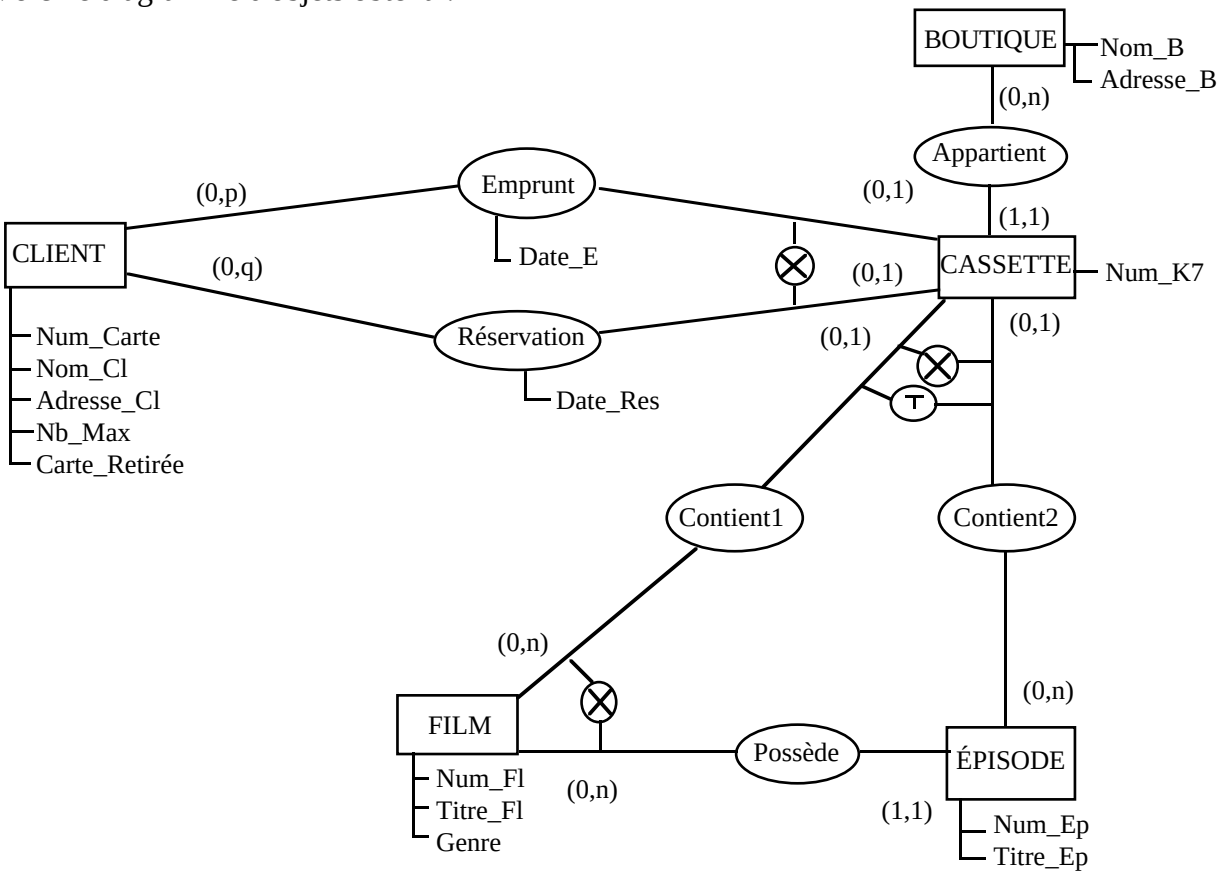


Figure 1. *Le diagramme d'objets du cas "Gestion d'un vidéo club"*

II.2. Le modèle dynamique

Le modèle dynamique est représenté graphiquement par les scénarios et les diagrammes E/T. Il nous faut tout d'abord déterminer les événements externes déclencheurs, l'enchaînement des événement internes, puis les entités des objets émetteurs et récepteurs de ces événements pour décrire le diagramme de suivi d'événements. Et enfin, nous construisons les diagrammes E/T pour ces entités. Dans cette étude de cas, les différentes entités sont : Client, Cassette. Nous commençons par le diagramme de suivi d'événements qui se compose de tous les scénarios du système.

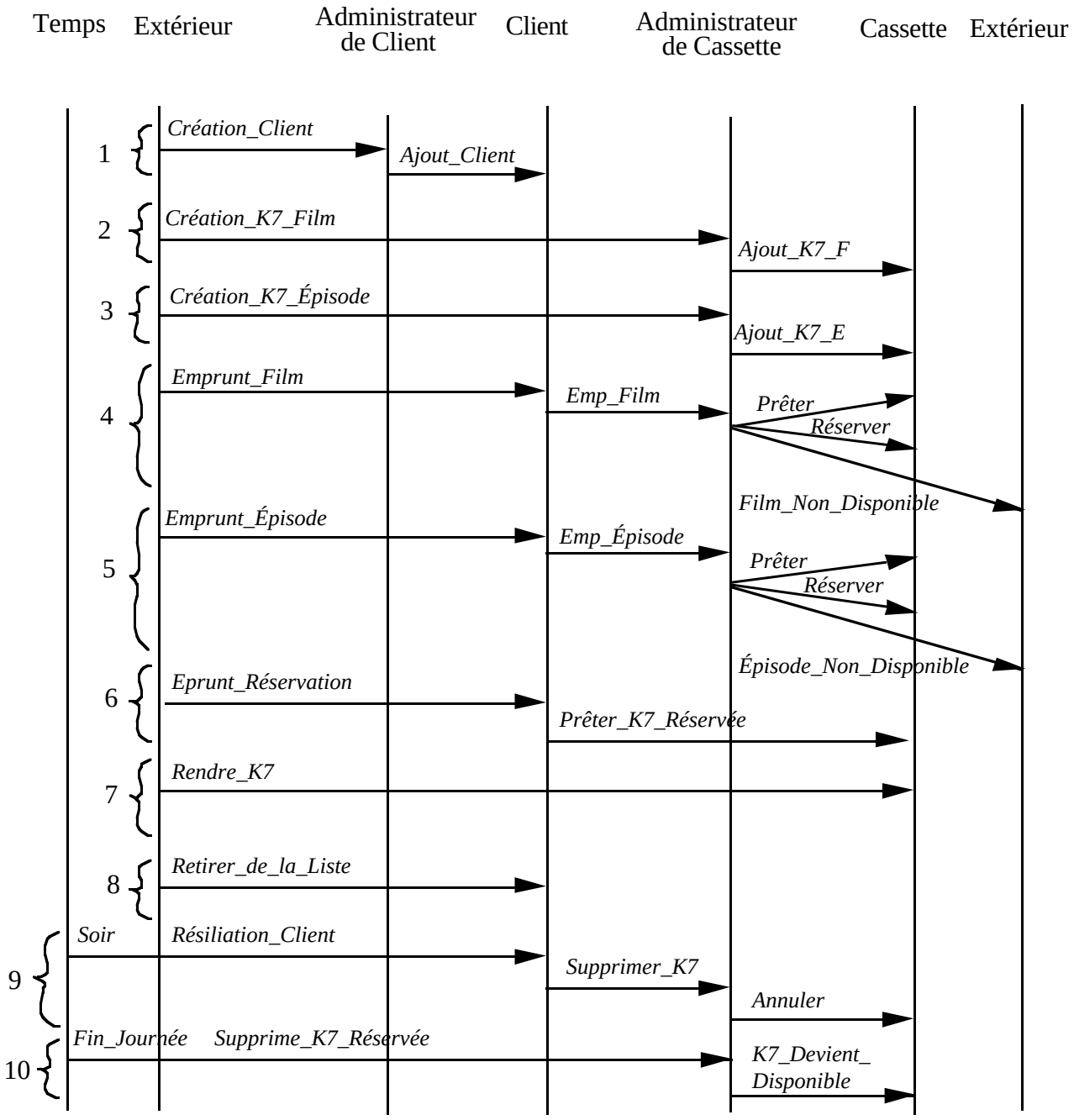


Figure 2. *Le diagramme de suivi d'événements*

Nous présentons ci-dessus les diagrammes E/T pour les entités Client, Cassette.

Diagramme Administrateur de Client :

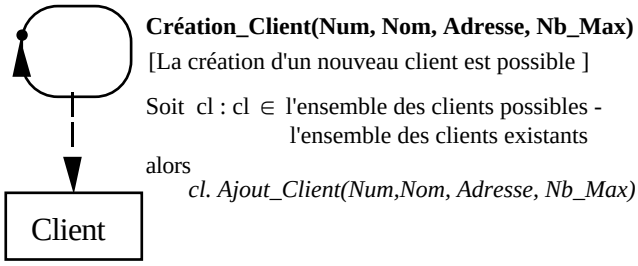


Figure 3. *Le diagramme administrateur de Client*

Diagramme E/T de Client :

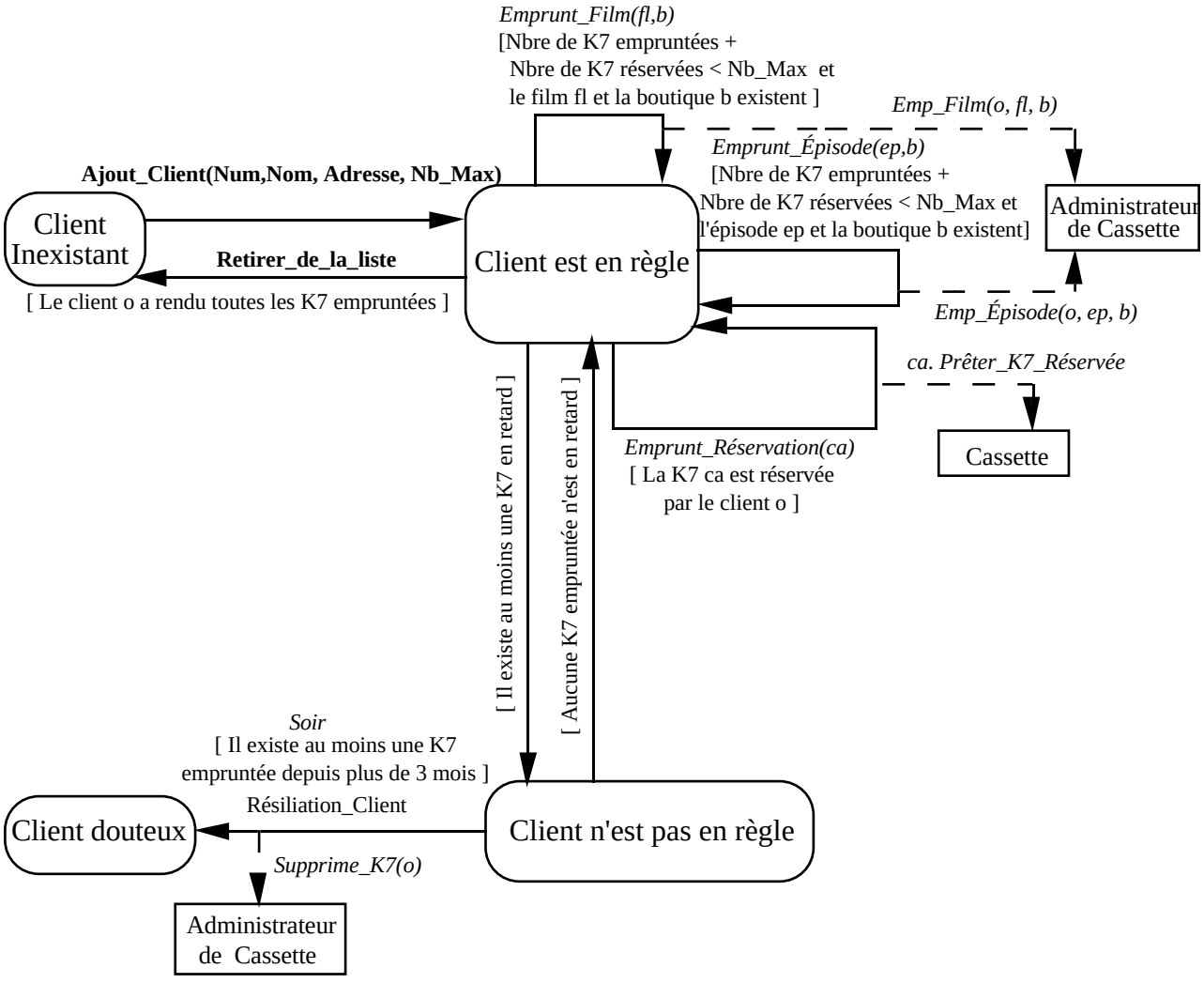


Figure 4. *Le diagramme E/T de Client*

Diagramme Administrateur de Cassette :

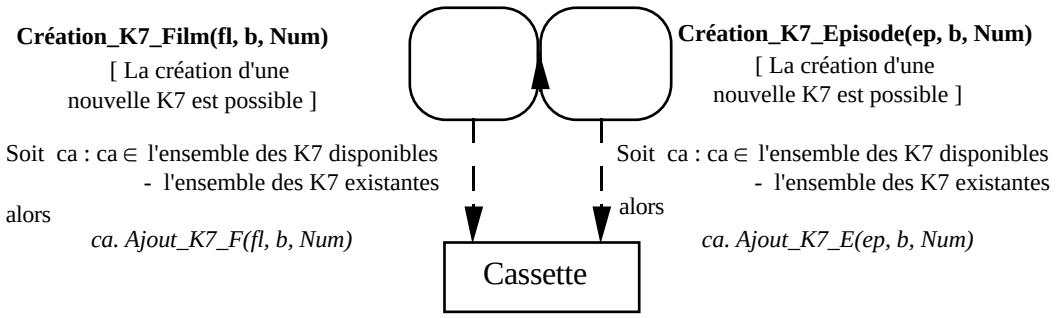


Figure 5. *Diagramme administrateur pour la création d'une nouvelle casette*

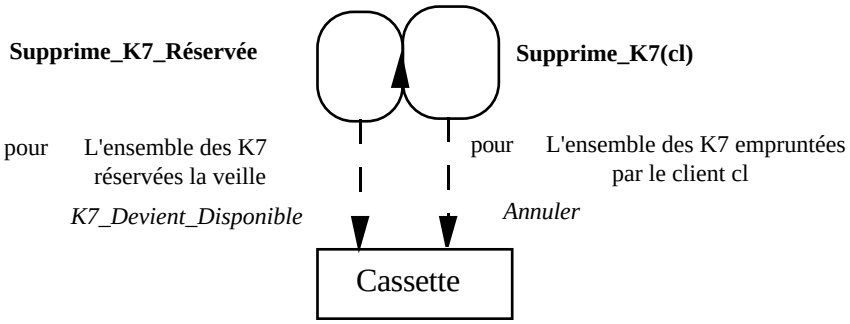


Figure 6. *Diagramme administrateur pour la suppression de toutes les cassettes empruntées par un client et la suppression d'une réservation*

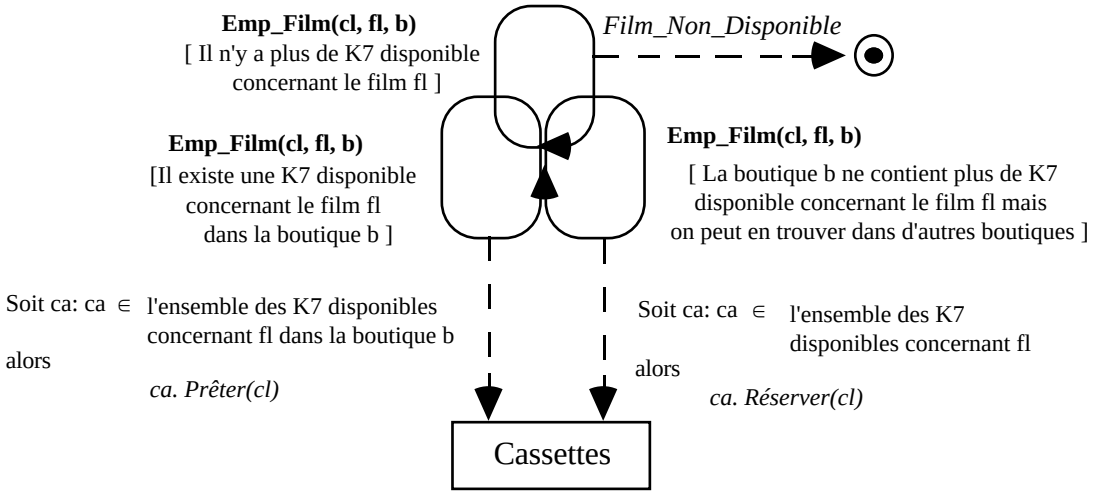


Figure 7. *Diagramme administrateur pour un emprunt de film*

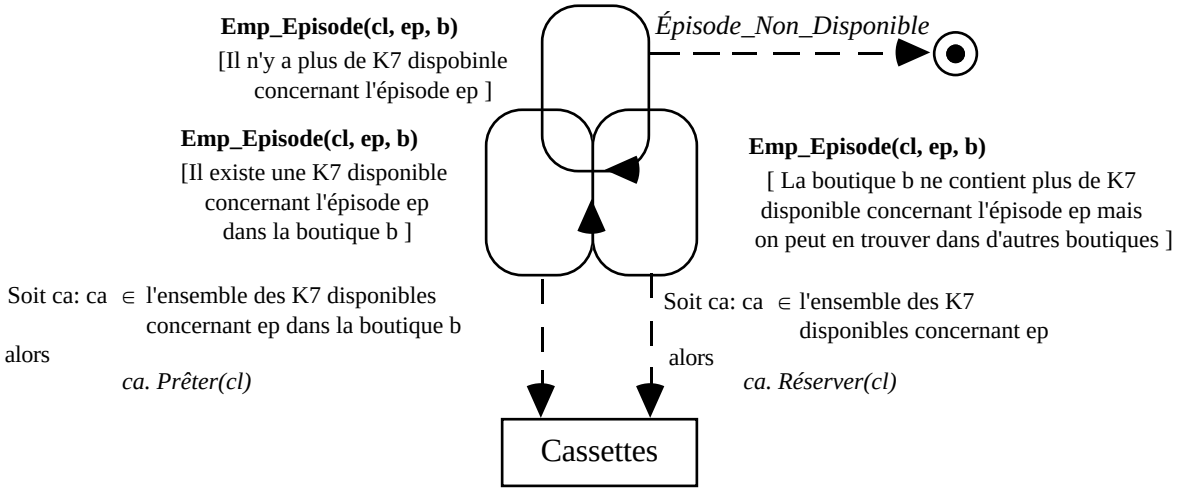


Figure 8. *Diagramme administrateur pour un emprunt d'épisode*

Diagramme E/T de Cassette :

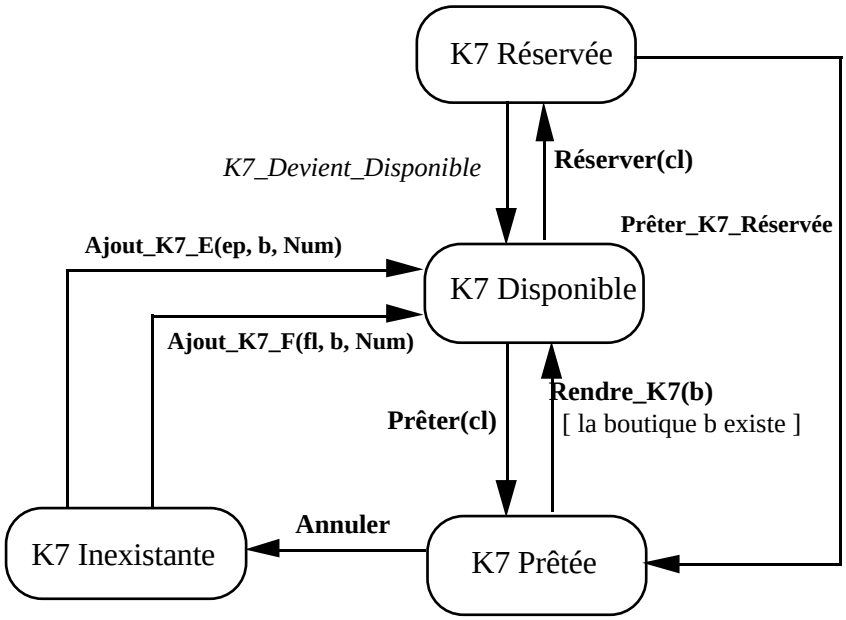


Figure 9. *Diagramme E/T de cassette*

Ci-dessous, nous allons donner les explications sur les scénarios et les diagrammes E/T.

II.2.1. Scénario pour la création d'un client

Création_Client est un événement externe vers l'Administrateur de Client (figure 3) qui doit d'abord choisir un nouveau client et ensuite l'ajouter dans l'ensemble des clients. Nous

avons l'événement interne Ajout_Client du diagramme de l'Administrateur de Client vers le diagramme de Client (figure 4).

II.2.2. Scénario pour la création d'une cassette concernant un film (ou un épisode)

Création_K7_Film est un événement externe vers l'Administrateur de Cassette (figure 5) qui doit d'abord choisir une nouvelle cassette et ensuite l'ajouter dans l'ensemble des cassettes. Nous avons l'événement interne Ajout_K7_F du diagramme de l'Administrateur de Cassette vers le diagramme de Cassette (figure 9).

II.2.3. Scénario pour l'emprunt d'un film (ou un épisode)

La figure 4 indique qu'Emprunt_Film est un événement externe vers le diagramme de Client. Il s'agit d'un choix de spécification. Un autre choix aurait pu consister à décrire Emprunt_Film comme un événement externe vers le diagramme de Film. Le client désirant emprunter un film est un paramètre implicite de cet événement. Notons que l'utilisateur doit vérifier les conditions de déclenchement (le nombre de cassettes empruntées et réservées par le client est inférieur à Nb_Max) avant d'émettre l'événement. Cet événement n'entraîne pas de changement d'état du client (transition réflexive). Par contre, il provoque un événement interne Emp_Film vers le diagramme de l'Administrateur de Cassette (figure 7). S'il existe une cassette disponible concernant le film dans la boutique, l'Administrateur de Cassette va en choisir une pour la prêter. Nous avons donc, l'événement interne Prêter vers le diagramme de Cassette (figure 9). En revanche, s'il existe une cassette disponible dans une autre boutique, l'Administrateur de Cassette va en choisir une pour la réserver, nous avons l'événement interne Réserver vers le diagramme de Cassette (figure 9). Si aucune cassette n'est disponible, l'événement Film_Non_Disponible est envoyé vers l'Extérieur (figure 7).

II.2.4. Scénario pour l'emprunt d'une cassette réservée

Emprunt_Réservation est un événement externe vers le diagramme de Client (figure 4). L'utilisateur doit vérifier si la cassette a bien été réservée par le client avant d'émettre cet événement. L'emprunt d'une cassette réservée ne change pas l'état du client. En revanche, la cassette réservée doit passer maintenant à l'état K7 Prêtée, nous avons donc l'événement interne Prêter_K7_Réservée du diagramme de Client vers le diagramme de Cassette (figure 9).

II.2.5. Scénario pour le retour d'une cassette

Rendre_K7 est un événement externe vers le diagramme de Cassette (figure 9). Cet événement changera l'état de la cassette en K7 Disponible.

II.2.6. Scénario pour la résiliation sur demande du client

Retirer_de_la_Liste est un événement externe vers le diagramme de Client (figure 4). Le client qui veut retirer son nom de la liste est un paramètre implicite de cet événement. Notons qu'une résiliation n'est possible que si le client a rendu toutes les cassettes qu'il a empruntées.

II.2.7. Scénario pour la résiliation automatique

Tous les soirs, chaque client n'étant pas en règle et gardant au moins une cassette plus de 90 jours est retiré de la liste (figure 4). Ceci entraîne la suppression de toutes les cassettes empruntées par ce client. Cette suppression est représentée par un événement interne Supprime_K7 du diagramme de Client vers l'Administrateur de Cassette (figure 6). L'Administrateur de Cassette doit d'abord sélectionner l'ensemble des cassettes empruntées par le client puis supprime chaque cassette de cet ensemble. Nous avons donc, l'événement interne Annuler de l'Administrateur de Cassette vers le diagramme de Cassette (figure 9).

II.2.8. Scénario pour la suppression des cassettes réservées

À la fin de la journée, il faut supprimer tous les cassettes réservées la veille. L'événement externe Supprime_K7_Réservée est envoyé à l'Administrateur de Cassettes afin de sélectionner toutes les cassettes en question (figure 6).

III. Spécification formelle

Dans cette partie, nous allons appliquer les règles de traduction pour obtenir la spécification B à partir des diagrammes OMT. Les machines B sont obtenues en quatre étapes :

- Traduction du modèle d'objets en spécification B et la génération des opérations de base,
- Utilisation des notations B dans les diagrammes E/T en s'aidant les résultats du premier étape,
- Génération de squelettes d'opération B à partir des diagrammes E/T et scénarios,

- Achèvement des machines obtenues par le corps d'opérations et les invariants correspondant aux contraintes d'intégrité non exprimables graphiquement.

III.1. Traduction du modèle d'objets en spécification B et génération des opérations de base

III.1.1. Traduction du modèle d'objets : les machines obtenues

III.1.1.1. Boutique

La machine MA_Boutique se compose de l'ensemble des boutiques possibles (BOUTIQUES), des variables qui représentent les boutiques existantes (Boutiques), les attributs Nom_B et Adresse_B, et des invariants décrivant les contraintes entre les variables et les ensembles, les types prédéfinis.

III.1.1.2. Client

La machine MA_Client est associée à l'entité Client. Elle contient l'ensemble des clients possibles (CLIENTS), la variable Clients décrit les clients existants du club, les variables Num_Carte, Nom_Cl, Adresse_Cl, Nb_Max, Carte_Retirée correspondant à ses attributs, et les invariants.

III.1.1.3. Film

La machine MA_Film contient l'ensemble des films possibles (FILMS), la variable Films correspondant à l'ensemble des films existants du club, ainsi que les variables qui représentent les attributs : Num_Fl, Titre_Fl et Genre. Les invariants décrivent les contraintes des variables.

III.1.1.4. Épisode

La machine MA_Épisode contient l'ensemble des épisodes possibles (ÉPISODES), les variables qui correpondent à l'ensemble des épisodes existants du club (Épisodes), aux attributs Num_Ep, Titre_Ep, et à l'association Possède. Cette association est définie dans la machine Épisode car elle est fixe pour cette entité. Notons qu'à chaque création d'épisode, nous devons l'associer à un film et ceci ne peut être modifié ensuite.

III.1.1.5. Cassette

La machine MA_Cassette est associée à l'entité Cassette. Elle contient l'ensemble des cassettes possibles (CASSETTES), la variable Cassettes décrivant les cassettes existantes du club, la variable Num_K7 correspondant à l' attribut de même nom et les attributs Contient1, Contient2 représentant les associations. Elles sont définies dans cette machine car elles sont fixes pour l'entité Cassette. Nous définissons également dans cette machine les contraintes de totalité et d'exclusion entre les deux associations Contient1, Contient2.

III.1.1.6. Appartient

La machine MA_Appartient a été créée pour définir l'association Appartient car elle n'est fixe pour aucune entité. Elle contient la variable Appartient qui décrit les liens entre Boutique et Cassette. Le lien *Uses* a été défini vers les machines Cassette et Boutique afin de réutiliser les variables de ces deux machines.

III.1.1.7. Emprunt

La machine MA_Emprunt est associée à l'association Emprunt. Elle contient l'attribut Emprunt représentant les liens Emprunt entre Client et Cassette, la variable Date_E correspondant à l'attribut Date_E de l'association. Nous avons créé cette machine avec le lien *Uses* vers la machine Client et Cassette car l'association n'est fixe pour aucune entité et elle contient un attribut.

III.1.1.8. Réservation

Avec les mêmes explications que pour le cas de l'association Emprunt, nous définissons la machine MA_Réservation dans laquelle se trouve l'attribut Réservation représentant les liens Réservation entre Client et Cassette, ainsi que la variable Date_Res correspondant à son attribut.

III.1.1.9. Interface

La machine MA_Interface inclut toutes les machines du système. Nous décrivons également dans cette machine, les contraintes d'intégrité entre les variables de plusieurs machines. Ces contraintes sont exprimables graphiquement sur le diagramme d'objets.

$$C1 : ran(Emprunt) \cap ran(Réservation) = Ø$$

C2 : ran(Contient1) ∩ dom(Possède) = Ø

Voici la structure des machines obtenues :

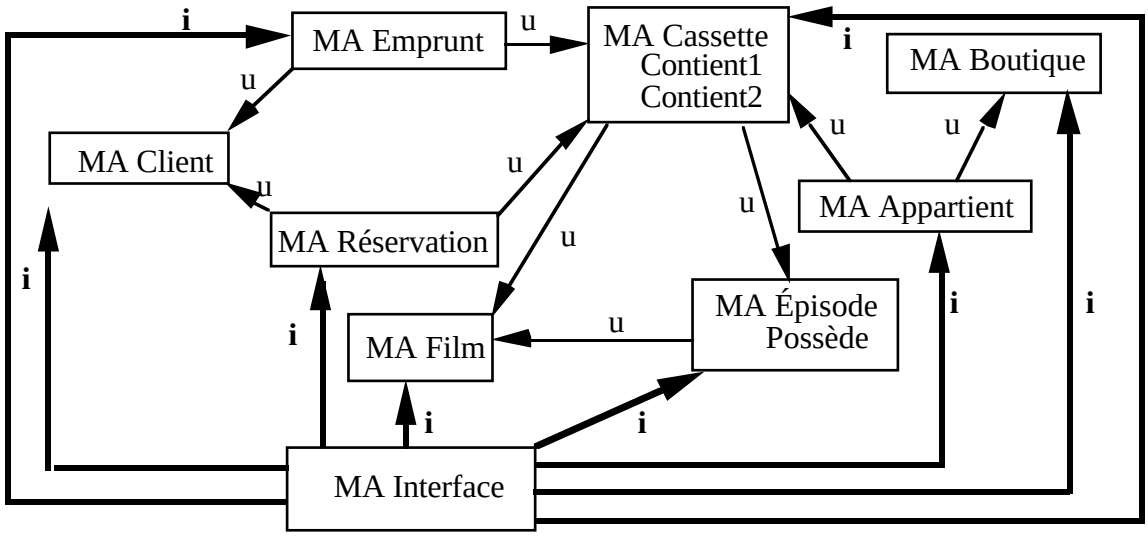


Figure 10. *La structure des machines B*

Légende : "i" signifie le lien *Includes* entre machines et
"u" signifie le lien *Uses* entre machines

III.1.2. Génération des opérations de base pour chaque machine

III.1.2.1. Liste des opérations de base de la machine MA_Boutique

Nous définissons les opérations de base suivantes :

- Ajout_Boutique(bb, nom_b, adresse_b) : ajoute une nouvelle boutique,
- Sup_Boutique(bb) : permet de supprimer une boutique existante,
- Chg_Nom_B(bb) : change le nom d'une boutique,
- Chg_Adresse_B(bb) : change l'adresse d'une boutique.

III.1.2.2. Liste des opérations de base de la machine MA_Client

Nous générons les opérations de base suivantes :

- Ajout_Client(cl, num_carte, nom_cl, adresse_cl, nb_max) : ajoute un nouveau client,
- Sup_Client(Cl) : supprime un ensemble de clients existants,
- Chg_Nom_Cl(cl) : change le nom d'un client,

- Chg_Adresse_Cl(cl) : change l'adresse d'un client,
- Chg_Carte(cl) : change la valeur de la carte en "Carte_Retirée".

Nous n'avons pas défini les opérations qui changent les valeurs des attributs Num_Carte, Nb_Max car leur valeur a été donnée à la création d'un client et ne change pas.

III.1.2.3. Liste des opérations de base de la machine MA_Film

Nous avons les opérations de base suivantes :

- Ajout_Film(fl, num_fl, titre_fl, genre_fl) : ajoute un nouveau film,
- Sup_Film(Fl) : supprime un ensemble de films existants.

Nous n'avons pas spécifié les opérations de changement de valeur des attributs Num_Fl, Titre_Fl et Genre car ces caractéristiques sont liées à un film à sa création et ne changent pas de valeur.

III.1.2.4. Liste des opérations de base de la machine MA_Épisode

Nous définissons les opérations de base suivantes :

- Ajout_Épisode(ep, num_ep, titre_ep, fl) : ajoute un nouvel épisode,
- Sup_Épisode(Ep) : supprime un ensemble d'épisodes existant.

Nous n'avons pas généré les opérations de changement de valeur des attributs Num_Ep et Titre_Ep car ces caractéristiques ne changent pas de valeur.

III.1.2.5. Liste des opérations de base de la machine MA_Cassette

Trois opérations de base sont définies :

- Ajout_Cassette_F(ca, num_ca, fl) : ajoute une nouvelle cassette concernant un film,
- Ajout_Cassette_E(ca, num_ca, ep) : ajoute une nouvelle cassette concernant un épisode,
- Sup_Cassette(Ca) : supprime un ensemble de cassettes existantes.

III.1.2.6. Liste des opérations de base de la machine MA_Appartient

Nous avons quatre opérations de base :

- Ajout_Appartient(ca, bb) : sert à effectuer un lien Appartient vers une boutique bb à la création de la cassette ca,
- Change_Boutique(ca, bb) : change la boutique d'une cassette,
- Sup_Appartient_Ca(Ca) : supprime les liens relatifs à l'ensemble des cassettes Ca,
- Sup_Appartient_B(Bb) : supprime tous les liens liés à l'ensemble des boutiques Bb.

III.1.2.7. Liste des opérations de base de la machine MA_Emprunt

Nous spécifions trois opérations de base :

- Ajout_Emprunt(cl, ca, date) : ajoute un nouveau lien,
- Sup_Emprunt_Cl(Cl) : supprime tous les liens relatifs à l'ensemble des clients Cl,
- Sup_Emprunt_Ca(Ca) : supprime les liens relatifs à l'ensemble des cassettes Ca.

III.1.2.8. Liste des opérations de base de la machine MA_Réservation

Nous spécifions trois opérations de base :

- Ajout_Réservation(cl, ca, date) : ajoute un nouveau lien
- Sup_Réservation_Cl(Cl) : supprime tous les liens relatifs à l'ensemble des clients Cl,
- Sup_Réservation_Ca(Ca) : supprime les liens relatifs à l'ensemble des cassettes Ca.

III.2. Utilisation des notation B dans les diagrammes E/T

Dans cette partie, nous précisons les diagrammes E/T présentés dans la partie II.2 en remplaçant les notations en langue naturelle par des notations B.

Diagramme Administrateur de Client :

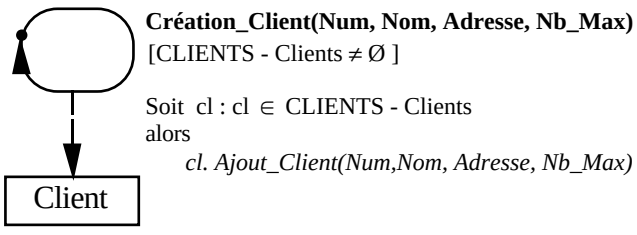


Figure 11. *Le diagramme administrateur de Client*

Diagramme E/T de Client :

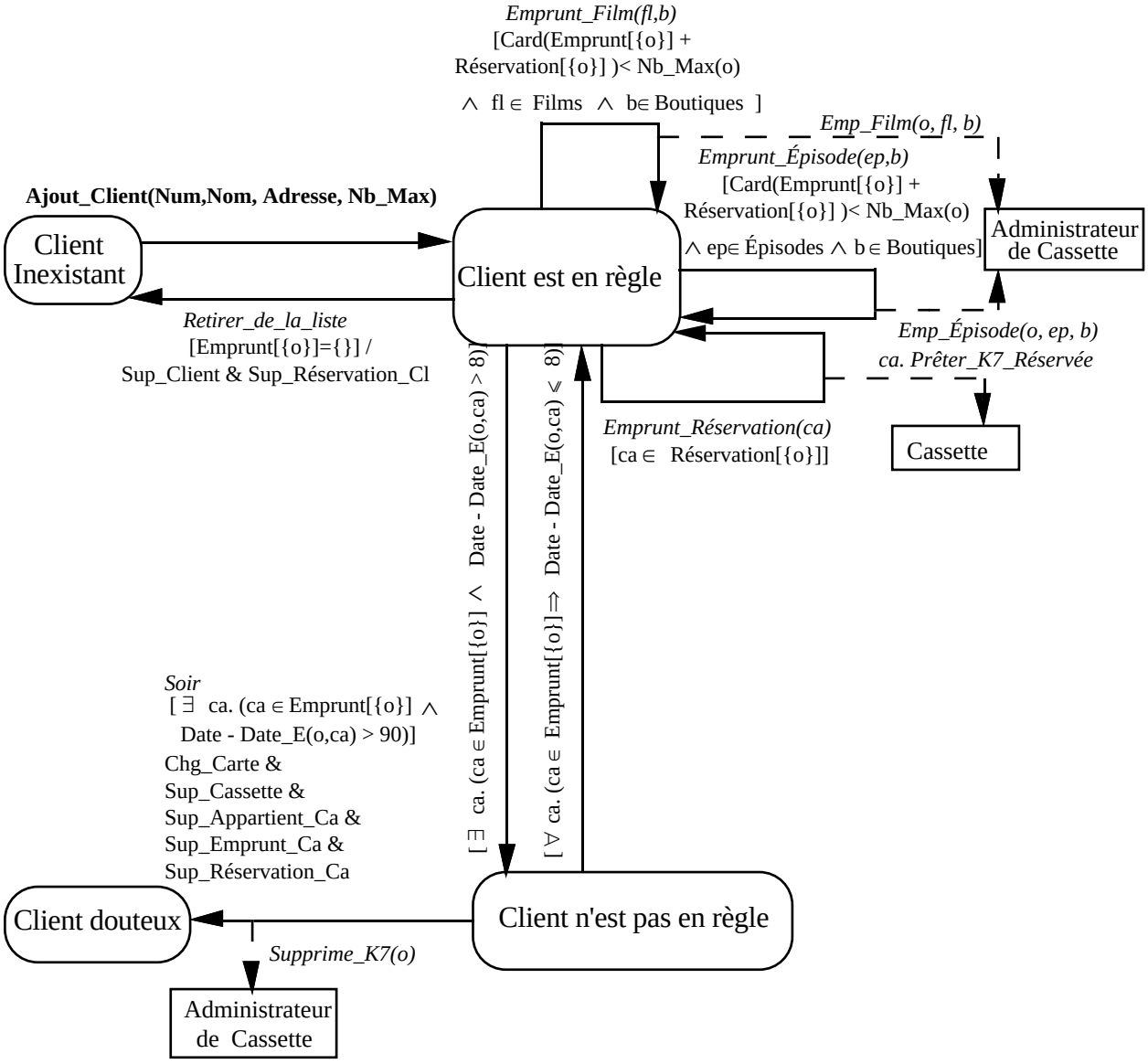


Figure 12. *Le diagramme E/T de Client*

Diagramme Administrateur de Cassette :

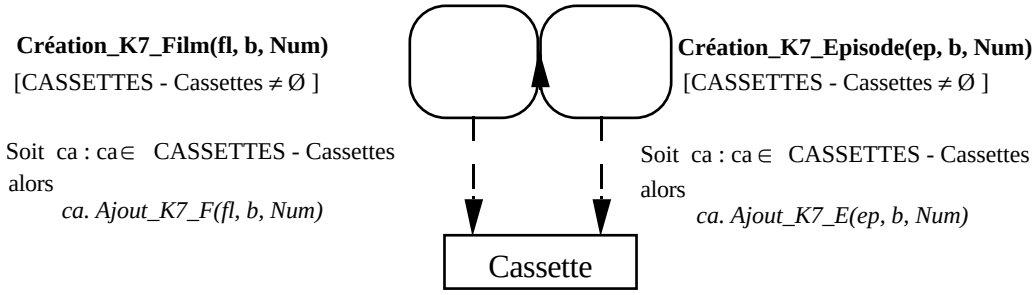


Figure 13. *Diagramme administrateur pour la création d'une nouvelle casette*

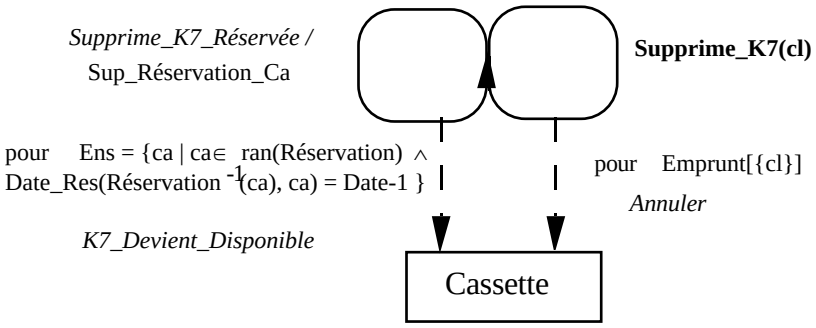


Figure 14. *Diagramme administrateur pour la suppression de toutes les cassettes empruntées par un client et la suppression d'une réservation*

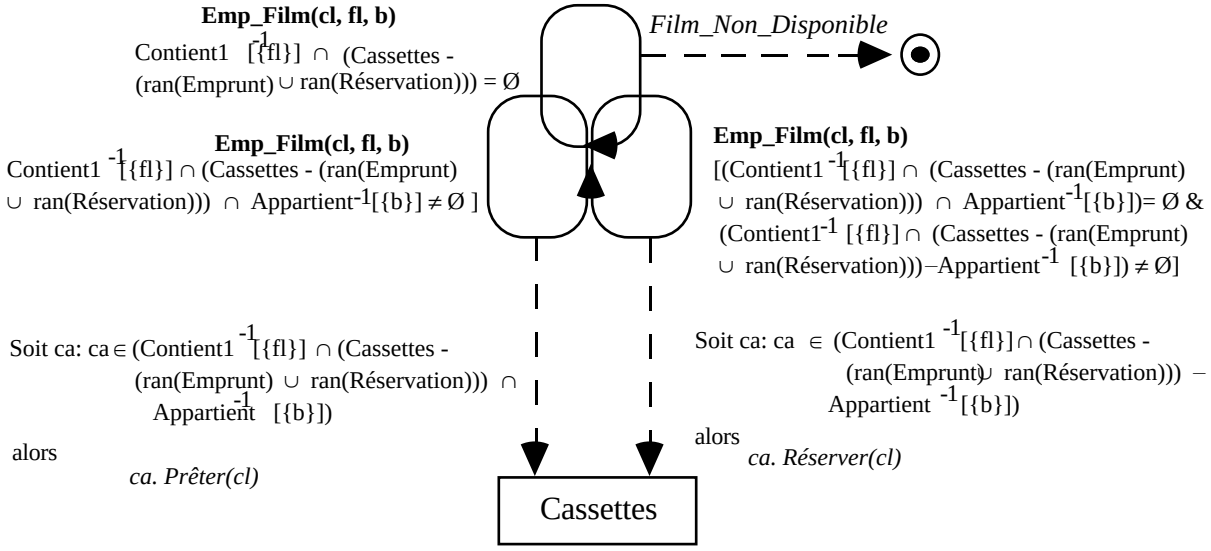


Figure 15. *Diagramme administrateur pour un emprunt de film*

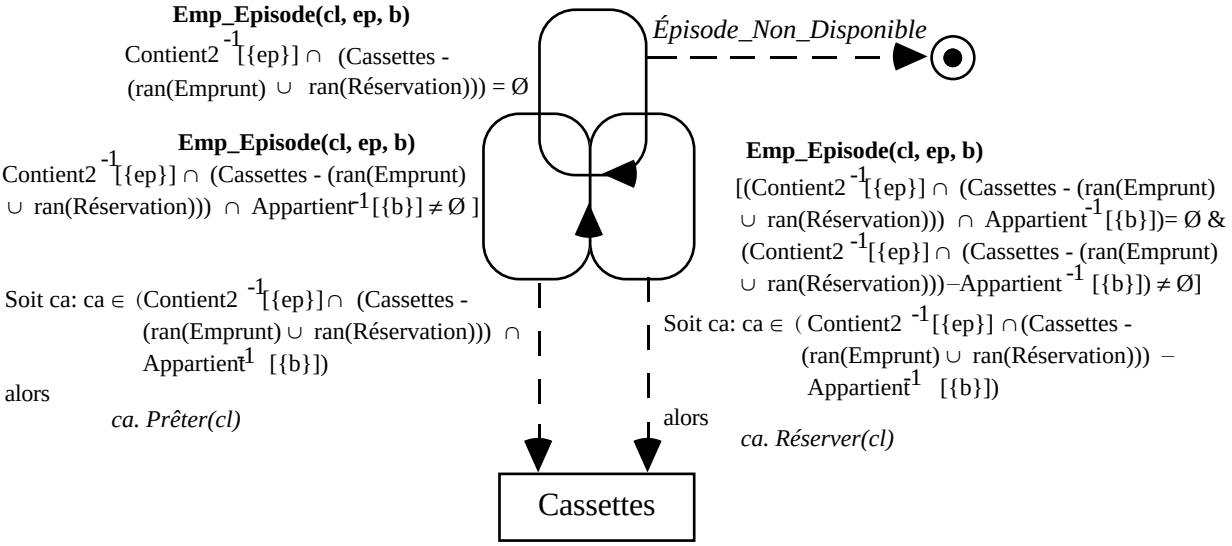


Figure 16. *Diagramme administrateur pour un emprunt d'épisode*

Diagramme E/T de Cassette :

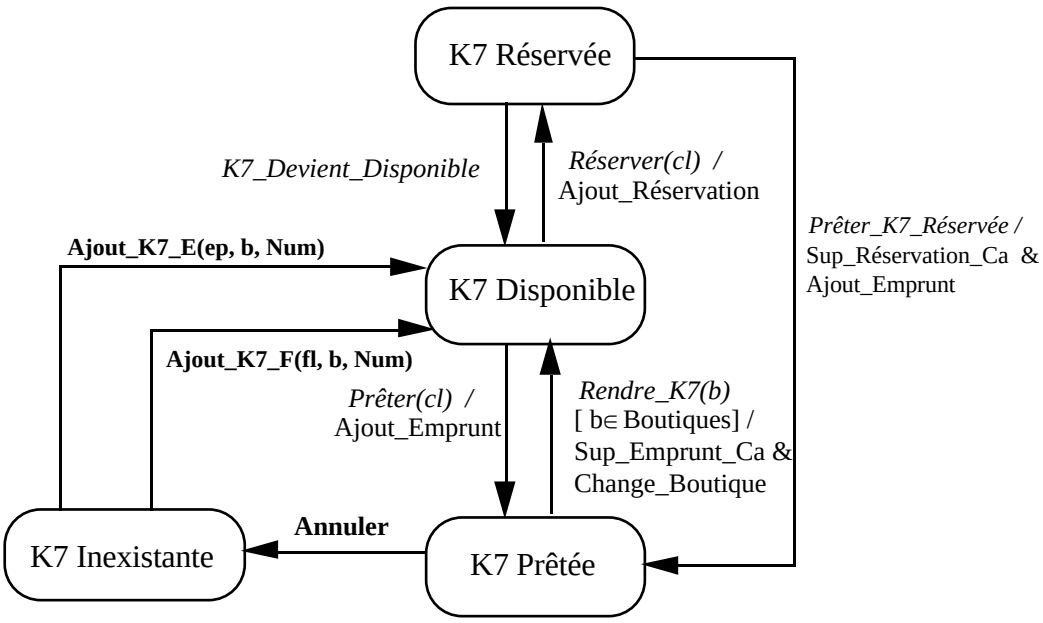


Figure 17. *Diagramme E/T de cassette*

III.3. Description des états et génération des squelettes d'opération B à partir des diagrammes E/T et scénarios

III.3.1. Description des états

III.3.1.1. Description des états du diagramme E/T de Client

Dans le diagramme E/T de l'entité Client, il y a quatre états : Client_Inexistant, Client_est_en_règle, Client_n'est_pas_en_règle et Client_douteux. Nous allons spécifier chaque état par un prédicat à l'aide des ensembles, des variables concernant l'entité Client.

Client_Inexistant(cl) ≙ cl ∈ CLIENTS - Clients ;

Client_est_en_règle(cl) ≙ cl ∈ Clients ∧ {ca | ca ∈ Emprunt[{cl}] ∧ (Date - Date_E(cl,ca)) > 8} = {} ;

Client_n'est_pas_en_règle(cl) ≙ cl ∈ Clients ∧ {ca | ca ∈ Emprunt[{cl}] ∧ (Date - Date_E(cl,ca)) > 8} ≠ {} ;

Client_douteux(cl) ≙ cl ∈ Carte_Retirée $^{-1}$ [{TRUE}]

III.3.1.2. Description des états du diagramme E/T de Cassette

Il s'agit de définir les quatre états : K7_Inexistante, K7_Disponible, K7_Prêtée, K7_Réservée. Nous avons quatre prédicats suivants :

K7_Inexistant(ca) ≙ ca ∈ CASSETTES - Cassettes ;

K7_Disponible(ca) ≙ ca ∈ (Cassettes - (ran(Emprunt) ∪ ran(Réservation)));

K7_Prêtée(ca) ≙ ca ∈ ran(Emprunt) ;

K7_Réservée(ca) ≙ ca ∈ ran(Réservation)

III.3.2. Génération de squelettes d'opération B à partir des diagrammes E/T et scénarios

III.3.2.1. L'opération Création_Client

Cette opération consiste à décrire l'effet du scénario associé à l'événement externe déclencheur Création_Client. En appliquant la règle D7_0, nous obtenons :

Création_Client(Num, Nom, Adresse, Nb_Max)
PRE CLIENTS- Clients ≠ Ø ∧
Num ∈ NAT ∧ Num ∉ ran(Num_Cl) ∧
Nom ∈ STRING ∧ Adresse ∈ STRING ∧ Nb_Max ∈ NAT

THEN **ANY** cl **WHERE** cl ∈ CLIENTS- Clients **THEN**
S /* Substitution correspondant à l'ajout du nouveau client cl*/
END
END

III.3.2.2. L'opération Création_K7_Film

Cette opération correspond au scénario Création_K7_Film. Nous appliquons la règle D7_O et nous obtenons :

Création_K7_Film(fl, bb, Num)
PRE CASSETTES - Cassettes ≠ Ø ∧
fl ∈ Films ∧
bb ∈ Boutiques ∧
Num ∈ NAT ∧ Num ∉ ran(Num_K7)
THEN **ANY** ca **WHERE** ca ∈ CASSETTES - Cassettes **THEN**
S /* Substitution correspondant à l'ajout
d'une nouvelle cassette ca pour le film fl et
à l'ajout d'un lien Appartient entre ca et bb */
END
END

III.3.2.3. L'opération Création_K7_Épisode

Cette opération correspond au scénario Création_K7_Épisode. De la même façon que l'opération précédente, nous obtenons :

Création_K7_Épisode(ep, bb, Num)
PRE CASSETTES - Cassettes ≠ Ø ∧
ep ∈ Épisodes ∧
bb ∈ Boutiques ∧
Num ∈ NAT ∧ Num ∉ ran(Num_K7)
THEN **ANY** ca **WHERE** ca ∈ CASSETTES - Cassettes **THEN**
S /* Substitution correspondant à l'ajout
d'une nouvelle cassette ca pour l'épisode ep et
à l'ajout d'un lien Appartient entre ca et bb */
END
END

III.3.2.4. L'opération Emprunt_Film

Cette opération décrit l'effet de l'événement externe déclencheur Emprunt_Film sur les données du système. Elle consiste à sélectionner une cassette disponible pour un emprunt ou une réservation d'un film. Nous appliquons les règles D6_O, D7_O pour spécifier cette opération. Nous avons donc :

Emprunt_Film(cl, fl, bb)
PRE Client_est_en_règle(cl) ∧
Card(Emprunt[{cl}] + Réservation[{cl}]) ≤ Nb_Max(cl) ∧
fl ∈ Films ∧
bb ∈ Boutiques
THEN **IF** $Contient1^{-1}[\{fl\}] \cap Appartient^{-1}[\{bb\}] \cap$
(Cassettes - (ran(Emprunt) ∪ ran(Réservation))) ≠ Ø
THEN
ANY ca **WHERE** ca ∈ $Contient1^{-1}[\{fl\}] \cap Appartient^{-1}[\{bb\}]$ ∩ (Cassettes - (ran(Emprunt) ∪ ran(Réservation)))
THEN S /* Substitution correspondant à l'ajout d'un lien emprunt entre cl et ca */
END
ELSE
IF ($Contient1^{-1}[\{fl\}] \cap Appartient^{-1}[\{bb\}] \cap$ (Cassettes - (ran(Emprunt) ∪ ran(Réservation))) = Ø) ∧ ($Contient1^{-1}[\{fl\}] \cap$ (Cassettes - (ran(Emprunt) ∪ ran(Réservation))) - $Appartient^{-1}[\{bb\}]$) ≠ Ø
THEN **ANY** ca **WHERE** ca ∈ ($Contient1^{-1}[\{fl\}] \cap$ (Cassettes - (ran(Emprunt) ∪ ran(Réservation))) - $Appartient^{-1}[\{bb\}]$)
THEN
S /* Substitution correspondant à l'ajout d'un lien réservation entre cl et ca */
END
ELSE S /* Substitution correspondant à l'envoi de message vers l'extérieur */

END

END

END

III.3.2.5. L'opération Emprunt_Épisode

Dans ce cas, l'opération décrit l'effet de l'événement externe déclencheur Emprunt_Épisode sur les données du système. Elle consiste à sélectioner une cassette disponible pour un emprunt ou une réservation d'un épisode. Nous appliquons les règles D6_O, D7_O pour spécifier cette opération. Nous avons donc :

Emprunt_Épisode(cl, ep, bb)
PRE Client_est_en_règle(cl) ∧
Card(Emprunt[{cl}] + Réservation[{cl}]) ≤ Nb_Max(cl) ∧
ep ∈ Épisodes ∧ bb ∈ Boutiques
THEN **IF** Contient2^{-1}[{ep}] ∩ Appartient^{-1}[{bb}] ∩
(Cassettes - (ran(Emprunt) ∪ ran(Réservation))) ≠ Ø
THEN
ANY ca **WHERE** ca ∈ (Contient2^{-1}[{ep}] ∩ Appartient^{-1}[{bb}]
∩ (Cassettes - (ran(Emprunt) ∪ ran(Réservation))))
THEN S /* Substitution correspondant à l'ajout
d'un lien emprunt entre cl et ca */
END
ELSE
IF (Contient2^{-1}[{ep}] ∩ Appartient^{-1}[{bb}] ∩
(Cassettes - (ran(Emprunt) ∪ ran(Réservation))) = Ø) ∧
(Contient2^{-1}[{ep}] ∩ (Cassettes - (ran(Emprunt)
∪ ran(Réservation))) - Appartient^{-1}[{bb}]) ≠ Ø
THEN **ANY** ca **WHERE** ca ∈ (Contient2^{-1}[{ep}] ∩
(Cassettes - (ran(Emprunt) ∪ ran(Réservation))))
- Appartient^{-1}[{bb}]
THEN
S /* Substitution correspondant à l'ajout
d'un lien réservation entre cl et ca */
END
ELSE S /* Substitution correspondant à l'envoi de
message vers l'extérieur */

END

END

END

III.3.2.6. L'opération Emprunt_Réservation

En appliquant la règle D6_O, nous obtenons :

Emprunt_Réservation(cl, ca)
PRE Client_est_en_règle(cl) ∧
ca ∈ Réservation[{cl}]
THEN /* Substitution correspondant à la suppression de la réservation de ca et à l'ajout d'un lien emprunt entre cl et ca */
END

Remarquons que le corp de l'opération générée ne contient pas la structure IF... THEN ... ELSE" car dans le diagramme E/T de Cassette, la condition de déclenchement n'existe pas et la contrainte sur le prédicat de l'état de la cassette a été vérifiée dans la pré-condition ca ∈ Réservation[{cl}].

III.3.2.7. L'opération Rendre_K7

De la même façon, en appliquant la règle D6_O, nous obtenons l'opération correspondant au scénario Rendre_K7 comme spécifiée ci-dessous :

Rendre_K7(ca, bb)
PRE K7_Prêtée(ca) ∧
bb ∈ Boutiques
THEN /* Substitution correspondant à la suppression du lien emprunt et au changement de la boutique de ca */
END

III.3.2.8. L'opération Retirer_de_la_Liste

En appliquant la règle D2_O, nous obtenons :

Retirer_de_la_Liste(cl)
PRE Emprunt[{cl}] = Ø
THEN /* Substitution correspondant à la suppression d'un client cl et à la suppression de ses réservations */

END

Notons que nous n'avons pas vérifié le prédicat de l'état du client car cette condition a été vérifiée dans la condition de déclenchement.

III.3.2.9. L'opération Résiliation_Client

Cette opération consiste à résilier un client qui n'est pas en règle. Ceci entraîne la suppression de toutes les cassettes qui lui ont été prêtées. Nous appliquons les règles D6_O et D7_O.

Résiliation_Client(cl)
PRE Client_n'est_pas_en_règle(cl) $\wedge$
$\exists$ ca.(ca $\in$ Emprunt[{cl}] => Date - Date_E(cl, ca) > 90)
THEN
/* Substitution correspondant au changement de valeur
de l'attribut Carte_Retirée,
à la suppression de toutes les cassettes empruntées par le client cl,
à la suppression du lien Appartient lié aux cassettes empruntées par
cl, à la suppression des liens Emprunt liés à cl et
à la suppression des liens Réservation liés à cl */
END

III.3.2.10. L'opération Supprime_K7_Réservée

Cette opération consiste à supprimer toutes les cassettes ayant été réservées la veille.

Supprime_K7_Réservée(Ens_K7)
PRE Ens_K7 = {ca | ca $\in$ ran(Réservation)
$\wedge$ Date_Res(Réservation^{-1}(ca), ca) = Date -1}
THEN /* Substitution correspondant à la suppression
de réservation de l'ensemble Ens_K7 */
END

III.4. Achèvement des machines obtenues par les opérations de base et les invariants correspondant aux contraintes d'intégrité non exprimables graphiquement

Cette troisième étape consiste à déterminer les opérations élémentaires des machines entités et associations, à compléter le corps des opérations de la machine MA_Interface par les

opérations élémentaires et à ajouter l'invariant correspondant aux contraintes d'intégrité qui ne sont pas exprimables graphiquement. Dans cette étude de cas, la contrainte : "le nombre de cassettes empruntées et réservées par un client ne doit pas dépasser le nombre maximum autorisé" est spécifiée comme suit :

$\forall$ cl.(cl $\in$ Clients => (Card(Emprunt[{cl}]) + Card(Réservation[{cl}])) $\leq$ Nb_Max(cl))

Pour les opérations utilisant la structure "IF ... THEN ... ELSE" avec envoi de message vers l'extérieur, nous proposons d'ajouter un paramètre de sortie de l'opération qui permet de renvoyer les résultats de l'appel de l'opération afin de différencier les différents cas de figure. Dans cette étude de cas, on retient cette proposition pour les deux opérations : Emprunt_Film et Emprunt_Épisode pour lesquelles nous avons respectivement ajouté les paramètres de sortie Résultat_EF et Résultat_EE.

Pour plus de clarté, nous proposons de décrire chaque expression lorsqu'elle est complexe par une définition portant un nom explicite. Ces définitions remplacent par la suite les expressions correspondantes.

Voici les machines obtenues :

MACHINE	MA_Boutique
SETS	BOUTIQUES
VARIABLES	Boutiques, Nom_B, Adresse_B
INVARIANT	Boutiques $\subseteq$ BOUTIQUES $\wedge$
	Nom_B $\in$ Boutiques $\rightarrow$ STRING $\wedge$
	Adresse_B $\in$ Boutiques $\rightarrow$ STRING
INITIALISATION	Boutiques, Nom_B, Adresse_B := {}, {}, {}
END	

MACHINE	MA_Client
SETS	CLIENTS
VARIABLES	Clients, Num_Carte, Nom_Cl, Adresse_Cl, Nb_Max, Carte_Retirée
INVARIANT	Clients $\subseteq$ CLIENTS $\wedge$
	Num_Carte $\in$ Clients $\rightarrowtail$ NAT $\wedge$
	Nom_Cl $\in$ Clients $\rightarrow$ STRING $\wedge$
	Adresse $\in$ Clients $\rightarrow$ STRING $\wedge$

Nb_Max ∈ Clients → NAT ∧

Carte_Retiréé ∈ Clients → BOOLEAN

INITIALISATION Clients, Num_Carte, Nom_Cl := {}, {}, {} ||

Adresse, Nb_Max, Carte_Retirée := {}, {}, {}

OPERATIONS

Ajout_Client(cl, num_carte, nom_cl, adresse_cl, nb_max) =

PRE cl ∈ CLIENTS - Clients ∧

num_carte ∈ NAT ∧ num_carte ∉ ran(Num_Carte) ∧

nom_cl ∈ STRING ∧

adresse_cl ∈ STRING ∧ nb_max ∈ NAT

THEN Clients := Clients ∪ {cl} ||

Num_Carte := Num_Carte ∪ {cl ↦ num_carte} ||

Nom_Cl := Nom_Cl ∪ {cl ↦ nom_cl} ||

Adresse_Cl := Adresse_Cl ∪ {cl ↦ adresse_cl} ||

Nb_Max := Nb_Max ∪ {cl ↦ nb_max} ||

Carte_Retirée := Carte_Retirée ∪ {cl ↦ FALSE}

END ;

Sup_Client(Cl) =

PRE Cl ⊆ Clients

THEN Clients := Clients - Cl ||

Num_Carte := Cl ⩤ Num_Carte ||

Nom_Cl := Cl ⩤ Nom_Cl ||

Adresse_Cl := Cl ⩤ Adresse_Cl ||

Nb_Max := Cl ⩤ Nb_Max ||

Carte_Retirée := Cl ⩤ Carte_Retirée

END ;

Chg_Carte(cl) =

PRE cl ∈ Clients

THEN Carte_Retirée(cl) := TRUE

END

END

MACHINE MA_Film

SETS FILMS

VARIABLES Films, Num_Fl, Titre_Fl, Genre

INVARIANT Films $\subseteq$ FILMS $\wedge$ Num_Fl $\in$ Films $\rightarrowtail$ NAT $\wedge$
Titre_Fl $\in$ Films $\rightarrow$ STRING $\wedge$ Genre $\in$ Films $\rightarrow$ STRING

INITIALISATION Films, Num_Fl , Titre_Fl, Genre := {}, {}, {}, {}

END

MACHINE MA_Épisode

USES MA_Film

SETS ÉPISODES

VARIABLES Épisodes, Num_Ep, Titre_Ep, Possède

INVARIANT Épisodes $\subseteq$ ÉPISODES $\wedge$
Num_Ep $\in$ Épisodes $\rightarrowtail$ NAT $\wedge$
Titre_Ep $\in$ Épisodes $\rightarrow$ STRING $\wedge$
Possède^{-1} $\in$ Épisodes $\rightarrow$ Films

INITIALISATION
Épisodes, Num_Ep , Titre_Ep, Possède := {}, {}, {}, {}

END

MACHINE MA_Cassette

USES MA_Film, MA_Épisode

SETS CASSETTES

VARIABLES Cassettes, Num_K7, Contient1, Contient2

INVARIANT Cassettes $\subseteq$ CASSETTES $\wedge$
Num_K7 $\in$ Cassettes $\rightarrowtail$ NAT $\wedge$
Contient1 $\in$ Cassettes $\nrightarrow$ Films $\wedge$
Contient2 $\in$ Cassettes $\nrightarrow$ Épisodes $\wedge$
dom(Contient1) $\cap$ dom(Contient2) = Ø $\wedge$
dom(Contient1) $\cup$ dom(Contient2) = Cassettes

INITIALISATION
Cassettes, Num_K7, Contient1, Contient2 := {}, {}, {}, {}

OPERATIONS

Ajout_Cassette_F(ca, num_ca, fl) =

PRE ca $\in$ CASSETTES - Cassettes $\wedge$
num_ca $\in$ NAT $\wedge$ num_ca $\notin$ ran(Num_K7) $\wedge$
fl $\in$ Films

THEN Cassettes := Cassettes ∪ {ca} ||
Num_K7 := Num_K7 ∪ {ca ↦ num_ca } ||
Contient1 := Contient1 ∪ {ca ↦ fl }
END ;
Ajout_Cassette_E(ca, num_ca, ep) =
PRE ca ∈ CASSETTES - Cassettes ∧
num_ca ∈ NAT ∧ num_ca ∉ ran(Num_K7) ∧
ep ∈ Épisodes
THEN Cassettes := Cassettes ∪ {ca} ||
Num_K7 := Num_K7 ∪ {ca ↦ num_ca } ||
Contient2 := Contient1 ∪ {ca ↦ ep }
END ;
Sup_Cassette(Ca) =
PRE Ca ⊆ Cassettes
THEN Cassettes := Cassettes - Ca ||
Num_K7 := Ca ⩤ Num_K7 ||
Contient1 := Ca ⩤ Contient1 ||
Contient2 := Ca ⩤ Contient2
END
END

MACHINE MA_Appartient
USES MA_Cassette, MA_Boutique
VARIABLES Appartient
INVARIANT Appartient ∈ Cassettes → Boutiques
INITIALISATION Appartient := {}
OPERATIONS
Ajout_Appartient(ca, bb) =
PRE ca ∈ CASSETTES ∧ bb ∈ Boutiques
THEN Appartient := Appartient ∪ {ca ↦ bb}
END ;
Change_Boutique(ca, bb) =
PRE ca ∈ Cassettes ∧ bb ∈ Boutiques
THEN Appartient(ca) := bb
END ;

Sup_Appartient_Ca(Ca) =
PRE $Ca \subseteq Cassettes$
THEN $Appartient := Ca \lhd Appartient$ **END**
END

Machine MA_Emprunt
USES MA_Cassette, MA_Client
VARIABLES Emprunt, Date_E
INVARIANT $Emprunt^{-1} \in Cassettes \nrightarrow Clients \wedge$
$Date\_E \in Emprunt \rightarrow NAT$
INITIALISATION Emprunt, Date_E := {}, {}
OPERATIONS
Ajout_Emprunt(cl, ca, date) =
PRE $cl \in Clients \wedge date \in NAT$
$ca \in (Cassettes - ran(Emprunt)) \wedge$
THEN $Emprunt := Emprunt \cup \{cl \mapsto ca\} \parallel$
$Date\_E := Date\_E \cup \{cl \mapsto ca \mapsto date\}$
END ;
Sup_Emprunt_Ca(Ca) =
PRE $Ca \subseteq ran(Emprunt)$
THEN $Emprunt := Emprunt \rhd Ca \parallel$
$Date\_E := (Clients * Ca) \lhd Date\_E$
END
END

Machine MA_Réservation
USES MA_Cassette, MA_Client
VARIABLES Réservation, Date_Res
INVARIANT $Réservation^{-1} \in Cassettes \nrightarrow Clients \wedge$
$Date\_Res \in Réservation \rightarrow NAT$
INITIALISATION
Réservation, Date_Res := {}, {}
OPERATIONS
Ajout_Réservation(cl, ca, date) =

PRE $cl \in Clients \wedge$
$ca \in (Cassettes - ran(Réservation)) \wedge$
$date \in NAT$
THEN $Réservation := Réservation \cup \{cl \mapsto ca\} \parallel$
$Date\_Res := Date\_Res \cup \{cl \mapsto ca \mapsto date\}$
END ;
Sup_Réservation_Cl(Cl) =
PRE $Cl \subseteq Clients$
THEN $Réservation := Cl \lhd Réservation \parallel$
$Date\_Res := (Cl * Cassettes) \lhd Date\_Res$
END ;
Sup_Réservation_Ca(Ca) =
PRE $Ca \subseteq Cassettes$
THEN $Réservation := Réservation \rhd Ca \parallel$
$Date\_Res := (Clients * Ca) \lhd Date\_Res$
END
END

MACHINE MA_Interface
INCLUDES MA_Boutique, MA_Client, MA_Film, MA_Épisode, MA_Cassette, MA_Appartient, MA_Emprunt, MA_Réservation
DEFINITIONS

Client_Inexistant(cl)	$\widehat{=}$	$cl \in CLIENTS - Clients ;$
Client_est_en_règle(cl)	$\widehat{=}$	$cl \in Clients \wedge \{ca \mid ca \in Emprunt[\{cl\}] \wedge (Date - Date\_E(cl,ca)) > 8\} = \{\} ;$
Client_pas_en_règle(cl)	$\widehat{=}$	$cl \in Clients \wedge \{ca \mid ca \in Emprunt[\{cl\}] \wedge (Date - Date\_E(cl,ca)) > 8\} \neq \{\} ;$
Client_douteux(cl)	$\widehat{=}$	$cl \in Carte\_Retirée^{-1}[\{TRUE\}] ;$
K7_Inexistant(ca)	$\widehat{=}$	$ca \in CASSETTES - Cassettes ;$
K7_Disponible(ca)	$\widehat{=}$	$ca \in (Cassettes - (ran(Emprunt) \cup ran(Réservation)))$

;

K7_Prêtée(ca)	$\widehat{=}$	$ca \in ran(Emprunt) ;$
K7_Réservée(ca)	$\widehat{=}$	$ca \in ran(Réservation) ;$
Ens_K7_du_film(fl)	$\widehat{=}$	$Contient1^{-1}[\{fl\}] ;$
Ens_K7_de_l'épisode(ep)	$\widehat{=}$	$Contient2^{-1}[\{ep\}] ;$

Ens_K7_de_la_boutique(bb) ≙ *Appartient*$^{-1}$*[{bb}] ;*

Ens_K7_Disponible ≙ *(Cassettes - (ran(Emprunt) ∪ ran(Réservation))*

VARIABLES Date /* représente la date courante */

INVARIANT Date ∈ NAT ∧

ran(Emprunt) ∩ ran(Réservation) = Ø ∧

ran(Contient1) ∩ dom(Possède) = Ø ∧

∀ cl.(cl ∈ Clients => (Card(Emprunt[{cl}]) +

Card(Réservation[{cl}])) ≤ Nb_Max(cl))

OPERATIONS

Création_Client(Num, Nom, Adresse, Nb_Max) =

PRE CLIENTS- Clients ≠ Ø ∧

Num ∈ NAT ∧

Num ∉ ran(Num_Carte) ∧

Nom ∈ STRING ∧

Adresse ∈ STRING ∧

Nb_Max ∈ NAT

THEN **ANY** cl **WHERE** cl ∈ CLIENTS- Clients THEN

Ajout_Client(cl, Num, Nom, Adresse, Nb_Max)

END

END ;

Création_K7_Film(fl, bb, Num) =

PRE CASSETTES - Cassettes ≠ Ø ∧

fl ∈ Films ∧

bb ∈ Boutiques ∧

Num ∈ NAT ∧ Num ∉ ran(Num_K7)

THEN **ANY** ca **WHERE** ca ∈ CASSETTES - Cassettes **THEN**

Ajout_Cassette_F(ca, Num, fl) ||

Ajout_Appartient(ca, bb)

END

END ;

Création_K7_Épisode(ep, bb, Num) =

PRE CASSETTES - Cassettes ≠ Ø ∧ ep ∈ Épisodes ∧

bb ∈ Boutiques ∧

Num ∈ NAT ∧ Num ∉ ran(Num_K7)

THEN **ANY** ca **WHERE** ca ∈ CASSETTES - Cassettes **THEN**

Ajout_Cassette_E(ca, Num, ep) ||
Ajout_Appartient(ca, bb)
END
END ;
Résultat_EF <- Emprunt_Film(cl, fl, bb)
PRE Client_est_en_règle(cl) ∧
card(Emprunt[{cl}] ∪ Réservation[{cl}]) ≤ Nb_Max(cl) ∧
fl ∈ Films ∧ bb ∈ Boutiques
THEN **IF** Ens_K7_du_film(fl) ∩ Ens_K7_de_la_boutique(bb) ∩
Ens_K7_Disponible ≠ Ø
THEN
ANY ca **WHERE** ca ∈ Ens_K7_du_film(fl) ∩
Ens_K7_de_la_boutique(bb) ∩
Ens_K7_Disponible
THEN
Ajout_Emprunt(cl, ca, Date) ||
Résultat_EF := 0
END
ELSE
IF (Ens_K7_du_film(fl) ∩ Ens_K7_de_la_boutique(bb) ∩
Ens_K7_Disponible = Ø) ∧
(Ens_K7_du_film(fl) ∩ Ens_K7_Disponible -
Ens_K7_de_la_boutique(bb)) ≠ Ø
THEN
ANY ca **WHERE** ca ∈ Ens_K7_du_film(fl) ∩
Ens_K7_Disponible -
Ens_K7_de_la_boutique(bb)
THEN *Ajout_Réservation(cl, ca, Date)* ||
Résultat_EF := 1
END
ELSE *Résultat_EF := 2*
END
END
END ;
Résultat_EF <- Emprunt_Épisode(cl, ep, bb) =

PRE Client-est-en-règle(cl) ∧
card(Emprunt[{cl}] ∪ Réservation[{cl}]) ≤ Nb_Max(cl) ∧
ep ∈ Épisodes ∧
bb ∈ Boutiques

THEN **IF** Ens_K7_de_l'épisode(ep) ∩ Ens_K7_de_la_boutique(bb) ∩ Ens_K7_Disponible ≠ Ø

THEN

ANY ca **WHERE** ca ∈ Ens_K7_de_l'épisode(ep) ∩ Ens_K7_de_la_boutique(bb) ∩ Ens_K7_Disponible

THEN *Ajout_Emprunt(cl, ca, Date)* ||
Résultat_EE := 0

END

ELSE

IF (Ens_K7_de_l'épisode(ep) ∩ Ens_K7_de_la_boutique(bb) ∩ Ens_K7_Disponible = Ø) ∧
(Ens_K7_de_l'épisode(ep) ∩ Ens_K7_Disponible - Ens_K7_de_la_boutique(bb)) ≠ Ø

THEN

ANY ca **WHERE** ca ∈ Ens_K7_de_l'épisode(ep) ∩ Ens_K7_Disponible - Ens_K7_de_la_boutique(bb)

THEN *Ajout_Réservation(cl, ca, Date)* ||
Résultat_EE := 1

END

ELSE

Résultat_EE := 2

END

END

END ;

Emprunt_Réservation(cl, ca) =

PRE Client_est_en_règle(cl) ∧
ca ∈ Réservation[{cl}]

THEN *Sup_Réservation_Ca({ca})* ||
Ajout_Emprunt(cl, ca)

END ;
Rendre_K7(ca, bb) =
PRE K7_Prêtée(ca) ∧
bb ∈ Boutiques
THEN *Sup_Emprunt_Ca({ca})* ||
Change_Boutique(ca, bb)
END ;
Retirer_de_la_Liste(cl) =
PRE Client_est_en_règle(cl) ∧
Emprunt[{cl}] = Ø
THEN *Sup_Client({cl})* ||
Sup_Réservation_Cl({cl})
END ;
Résiliation_Client(cl) =
PRE Client_n'est_pas_en_règle(cl) ∧
∃ ca.(ca ∈ Emprunt[{cl}] => Date - Date_E(cl, ca) > 90)
THEN
Chg_Carte(cl) ||
Sup_Cassette(Emprunt[{cl}]) ||
Sup_Appartient_Ca(Emprunt[{cl}]) ||
Sup_Emprunt_Ca(Emprunt[{cl}]) ||
Sup_Réservation_Ca(Réservation[{cl}])
END ;
Supprime_K7_Réservée(Ens_K7) =
PRE Ens_K7 = {ca | ca ∈ ran(Réservation)
∧ Date_Res(Réservation^{-1}(ca), ca) = Date -1}
THEN
Sup_Réservation_Ca(Ens_K7)
END
END

III.4. Validation de la spécification

Nous présentons ci-dessous le tableau indiquant pour chaque machine le nombre d'obligations de preuve générées, le nombre de preuves démontrées automatiquement et le nombre de preuves restant à démontrer en mode interactif.

Machines	Obligations de preuve	Preuves automatiques	Preuves Interactives
Boutique	3	3	0
Client	31	31	0
Film	5	5	0
Épisode	5	5	0
Cassette	21	19	2
Appartient	Aucune OP n'a été retenue par le générateur d'OP (preuves triviales)		
Emprunt	6	3	3
Réservation	8	4	4
Interface	84	63	21
Total	163	133	30

En conclusion, cette étape de validation nous a permis de nous assurer que la spécification obtenue en appliquant les règles de dérivation est cohérente, malgré la complexité de l'étude de cas : les preuves restant à démontrer en mode interactif ne présentent aucune contradiction apparente.