UNIVERSITÉ DE NANTES
Faculté des sciences et des techniques

Le projet CordiFormes : une plate-forme pour la construction de modeleurs déclaratifs

THÈSE DE DOCTORAT

Spécialité : Informatique
École doctorale : Sciences pour l'ingénieur de Nantes

Présentée et soutenue publiquement par :

DESMONTILS Emmanuel

le 14 janvier 1998,
à la faculté des sciences et des techniques de Nantes,

devant le jury ci-dessous :

Président : Michel LUCAS, Professeur - École Centrale, Nantes
Rapporteurs : Didier DUBOIS, Directeur de Recherche CNRS - IRIT, Toulouse
Bernard PEROCHE, Professeur - École des Mines, Saint-Étienne
Examinateurs : Bruno ARNALDI, Professeur - INSA de Rennes
Gérard HEGRON, Professeur - École des Mines, Nantes
Daniel PACHOLCZYK, Professeur - Université d'Angers

Directeur de thèse : Michel LUCAS
Laboratoire : Institut de Recherche en Informatique de Nantes (IRIN)

N° ED 82-301

REMERCIEMENTS

Je remercie Michel LUCAS d'avoir encadré cette thèse et de m'avoir toujours justement conseillé et soutenu. Il a su m'encourager dans les nouvelles orientations que je me proposais d'explorer. Sa patience et sa sagesse m'ont toujours bien guidé tout au long de ces années qui se sont révélées très riches tant au niveau humain qu'au niveau scientifique.

Je remercie vivement Messieurs Didier DUBOIS et Bernard PEROCHE pour avoir accepté d'être rapporteur de ce travail, Messieurs Bruno ARNALDI, Gérard HÉGRON et Daniel PACHOLCZYK d'avoir bien voulu faire partie de mon jury et Michel LUCAS pour l'avoir efficacement présidé.

Je remercie aussi Messieurs Didier DUBOIS et Daniel PACHOLCZYK pour leurs remarques qui ont permis l'amélioration de ce manuscrit.

Je remercie aussi Monsieur Daniel PACHOLCZYK pour la collaboration très fructueuse que nous avons eu depuis maintenant plus de deux ans. Nos journées de travail furent denses et particulièrement enrichissantes. La diversité faisant la richesse, cette collaboration m'a permis de découvrir un monde de la Recherche en Intelligence Artificielle bien différent de celui de la Synthèse d'Images.

Merci à Jean-Yves MARTIN. Forçat du travail mais toujours disponible, il n'a pas hésité à s'engager à mes côtés dans le projet CordiFormes. Ses qualités en font un collègue tout à fait remarquable.

Je remercie Christian COLIN (toujours l'esprit vif et combatif, sauf en avion !) et Jean-Yves MARTIN pour avoir contribué, par leurs commentaires et leurs remarques à l'amélioration de ce rapport.

Je remercie l'équipe MGII et plus particulièrement Marc DANIEL, Éric LANGUÉNOU, Dominique MARTIN, Jean-Yves MARTIN et Philippe MARTIN pour m'avoir régulièrement écouté, conseillé et encouragé. L'ambiance de travail de cette équipe ne peut qu'aider au bon déroulement d'une thèse. Merci aussi aux « camarades » qui ont partagé avec moi le bureau (Olivier « Aristide » GRANGE et Vincent ROSSIGNOL) ou les soucis d'une telle épreuve (Emmanuel MALGRAS et Jean-Philippe MOUNIER).

Je remercie le groupe SCHÉMA dont les interminables palabres ont été plus que profitables (si si Jean-Philippe !). Merci aussi au groupe GEODE dans son entier. Les réflexions et

les travaux effectués dans le cadre de ce groupe de travail ont été particulièrement bénéfiques pour cette thèse.

Je remercie l'IRIN et son Directeur, Monsieur Jean-François NICAUD, de m'avoir accueilli et permis de mener à bien ces recherches.

Je n'oublie pas non plus, les personnes du département informatique de l'École Centrale de Nantes et, particulièrement, Alix POTET et Jean HAMÉON qui m'ont accueilli avec gentillesse au tout début, alors que je n'étais qu'un « sans bureau fixe » !

Je remercie enfin ma famille et, tout particulièrement ma femme pour m'avoir encouragé, supporté, motivé (et bien d'autres choses encore) pendant ces longues années de faculté et surtout pendant ces trois dernières.

à Christelle

TABLE DES MATIÈRES

TABLES DES FIGURES, TABLEAUX ET ALGORITHMES

INTRODUCTION

1. La modélisation déclarative

Depuis son apparition et encore aujourd'hui, l'essentiel des efforts en synthèse d'images se porte sur la définition de modèles géométriques performants et de techniques de visualisation rapides et réalistes. Par contre, l'aide à la conception des scènes à visualiser a été assez peu abordée en dehors d'outils CAO généralement de bas niveau.

La *modélisation déclarative* ([LMM89], [Luc91] et [CDMM97b]) se propose de fournir des outils de haut niveau pour concevoir ces scènes. Elle a donné naissance à de nombreuses études dont notamment les projets *PolyFormes* (modélisation déclarative de polyèdres [MaM89], [MaM90] et [MaM93]), *AutoFormes* (modélisation déclarative d'automates cellulaires [Mar90]), *VoluFormes* (contrôle spatial déclaratif [Chau94b]), *CurviFormes* (modélisation déclarative de courbes [Dem94], [Dan95] et [DaL96]), *MégaFormes* (modélisation déclarative de sites mégalithiques [LuP94], [Pou96] et [PoL96]).

Comme nous l'avons montré lors d'études exhaustives ([Des95a] et [LuD95]), les travaux en modélisation déclarative, telle que nous la définissons dans la suite, sont essentiellement effectués par les membres du groupe GEODE. C'est une équipe du GDR-PRC AMI dont le thème de recherche est la modélisation déclarative. Elle est constituée de l'équipe MGII de l'Institut de Recherche en Informatique de Nantes, du CERMA de l'École d'Architecture de Nantes et de l'équipe de synthèse d'images de l'École des Mines de Nantes.

En dehors de ce groupe, il existe aussi des travaux à l'Université de Limoges ([Ple94] et [BoP96]) et à l'Institut de Recherche en Informatique de Toulouse ([Dje91], [GCR93] et [KGC97]). On trouve aussi un certain nombre d'études se rapprochant plus ou moins partiellement de la modélisation déclarative sous les dénominations :

- *Cooperative Computer Aided Design* ([Koc94] et [Koc97]) ;
- *Generative Expert System*, *Generative Tool* ou *Generative CAD* ([Woo91], [Khe95], [RMD96] et [RMS96]) ;
- *Automated Architectural Design* ou *Knowledge-based Computer Aided Architectural Design* ([RMD96] et [RMS96]) ;
- *Scene Description* ([ADG84], [DAG86], [GFT92] et [GAT96]) ;
- *Simulation inverse ou problème inverse* ([KaB92], [SDS93] et [PRJ97]).

La plupart de ces projets mettent en œuvre des techniques similaires. Malgré cela, pour développer un modeleur déclaratif, un concepteur est systématiquement obligé de tout refaire

(outils de saisie de la description, algorithme de génération des solutions et outils pour en prendre connaissance…). Il apparaît donc nécessaire :

- de formaliser les connaissances manipulées en prenant en compte des aspects importants liés à l'utilisation d'un langage de description ;
- de proposer une plate-forme pour la construction de modeleurs déclaratifs.

Cette plate-forme exploitera au mieux le formalisme et devra respecter un certain nombre de caractéristiques dont les principales sont la simplicité de développement, la souplesse de programmation, la réutilisabilité et l'extensibilité des connaissances, l'efficacité des modeleurs produits et l'utilisation de méthodes génériques classiques.

Dans cette introduction, nous présenterons d'abord les modeleurs déclaratifs et leurs différences par rapport aux modeleurs traditionnels (section 2). Puis, nous verrons les phases essentielles de la modélisation déclarative (section 3). Nous terminerons enfin par une présentation des deux grandes parties de ce document (section 4).

2. Modélisation classique et modélisation déclarative

Les processus de conception à l'aide de modeleurs traditionnels et de modeleurs déclaratifs sont relativement différents. S'ils ont tout les deux l'objectif de construire une scène pour résoudre un problème donné, les moyens pour y parvenir ne mettent pas en œuvre les mêmes mécanismes.

Avec un modeleur traditionnel, la création d'une scène nécessite deux étapes :

- conception, détermination des actions élémentaires et vérification : à la charge de l'opérateur ;
- calcul et visualisation : à la charge du système.

L'utilisateur du modeleur doit, à partir de son idée, déterminer les spécifications de son problème. Puis, à l'aide de l'objet mental, il fournit les instructions élémentaires permises par le modeleur et nécessaires à la conception de l'objet. Le modeleur permet alors de visualiser l'objet mental ainsi décrit. Ensuite, un ensemble important de tests permet de valider cette construction. S'ils sont négatifs, l'utilisateur doit revoir son objet mental et modifier les instructions entrées. Le processus de construction recommence ainsi jusqu'à obtention d'une solution satisfaisante. Si l'utilisateur n'arrive pas à atteindre cette solution, il doit reprendre ses spécifications et tout le processus de conception.

Un modeleur déclaratif s'appuie, quant à lui, sur un principe différent : «*l'objectif de la modélisation déclarative de formes est de permettre d'engendrer des formes (ou des ensembles de formes) par la simple donnée d'un ensemble de propriétés ou de caractéristiques*.

L'ordinateur est chargé d'explorer l'univers des formes potentielles, afin de sélectionner celles correspondant à la définition donnée. Le concepteur n'a plus qu'à choisir, à l'aide d'outils appropriés, la ou les formes qui lui conviennent» [Luc93]. Le processus de conception est donc composé de trois étapes essentielles ([LMM89], [Luc91] et [CDMM97b] ; Figure 1) : *la description* du problème et de la solution désirée, *la génération* (le calcul) de l'ensemble des solutions valides (répondant à la description) et *la prise de connaissance* de des solutions obtenues. Aucune phase de tests n'est donc nécessaire puisque le modeleur produit des solutions en adéquation avec la demande de l'utilisateur. Si ce dernier ne trouve pas de solutions qui lui conviennent, il n'a alors qu'à modifier les propriétés fournies et à relancer une exploration.

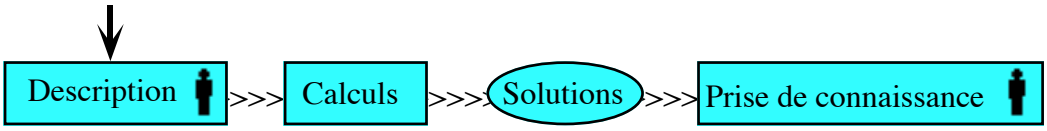

Figure 1. Vue globale d'un modeleur déclaratif [CDMM97b]

Une différence importante entre les deux approches est la quantité et le type de travail réalisés par l'homme. Les modeleurs déclaratifs assurent une part de travail beaucoup plus importante que les modeleurs traditionnels, laissant l'homme se concentrer sur les tâches de haut niveau. Ainsi, l'opérateur se trouve libéré des calculs et peut se focaliser davantage sur la phase de création.

La Figure 2 illustre cette différence entre les deux types de modeleurs. Les zones claires correspondent au travail effectué par l'homme et les zones foncées à celui réalisé par la machine. Les traits pointillés mettent en valeur la correspondance des phases entre les deux types de modeleurs. La position verticale des boîtes indique la part respective de travail réalisée par l'homme ou la machine pour l'action représentée. ([CDMM97b])

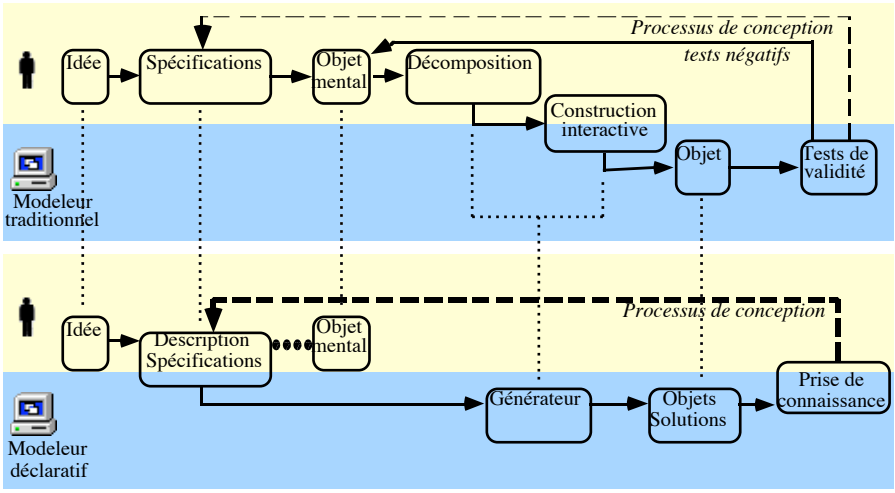

Figure 2. Comparaison de modeleurs traditionnel et déclaratif [CDMM97b]

3. Les phases importantes d'un modeleur déclaratif

3.1. La description

La phase de description de la scène permet à l'utilisateur de décrire à l'aide de propriétés de haut niveau (imprécises ou vagues) ce qu'il veut obtenir. Il n'a pas à s'occuper des détails de construction et peut donner sa description par le moyen le plus adapté en fonction de la propriété décrite et du domaine d'application. Il est possible d'envisager plusieurs moyens de description tels que le « langage naturel » écrit ou parlé ([Chau94], [ADF83], [ADG84], [DAG86], [GFT92] et [GAT96]), les dessins ([Mar90], [SDS93], [Pou94] et [ZHH96]), les graphiques ([Chau94]), les questionnaires et les dialogues classiques ([Mar90], [MaM93], [Chau94], [RMS96], [Lie96]…), les gants de données, la vidéo… Ces média peuvent être utilisés simultanément dans une même description ([LMM89], [Chau94], [Pou94], [GCR93], [KGC97]…).

3.2. La génération

Le calcul des solutions permet de parcourir l'univers des scènes possibles et de sélectionner une ([SDS93], [GAT96], [ZHH96], [PRJ97]…), plusieurs ([RMS96]) ou, de préférence, toutes celles vérifiant la description ([LMM89], [Col90], [Mar90], [Ple91], [Woo91], [Paj94], [Pou94], [Koc94], [Khe95]…). Plusieurs méthodes algorithmiques ont été étudiées pour la phase de génération :

- les arbres de parcours explicites ([Col90], [Mar90], [Paj94], [Pou94], [Khe95]…) ;
- les moteurs d'inférence ([Ple91] et [MaM93]) ;
- les grammaires génératives ([Woo91], [RMS96] et [Koc94]) ;
- les systèmes à base d'arithmétique des intervalles ([SnK92], [Chau94] et [MaM96]) et de contraintes ([Lie96], [GFT92], [GAT96], [ZHH96], [PRJ97] et [Cham97a]) ;
- les simulations inverses ([SDS93])…

Les méthodes de génération ont été caractérisées d'un point de vue global principalement par [Col90] pour les arbres explicites, par [MaM90] pour les moteurs d'inférence et par [Woo91] pour les grammaires génératives.

3.3. La prise de connaissance

La phase de prise de connaissance permet à l'utilisateur de découvrir les solutions à sa description. Pour cela, le modeleur lui propose des outils de haut niveau exploitant aussi bien les caractéristiques des solutions que les propriétés décrites ou déduites lors de la génération. Il dispose d'outils permettant d'explorer l'ensemble des solutions en utilisant différentes techniques comme la classification et l'utilisation de classes d'équivalence ([Cham97b] et [RMD96]), la retouche contrôlée ([GFT92], [GAT96], [RMD96] et [Sir97]), la maîtrise de

l'ordre de parcours des solutions ([Col90] et [Paj94]), le raffinement des solutions ([Col90], [Ple91] et [Pou94]), des vues globales des solutions ([Koc94] et [Koc97])…

De plus, pour chaque solution présentée, il dispose de méthodes pour aider à la compréhension et mettre en évidence les caractéristiques essentielles comme :

- des outils pour « circuler » dans la scène ([RMD96], [GAT96] et [GCR93]) ;
- des calculs de bons points de vue ([Col90], [Ple91], [Dje91], [Pou94], [GFT92], [GAT96] et [RMD96]) et de bons éclairages ([RMD96] et [PRJ97]) ;
- des visualisations à différents niveaux de détail et de précision ([Koc94], [Ple91] et [ZHH96]) ;
- des modes de visualisation différents ([Col90])…

Lorsque l'utilisateur trouve une solution très proche de ce qu'il attend, plutôt que de changer la description et de relancer une génération, il peut modifier une solution par *modification déclarative* en utilisant les propriétés modificatrices ([GCR93], [Chau94], [Paj94] et [Pou94]).

3.4. Spécificité de la modélisation déclarative

Beaucoup de travaux décrivent des outils d'interaction de haut niveau en CAO (par exemple [SDS93], [ZHH96], [RMS96], [RMD96] ou [PRJ97]) mais rarement au niveau des trois phases. Ce qui fait la spécificité et l'originalité de la modélisation déclarative est qu'elle propose de regrouper tous ces outils autour de ses principaux préceptes :

- une description souple, de haut niveau, avec des outils adaptés, la plupart du temps sans s'intéresser aux valeurs et aux coefficients effectifs ;
- une génération exhaustive, efficace et contrôlée des solutions ;
- une prise de connaissance « intelligente » des solutions en utilisant conjointement des outils performants et des connaissances issues de la scène elle-même, de la description et de la génération.

Une étude exhaustive sur les modeleurs déclaratifs est proposée dans [Des95a] et [LuD95]. Une synthèse de leur fonctionnement du point de vue des utilisateurs ainsi qu'une étude des processus de conception qu'ils rendent possibles sont proposées dans [CDMM97b].

3.5. Schémas et notations

Sur la Figure 1 et sur certaines autres figures dans la seconde partie, nous utilisons la même notation. Les actions sont représentées par des rectangles (voir Figure 3) et les informations par des ellipses.

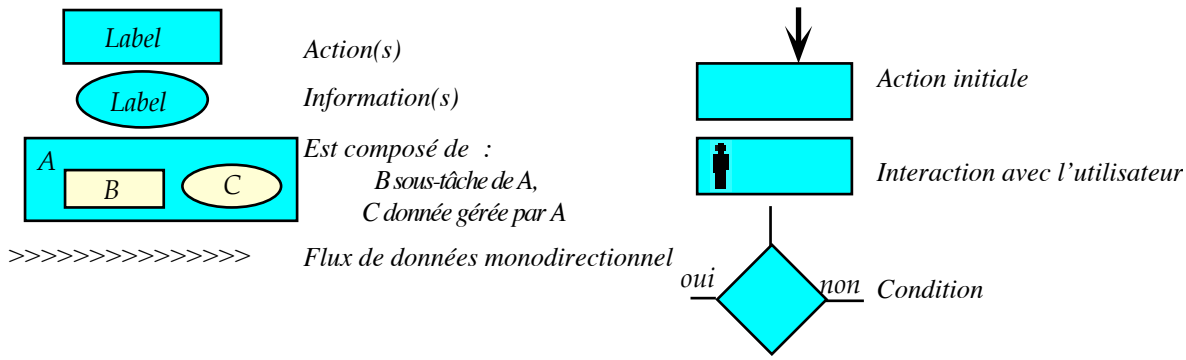

Figure 3. Notation des figures [CDMM97b]

Une action peut être constituée de sous-tâches et de données. Une flèche sur un rectangle indique l'action qui est exécutée en premier. Les actions accessibles directement par l'utilisateur, c'est-à-dire celles avec lesquelles il interagit, sont marquées à l'aide d'un petit humanoïde. Un losange traduit une condition. Enfin, les flèches constituées de chevrons soulignent un échange orienté de données entre deux actions. Elles ne signifient en aucun cas une quelconque notion de séquencement.

Afin de simplifier les références de ce document, nous avons choisi de numéroter les chapitres en indiquant la partie à laquelle ils appartiennent. Par exemple, le chapitre I.5 correspond au cinquième chapitre de la première partie. De même, les définitions sont numérotées en indiquant la partie et le chapitre dans lesquels elles se trouvent. Par exemple, la définition I.3.2 correspond à la seconde définition se trouvant dans le chapitre trois de la première partie.

4. Organisation du document

Nous avons pu constater dans les paragraphes précédents et dans [Des95a] que les travaux en modélisation déclarative sont nombreux et assez récents. Ils portent sur les problèmes de saisie et de construction d'une description, sur les techniques de génération et leur optimisation ainsi que sur les méthodes de visualisation. Ces recherches ont montré, à travers les différentes applications, la faisabilité de cet outil de conception qu'est la modélisation déclarative. Elles ont, pour l'instant, essentiellement porté sur les algorithmes de génération. Il devient nécessaire de regrouper toutes ces études afin de proposer une base de développement de modeleurs déclaratifs.

Pour cela, il est indispensable de bien structurer les connaissances. Nous nous proposons donc dans la première partie de ce document de mettre en place un formalisme permettant de représenter et de gérer les connaissances d'un modeleur déclaratif. Un des fondements des applications déclaratives est la possibilité de donner des descriptions qualitatives de l'univers (en particulier par l'intermédiaire du langage). Il faut donc tenir compte de la richesse des langages de description liée essentiellement au flou et à l'imprécision des actions et des ter-

mes utilisés ([ZHH96] pour l'aspect « schéma sommaire » et [ADF83], [ADG84] et [GFT92] pour le traitement de description en langage naturel). Par conséquent, nous nous attacherons dans la première partie à construire un formalisme de représentation des connaissances et des propriétés basé sur les sous-ensembles flous et l'utilisation d'opérateurs génériques. Après une présentation des propriétés répertoriées dans les différents projets, nous étudierons principalement les propriétés les plus simples que nous appellerons propriétés élémentaires. Nous introduirons ensuite une nouvelle approche de la négation d'une propriété à partir de notions linguistiques. Enfin, nous terminerons cette partie par des propositions de formalisation d'autres propriétés.

Dans la seconde partie, nous proposerons une plate-forme pour la construction d'applications déclaratives : le projet *CordiFormes*. Ce dernier a pour objectif de mettre à la disposition d'un concepteur de modeleur déclaratif un ensemble de structures de données basées sur le formalisme précédemment exposé. La plate-forme propose aussi un ensemble d'outils lui permettant de développer rapidement, simplement et facilement un premier prototype de son application. Le concepteur dispose d'une base de connaissances évolutive proposant les éléments les plus courants en modélisation ainsi que des méthodes génériques performantes. Cette plate-forme possède les caractéristiques essentielles de simplicité, de souplesse et d'efficacité. Ses outils sont répartis sur trois niveaux : le noyau, la couche interface et la couche application prototype. Nous étudierons principalement le niveau de base qu'est le noyau. Ce dernier propose les éléments algorithmiques de base pour l'application ainsi qu'une base de connaissance. La couche interface propose essentiellement des outils de description et de prise de connaissance. Enfin, la couche prototype permet de produire très rapidement une application indépendante primaire permettant d'évaluer le modeleur. Nous terminerons par la présentation de trois exemples simples de modeleurs développés à partir de cette plate-forme : les projets EngloFormes, LinéaFormes et ChromoFormes.

PARTIE I : MODÈLE DE REPRÉSENTATION DES PROPRIÉTÉS

CHAPITRE I.1 : INTRODUCTION

1. Introduction

La *propriété*, trait caractéristique de la scène par rapport à un concept donné, est une notion centrale en modélisation déclarative. Plusieurs auteurs (dont [Col92] et [Chau94b]) ont proposé des formalismes basés sur la manipulation d'intervalles classiques. Toutefois, ils ne semblent pas suffisamment universels, car ils restent très dépendants des propriétés sur lesquelles ils portent ainsi que du domaine d'application.

Contrairement aux approches traditionnelles de la modélisation déclarative utilisant les intervalles classiques, notre approche de *nature semi-qualitative* fait appel à la *théorie des sous-ensembles flous de Zadeh* ([Zad65]). Notre objectif est de concevoir les propriétés tout en généralisant la construction de modificateurs indépendants de celles-ci. On trouve les premiers résultats dans [Des95b], [Des96], [DeP96a], [DeP96b], [DeP97a], [PaD97a], [DeP97b] et [PaD97b]. Nous proposons un cadre formel de représentation des propriétés imprécises des concepts conduisant de façon satisfaisante à l'élaboration d'une scène. Après quelques rappels sur la représentation des propriétés en modélisation déclarative, nous poserons dans ce chapitre les définitions de base introduisant les concepts d'une scène et les propriétés qui lui sont attachées.

Au chapitre I.2, nous proposerons un classement des différentes propriétés en modélisation déclarative selon des critères syntaxiques ainsi que sémantiques. Nous nous intéresserons plus particulièrement au chapitre I.3 à la formalisation des *propriétés dites élémentaires*, c'est-à-dire des énoncés du type « x est m_α P_{ik} », ou encore des énoncés comportant des *propriétés* P_{ik} sur lesquelles opèrent des *modificateurs* m_α. Les *opérateurs flous* f_α introduits nous conduiront à la représentation d'une forme très générale de propriété élémentaire, à savoir l'énoncé « x est f_α m_β P_{ik} » où f_α est un opérateur flou sur la propriété élémentaire m_β P_{ik}. Le point délicat abordé au chapitre I.4 concerne la représentation, en termes de propriétés élémentaires, de la négation apparaissant dans « x n'est pas A ». L'interprétation de nature logique de cette négation n'étant pas satisfaisante, après une introduction à la notion de similarité des ensembles flous, nous proposerons une interprétation linguistique basée sur cette similarité. Au chapitre I.5, nous présenterons des idées de traitement des propriétés issues des propriétés élémentaires (en particulier les propriétés quantifiées et les propriétés modificatrices). Puis, au chapitre I.6, nous aborderons les relations. Enfin, nous conclurons au chapitre I.7 par quelques remarques sur l'intégration de ce modèle dans les différentes phases de la modélisation déclarative.

2. État de l'art

En modélisation déclarative, le vocabulaire plutôt vague utilisé pour décrire les scènes est caractéristique (entre autre au niveau des quantificateurs, des énumérations, des dimensions, des modifications…). Ce vocabulaire pose des problèmes d'interprétation. En effet, il est nécessaire, à un moment ou à un autre, d'attribuer des valeurs particulières aux paramètres de la scène. Mais alors par exemple, comment déterminer une taille définie par "plutôt haute" ? Cela revient à poser le problème de la subjectivité et de l'imprécision d'une description créée par ce type de vocabulaire. Cette subjectivité est d'autant plus importante que le monde visuel d'une personne dépend non seulement de son image rétinienne mais aussi de ses origines culturelles et de son expérience ([Hall66], [Jim97]). Actuellement, le choix fait dans les différentes applications utilisant des termes linguistiques ([DAG86], [Chau92], [Mou94], [Paj94], [Dem94], [Pou94a], [PoL96], [GAT96]...) consiste à définir une échelle où à chaque terme correspond une valeur ou un intervalle. Le choix de la valeur s'effectue alors :

- en prenant la valeur prévue ;
- en faisant un tirage aléatoire dans l'intervalle ;
- en prenant un ensemble de valeurs réparties (d'une manière spécifique) sur l'intervalle.

Il en est de même des coefficients permettant de rendre compte des quantificateurs. Si aucune référence n'est faite à une grandeur donnée, une valeur par défaut ou un tirage aléatoire sur tout ou une partie seulement du domaine est généralement prévu.

Remarque : Dans certains projets, l'utilisateur fournit directement l'intervalle valide ([Koc94] et [RMD96]).

Nous exposerons rapidement les principales modélisations utilisées pour représenter les propriétés selon les travaux précédents. La présentation de la théorie des sous-ensembles flous est proposée dans les annexes 1 et 2.

2.1. Propriétés en modélisation déclarative

[Col92] définit les propriétés comme « *une application de l'univers des formes U_f dans l'ensemble {vrai, faux}, qui à chaque forme associe une valeur booléenne indiquant si la forme possède ou non la propriété* ».

Une propriété est décomposable en deux parties ([Col92] et [Paj94]) :

1. une mesure qui permet d'associer une valeur à une solution par rapport à la propriété attendue ;
2. une quantification (<, >, =, …) qui détermine pour quelles valeurs la propriété peut être considérée comme vérifiée.

Ce quantificateur détermine en fait un intervalle de valeurs pour lesquelles la propriété est vérifiée. Pour une même mesure, il est possible de définir plusieurs quantificateurs donc plusieurs propriétés (associées à des termes linguistiques). La méthode classique en modélisation déclarative pour représenter les termes linguistiques est de partitionner l'ensemble des valeurs possibles en intervalles classiques disjoints. Par exemple, [Chau92] définit les termes concernant la taille des objets par rapport à celle de l'univers (appelée M). On obtient alors les termes et leurs intervalles respectifs suivants (Figure 4) :

- « Minuscule » =]0,M/10] ;
- « Petit » = [M/10,2M/5] ;
- « Moyen » = [2M/5,3M/5] ;
- « Grand » = [3M/5,9M/10] ;
- « Gigantesque » = [9M/10,M[.

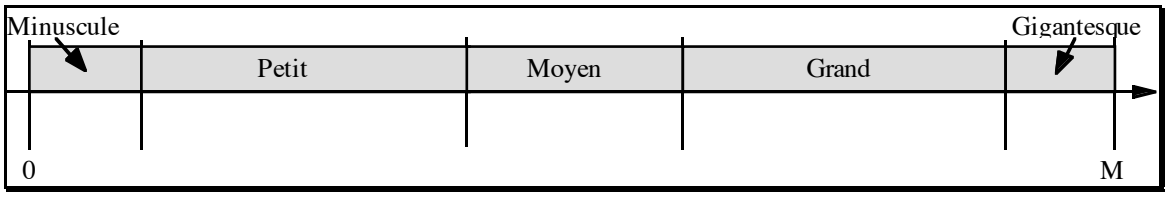

Figure 4. Tailles selon [Chau94b] par rapport à celle de l'univers (appelée M)

2.2. Modificateurs en modélisation déclarative

Très souvent, les propriétés ne sont pas utilisées seules. Le locuteur leur associe des adverbes permettant de modifier ou d'affiner leur intervalle de valeurs. Ils sont appelés *modificateurs*. Classiquement, une propriété associée à un opérateur est définie a priori par le concepteur de manière définitive. Généralement, ils sont associés à des « sous-intervalles » des intervalles de validité des propriétés. C'est le cas d'une première solution de [Chau94b] ainsi que de [Pou94a] et [PoL96].

Pour [Chau94b], les modificateurs « très » et « assez » sont définis dans les intervalles des propriétés (Figure 5).

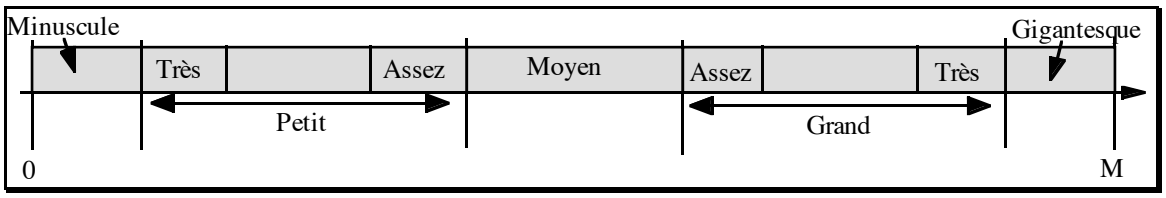

Figure 5. Modificateurs de tailles selon [Chau94b]

[Pou94a] détermine le remplissage d'une matrice de voxels en fonction du pourcentage de voxels utilisés (Figure 6) : « Très peu remplie » = 0 à 22% ; « Peu remplie » = 18 à 42% ; « Remplie » = 38 à 62% ; « Plutôt remplie » = 58 à 82% ; « Très remplie » = 78 à 100%.

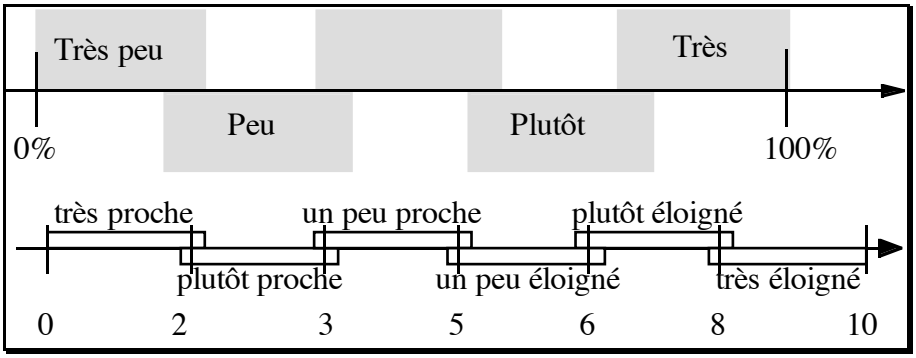

Figure 6. Remplissage d'une matrice de voxels selon [Pou94a] et définitions des distances selon [PoL96]

Entre deux zones, la "frontière" est précise, même si, pour [Pou94a] et [PoL96] deux zones se superposent sur un faible intervalle. Dans ces zones, les valeurs vérifient complètement deux propriétés différentes. On commence à trouver l'idée qu'une même valeur peut vérifier plusieurs propriétés même s'il n'y a encore aucune nuance.

2.3. L'utilisation des sous-ensembles flous

L'idée d'avoir des propriétés vérifiées à un coefficient près est déjà apparue chez [Pou94b], [MaM89], [Bea89] et chez [Ple91] (notion de multi-description et de déformation (orientée plutôt sur une idée de « géométrie approximative »), notion de vérification "à peu près" (pour une gestion des incohérences) ou de règle de modification). Seulement, elle n'était pas traitée de manière claire, systématique et cohérente. L'utilisation de la théorie des sous-ensembles flous de Zadeh ([Zad65]), bien que souvent citée comme intéressante, n'a jamais fait l'objet d'une étude de fond.

[Dje91] a tenté d'introduire dans son formalisme les ensembles flous mais sans les exploiter complètement. Il utilise les fonctions LR trapézoïdales (voir Annexe 2) pour représenter ses propriétés (Figure 7). Cependant, cette modélisation n'est pas satisfaisante car les ensembles sont disjoints, c'est-à-dire qu'il existe des valeurs n'appartenant à aucune propriété. En fait, il utilise le formalisme flou uniquement pour représenter ses propriétés sans exploiter les caractéristiques de l'imprécis.

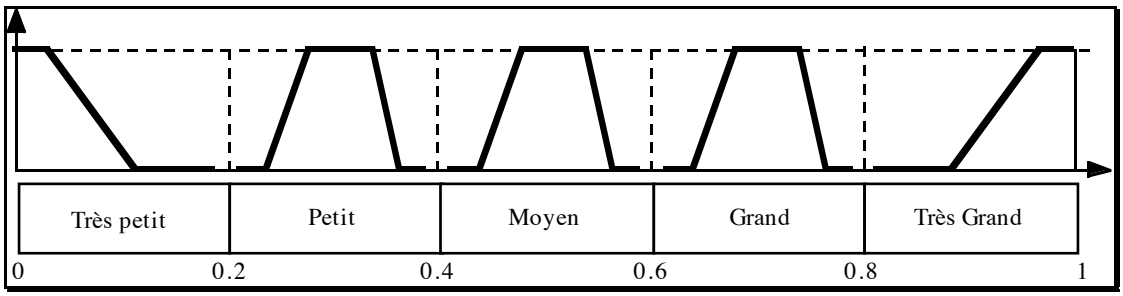

Figure 7. Représentation des qualificatifs d'une propriété sous forme d'intervalle flou ([Dje91])

[RRL94] ont aussi utilisé la logique floue pour modéliser les surfaces de type Bézier ou B-Splines. Ils utilisent des règles floues pour construire des surfaces à l'aide de termes linguistiques. Par contre, aucun formalisme cohérent n'est présenté. Ils proposent plus un modèle pour la génération que pour la représentation et la manipulation des connaissances (propriétés…). Ils proposent une solution très classique en logique floue pour la représentation des termes linguistiques associés aux propriétés (Figure 8). Cependant, ils n'exploitent pas les caractéristiques linguistiques qui font que, par exemple, les ensembles flous ne sont pas toujours identiques.

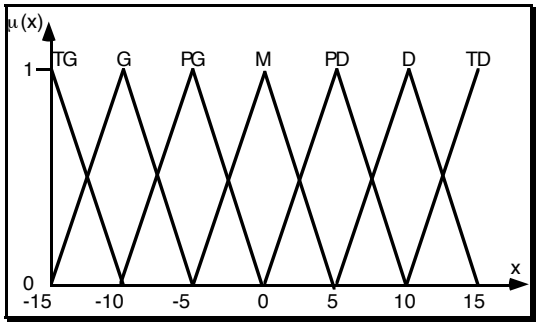

Figure 8. Définition des termes linguistiques selon [RRL94]

[DAG86] et [GAT96] proposent aussi d'utiliser les intervalles flous pour placer des objets dans l'espace et par rapport à d'autres. L'intervalle est construit spécifiquement en fonction de la relation spatiale fournie. Les fonctions d'appartenance des ensembles flous (appelées « fonctions de compatibilité ») sont utilisées pour déterminer des probabilités de présence d'un objet sur un axe de l'espace. Le centre de gravité d'un objet est placé dans l'intersection des fonctions de compatibilité qui le concernent. Aucun modificateur n'est proposé.

Globalement, tous ces travaux déterminent tous les ensembles de valeurs possibles a priori. Les sous-ensembles déterminés par les modificateurs, quand ces derniers existent, sont donnés dès le départ et pour toutes les propriétés possibles. Du point de vue d'un concepteur d'application, cela demande un travail important. [Chau94b] est la seule étude de modificateurs génériques effectuée en modélisation déclarative.

2.4. Modificateurs génériques

L'idée de *modificateurs génériques*, indépendants des propriétés, apparaît dans [Chau92], [Chau94a] et [Chau94b] en modélisation spatiale déclarative ainsi que dans [BeF91], [Oft94] et [MBF96] pour la construction de capteurs flous. La propriété avec modificateur est « calculée » à partir de la propriété seule et des caractéristiques du modificateur utilisé. Dans [Chau94b], il est proposé d'utiliser les notions de propriétés croissantes ("+"), décroissantes ("-") ou stationnaires afin de déterminer l'intervalle résultant de la combinaison d'une propriété et d'un modificateur (Figure 9). A chaque modificateur correspond un pourcentage de

modification “p”. Par exemple, nous avons : « Infiniment » = 90% ; « Très » = 50% ; « Assez » = 25%...

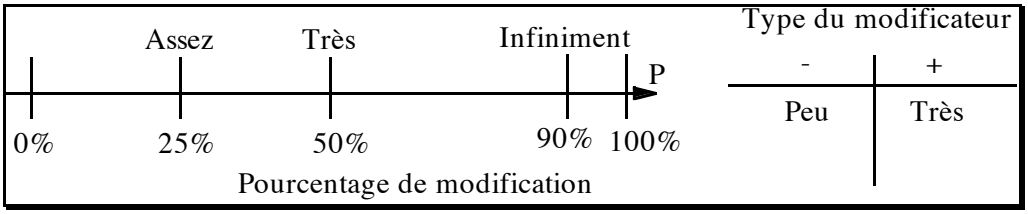

Figure 9. Les modificateurs génériques selon [Chau94b]

Cette méthode utilise aussi les notions de modificateur positif ou négatif. Une propriété est définie sur un intervalle par défaut [O,E]. L’objectif est d’obtenir un nouvel intervalle [O’,E’] prenant en compte le modificateur. L’intervalle par défaut est inclus dans l’intervalle [Min,Max] déterminant toutes les valeurs possibles : $[O,E] \subseteq [Min, Max]$. Trois solutions se présentent (Figure 10) :

1. La propriété est stationnaire, le modificateur n’a pas d’influence (Par exemple « Très moyen » = « Moyen ») ;
2. Le modificateur et la propriété sont de même signe, nous aurons : E’=(1-p)E+pMax et O’=E ;
3. Le modificateur et la propriété sont de signes opposés, nous aurons : O’=(1-p)O+pMin et E’=O.

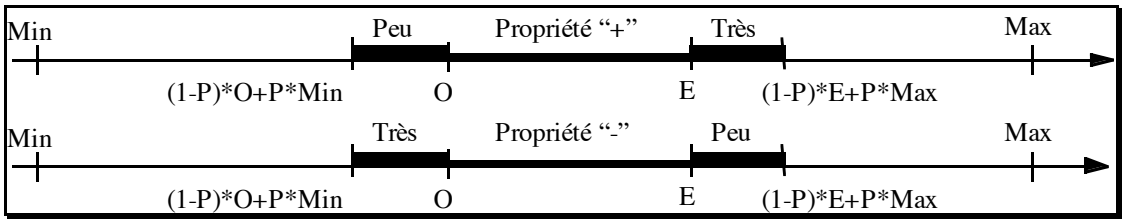

Figure 10. Application des modificateurs génériques selon [Chau94b]

Lorsque plusieurs modificateurs (par exemple les pourcentages p et q) agissent sur une propriété, le pourcentage P résultant sera tel que P = (p + q - p * q). Le sens (le signe) résultant sera celui des modificateurs s’ils sont de même signe ou négatif s’ils sont de signes différents.

2.5. Vers un nouveau formalisme

L’objectif du travail présenté dans cette partie est de proposer un nouveau formalisme. Il se situe dans le prolongement de ce qui a déjà été fait en modélisation déclarative et en particulier dans [Pou94b], [Paj94] et surtout dans [Col90] et [Chau94b]. Il se présente comme une unification et une généralisation de ce qui a déjà été proposé.

En résumé, les propriétés seront représentées par des sous-ensembles flous sur l'ensemble des valeurs possibles d'une mesure. Seules les propriétés concernant les valeurs « normales » sont définies a priori. Les propriétés seront « calculées » à l'aide d'opérateurs génériques, c'est-à-dire indépendants de la propriété sur laquelle ils s'appliquent. Cependant, la méthode d'application que nous choisirons sera différente de celle proposée dans [Chau94b]. Nous utiliserons bien sûr des modificateurs génériques (pour modifier les intervalles associés aux propriétés) mais aussi des opérateurs flous dont l'objectif est de modifier l'imprécision des propriétés. Ces derniers opérateurs sont nouveaux en modélisation déclarative.

Dans la suite, nous introduirons les définitions de base nécessaires à la description des scènes en synthèse d'images.

3. Notions de base

3.1. Univers de formes

Définition I.1.1 : Pour un domaine d'application donné, *l'univers*, noté U_f, est l'ensemble des *scènes* (ou *formes*) S_j, qu'il est possible de construire.

Remarques :

- U_f est un sous-ensemble de U_F (ensemble des scènes possibles pour le domaine d'application donné). U_f contient toutes les scènes que l'algorithme de génération utilisé est capable de construire. Dans la suite, nous supposerons $U_f = U_F$.
- Soit U_D un sous-ensemble de U_F, l'ensemble des scènes vérifiant une description *d*. La phase de calcul d'un modeleur déclaratif permet de déterminer le sous-ensemble U_d de U_D. Dans la suite, nous supposerons $U_d = U_D$.
- Les caractéristiques d'un algorithme de génération en fonction du modèle de représentation utilisé et de U_d, U_f, U_D et U_F ont été présentées dans [Col90] et [Woo91].
- Une scène S_j peut être décomposée en un ensemble *d'éléments* f_{jk} tels que $\cup f_{jk} = S_j$.

Pour décrire un objet d'une scène, il existe un certain nombre de propriétés ayant en commun une même caractéristique. Par exemple, demander « *Le cube est grand* », « *Le cube est de 2m* », « *Le cube est beaucoup plus grand que la pyramide* » ou « *Le cube n'est pas petit* », c'est faire référence à une notion commune aux propriétés « grand », « de 2m » et « petit », à savoir la « taille ». Nous appellerons cela un concept.

3.2. Concept et domaine

Définition I.1.2 : Une scène est caractérisée par plusieurs *concepts* C_i ($i \in J \subset N$). Pour le concept C_i, le *domaine de description* D_i est l'ensemble des valeurs qu'il est susceptible de recevoir.

Remarques :

- En logique floue, un concept est un référentiel associé à une variable linguistique.
- Toute forme peut être décrite par un ensemble de concepts qui ne sont pas forcément indépendants entre eux.

Définition I.1.3 : Le *type d'un domaine* de description est le type des valeurs de ce domaine.

Notons que D_i est généralement borné. Cette contrainte est vérifiée soit naturellement (sémantique du domaine) soit artificiellement (contrainte d'implémentation). De plus, le type des valeurs est a priori quelconque. Nous avons par exemple :

- $\mathbb{B}$, pour les concepts qui n'ont pas de mesure numérique et qui permettent de vérifier une qualité sans pouvoir la quantifier (la mesure se contente de signaler si la propriété est ou n'est pas présente) ;
- $\mathbb{N}$, pour les propriétés qui, par exemple, dénombrent une caractéristique ;
- $\mathbb{R}$, [0,1] très souvent utilisés ;
- ou d'autres types comme une énumération d'objets (chaînes de caractères…).

Nous venons de voir des domaines de dimension 1 mais rien n'interdit d'imaginer des concepts avec un domaine de dimension supérieure (en particulier de dimension 2 et 3). Cependant, en dehors des propriétés de comparaison (que nous verrons au paragraphe 2.1.9 du chapitre I.2 et au chapitre I.6), il nous semble difficile d'envisager simplement de tels concepts. Le problème essentiel n'est pas de trouver une mesure mais surtout de définir aisément des propriétés (précises, de base, intervalle…) et de les manipuler facilement et conformément à l'intuition. Par contre, il est tout à fait envisageable d'utiliser un modeleur déclaratif spécifique permettant de rechercher soit une solution convenable pour l'utilisateur soit de transformer le domaine en dimension 1 pour pouvoir manipuler plus facilement les différents opérateurs. Cette méthode s'applique pour tout concept trop complexe pour être manipulé directement.

Un domaine D_i d'un concept C_i est noté $[Bm, BM]_u$ où $Bm \in \mathbb{R}$ est sa borne minimale (éventuellement égale à $-\infty$), $BM \in \mathbb{R}$ sa borne maximale (éventuellement égale à $+\infty$) et $u \in \mathbb{R}$ l'unité (éventuellement égale à 0, c'est-à-dire un domaine continu). Notons que pour certains concepts dont l'ensemble de valeurs est un ensemble quelconque d'éléments, nous avons l'équivalence suivante : $[1, M]_1 \leftrightarrow \{e_j : 1 \leq j \leq M\}$

3.3. Remarque sur les exemples proposés

Dans le projet FiloFormes ([Paj94]), les *scènes* constituant l'univers sont des *configurations de segments de droite dans un plan*. Ce plan est représenté par une grille n×m. Les seg-

ments sont disposés dans cette grille. Un des objectifs de ce projet est de produire des jeux d'essai raisonnés pour des algorithmes de visualisation en synthèse d'images afin de contrôler les évaluations et mesurer l'impact des différentes améliorations que l'on peut mettre en œuvre [Luc94]. Une scène composée de segments dans un plan peut être caractérisée par le nombre de segments qu'elle comporte, la longueur de la configuration ou des composantes verticales et horizontales, le nombre d'intersections, la densité, le degré moyen de recouvrement… Le domaine de description concernant le nombre de segments de la configuration est alors [0..MaxSegments]. Signalons toutefois, qu'à la différence des concepteurs du projet FiloFormes, pour qui les propriétés étaient précises, nous avons ici plongé l'étude des scènes dans un contexte d'imprécision. Dans la suite de cette partie, le projet LinéaFormes, notre implémentation de FiloFormes, sera utilisé pour illustrer les différentes notions que nous présenterons. Ce projet sera détaillé au chapitre II.6 de la seconde partie.

3.4. Mesure d'un concept

Pour une scène donnée, la valeur qu'elle représente dans un domaine est déterminée à l'aide d'une fonction (quand elle existe) caractéristique du concept. Par exemple, le concept « taille » permettra d'évaluer une scène à l'aide d'une fonction calculant sa hauteur en mètres. Nous appellerons cette fonction une mesure du concept.

Définition I.1.4 : La *mesure d'un concept* C_i sur une scène S_j de U_f est une fonction de S_j^n dans D_i définie comme suit :

$$m_{C_i} : S_j \times S_j \times S_j \times \ldots \times S_j \rightarrow D_i$$
$$f_1, f_2, f_3, \ldots, f_n \rightarrow d_{ij}$$

Un concept avec une telle mesure est dit *concept n-aire*.

Définition I.1.5 : Un *concept unaire* Cu_i est un concept dont la mesure est de S_j dans D_i ($n = 1$), c'est-à-dire si :

$$m_{Cu_i} : S_j \rightarrow D_i$$
$$f \rightarrow d_{ij}$$

Définition I.1.6 : Un *concept binaire* Cb_i est un concept dont la mesure est de $S_j \times S_j$ dans D_i ($n = 2$), c'est-à-dire si :

$$m_{Cb_i} : S_j \times S_j \rightarrow D_i$$
$$f_A, f_B \rightarrow d_{ij}$$

Une mesure sera associée à tout concept. Par exemple, si un concepteur veut définir un concept « beauté » (même si c'est contestable) a priori sans mesure, l'ensemble des valeurs du domaine est : « Vrai » et « Faux ». Ce concept possède alors une formule qui détermine la "valeur" d'une forme selon cette caractéristique.

Exemple : La scène étant constituée de segments dans un plan, pour le domaine de description associé au concept concernant le nombre de segments, nous avons $m_{Nb\ Segments}$:

Plan $\rightarrow$ nb $\in$ [0 .. MaxSegments].

3.5. Propriétés d'un concept

Un concept comporte un certain nombre de propriétés. Dans le cas du concept « taille », on a par exemple « grand », « moyen » et « petit ». Ces propriétés seulement sont définies pour certaines valeurs du domaine.

Définition I.1.7 : A chaque concept C_i est associé un ensemble de propriétés P_{ik}, dites *propriétés de base*, se référant au domaine D_i (Figure 11).

Étant donné que *ces propriétés sont souvent de nature imprécise ou vague*, la théorie des sous-ensembles flous ([Zad65], [DuP93] et Annexe 1) nous permet de les représenter de manière satisfaisante. Dans la suite de ce travail, nous essayerons donc de présenter un formalisme basé sur une représentation à l'aide des ensembles flous.

Définition I.1.8 : A chaque propriété de base P_{ik} est associée une fonction d'appartenance $\mu_{P_{ik}}$ définie sur D_i telle que :

$\mu_{P_{ik}} : D_i = [Bm, BM]_u \rightarrow [0,1]$

$d \rightarrow \mu_{P_{ik}}(d)$

Notation : une propriété sera notée $P_{ik} = \{D_i, \mu_{P_{ik}}\} = \{[Bm, BM]_u, \mu_{P_{ik}}\}$

Exemple : Pour le concept C_i= « Nombre d'intersections » concernant le nombre d'intersections entre les segments du plan, de domaine D_i = [0..MaxIntersections], l'ensemble des propriétés P_{ik} peut être {faible, moyen, important…} (C*f.* Figure 11).

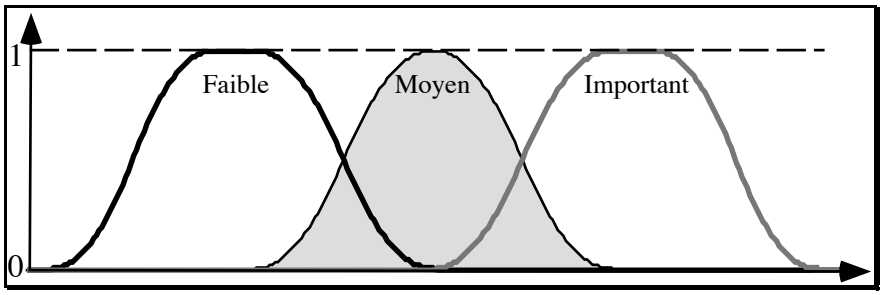

Figure 11. Domaine et propriétés de base associés au concept concernant le nombre d'intersections

A partir de ces concepts et de ces propriétés, nous pouvons construire différents types de propriétés comme les *propriétés élémentaires* (avec un concept unaire), les *propriétés relatives* (avec un concept binaire ou d'arité supérieure) ou des *propriétés de comparaison* (entre

plusieurs objets et selon un ou plusieurs concepts). Les propriétés qu'on peut retrouver en modélisation déclarative sont recensées au chapitre I.2 avec leurs différentes caractéristiques.

3.6. Catégorie linguistique

Dans [DuS95], on trouve une notion intéressante : la *catégorie sémantique* ([Bro43], [Bro48], [Hje35], [Hje37]). Il est possible de définir les catégories sémantiques (pour nous, les concepts) à partir d'une *catégorie maximale* constituée *d'unités sémantiques* (pour nous, les propriétés) prises dans la liste ci-dessous :

a) deux termes, I (*positif*) et F (*négatif*), disjoints présentant deux qualités incompatibles ;
b) un terme *neutre* M qui indique l'absence de l'une et l'autre de ces qualités (la non-application de la catégorie) ;
c) un terme complexe C qui recouvre à la fois I et F et qui indique seulement l'application de la catégorie ;
d) deux termes IM et FM équivalents à C mais qui insistent soit sur I soit sur F.

Toutes les catégories sont construites avec un sous-ensemble des unités sémantiques de la catégorie maximale avec cependant des contraintes de symétrie sur les éléments « signés ». Du point de vue d'une description, ce seront les éléments I, F et M qui seront le plus souvent présents. Ce sont les termes simples de base. Ceci revient à pouvoir construire les catégories suivantes : {I}, {F}, {F, I} et {F, M, I}.

Ceci justifie du point de vue linguistique la présence d'une, deux ou trois propriétés de base dans un concept. Dans ce qui suit, nous utiliserons parfois des concepts à une seule propriété mais le plus souvent ils en auront trois dont une neutre et deux signées (de signes opposés). Nous appellerons les propriétés de base :

- pour trois propriétés : « faible » (négative), « moyen » (neutre) et « important » (positive) (Figure 11) ;
- pour une seule propriété : « vérifiée ».

3.7. Marquage linguistique

Les notions de *marquage* et *d'antonyme* sont présentées dans [DuS95], [Mul91] et [Hor89]. Dans une catégorie sémantique, il arrive qu'un des deux termes soit utilisé comme unité sémantique mais aussi comme représentant de la catégorie entière. Ce terme est dit *non marqué* car il désigne tantôt l'unité tantôt la catégorie. L'autre terme est dit *marqué*. Généralement, le terme marqué tend à indiquer l'absence du terme non marqué. Il est souvent l'unité négative de la catégorie (l'unité Yin). Par contre, le terme non marqué correspond souvent à l'unité positive (le Yang). Il a une valeur « favorable ».

Exemple : Prenons le concept (la catégorie linguistique) de « largeur ». Nous avons alors trois propriétés de base (unités sémantiques) : « étroit », « moyen » et « large ». La propriété positive « large » est non-marquée alors que la propriété négative « étroit » est marquée.

Nous verrons que ces notions sont particulièrement intéressantes pour traiter la négation d'une propriété selon des critères linguistiques. Elles peuvent être aussi utiles dans d'autres cas pour affiner les traitements en prenant en compte des aspects linguistiques comme pour mieux gérer les propriétés de modification…

4. Conclusion

Avec les notions présentées dans ce chapitre, nous allons pouvoir construire un modèle de propriétés cohérent. Dans le prochain chapitre, nous proposons une taxonomie de ces propriétés avant de présenter, aux chapitres I.3 et I.4, les propriétés élémentaires, c'est-à-dire ne faisant référence qu'à un seul objet. Ces propriétés sont construites directement à partir de ces notions de propriétés de base et de concept. Les autres types de propriétés, déjà abordés dans [Des95b] et [Des96], seront traités aux chapitres I.5 et I.6.

CHAPITRE I.2 : CLASSIFICATION DES PROPRIÉTÉS

1. Introduction

La propriété, trait caractéristique de la scène par rapport à un concept donné, est une notion centrale en modélisation déclarative. Notre objectif est d'une part de mieux comprendre le fonctionnement d'une description et d'autre part de proposer des traitements automatiques. Ceci permettra en particulier une interprétation plus fine de la description, une gestion plus efficace de la cohérence et une optimisation performante de la génération des solutions. Pour atteindre cet objectif, nous devons nécessairement comprendre le mécanisme de fonctionnement des propriétés utilisées pour la description. Ce travail demande une étude de toutes les facettes d'une propriété ainsi qu'une formalisation cohérente. Nous allons nous intéresser ici aux propriétés en général. Nous tenterons d'en recenser les différents types selon certains critères. Ce recensement se base sur l'ensemble des propriétés utilisées dans les différents travaux en modélisation déclarative.

Nous présenterons d'abord un classement syntaxique des propriétés (section 2). Nous aborderons ensuite des éléments de classification du point de vue sémantique (section 3). Enfin, nous étudierons les différentes relations qu'il peut y avoir entre propriétés (section 4).

Remarque : [DeM97a] propose aussi un *classement structurel et comportemental* des propriétés. Le premier s'intéresse à étudier les propriétés selon leur structure, c'est-à-dire les différents types d'ensembles flous en fonction de leur définition, de leur forme et de leur construction. Le second s'intéresse à examiner le rôle et le comportement des propriétés selon les différents modes de génération en modélisation déclarative.

2. Classification syntaxique

Pour construire une scène à l'aide d'un modeleur déclaratif, le concepteur doit fournir une description. Celle-ci est formée d'une liste de propriétés. Intéressons nous à leur « forme ». Elles se regroupent en trois grandes catégories : les propriétés permettant de construire la description initiale, celles permettant de déterminer les éléments de la scène et celles modifiant la description en cours de génération.

Les schémas standard que nous proposons permettent d'interpréter cette description au niveau syntaxique et sont utilisés pour *construire automatiquement le modèle de description* nécessaire à la génération des scènes solutions.

2.1. Propriétés descriptives

Définition I.2.1 : Une *propriété descriptive* est une propriété qui permet de *construire une description* déterminant le sous-univers des formes solutions.

Les propriétés descriptives se classent en plusieurs catégories selon leur structure :

- les propriétés élémentaires ;
- les propriétés de concepts n-aires ;
- les propriétés de relations n-aires.

2.1.1 Propriétés élémentaires

Ce sont les propriétés les plus simples. Elles correspondent à des prédicats unaires en logique du premier ordre. Elles portent sur la forme entière, sur une partie seulement ou uniquement sur un élément.

Définition I.2.2 : Soit P_{ik} une propriété de base d'un concept C_i caractérisant une partie X (objet ou élément) d'une scène. Une *propriété élémentaire* est un énoncé du type :

« C_i de X est f_α m_β P_{ik} »,

où f_α est un *opérateur flou* (par exemple « précisément », « réellement », « plus ou moins », « grossièrement », « plutôt », « de l'ordre de », « presque », « normalement » (ou « $Ø_f$ »)…) et m_β est un *opérateur de modification*, ou *modificateur* (« extraordinairement », « très », « extrêmement », « assez peu », « normalement » (ou « Ø »)…).

Lorsqu'il n'y a pas d'ambiguïté sur le concept de la propriété P_{ik}, nous avons la forme « x est f_α m_β P_{ik} ». Par la suite, et pour simplifier la syntaxe, nous garderons cette dernière forme.

Exemples :

- La propriété de base étant « important », le modificateur étant « très », une propriété élémentaire relative à « la longueur de la configuration » (la somme des longueurs de tous les segments de la scène) sur « la scène » peut être « *La longueur de la configuration de la scène est très importante* » ou, avec la simplification si on a l'équivalence entre « important » et « long » pour ce concept, « *La scène est très longue* ».
- « *Le nombre d'intersections [de la scène] est assez important* » où « Le nombre d'intersections » est le concept , « assez » un modificateur et « important » une propriété de base du concept ;
- « *La longueur du segment est vraiment très faible* » où « La longueur du segment » est le concept, « vraiment » un opérateur flou, « très » un modificateur et « faible » une propriété de base du concept.

Les propriétés élémentaires seront étudiées plus en détail au chapitre I.3. Il arrive parfois que ces propriétés, comme les suivantes, soient niées. Nous obtenons alors une négation. Le traitement de cette dernière pour les propriétés élémentaires sera étudié au chapitre I.4. Ce traitement, de type linguistique, est basé sur la détermination d'une propriété élémentaire affirmative implicite à la négation d'une propriété élémentaire.

2.1.2 Propriétés paramétrées

Les propriétés élémentaires, telles que nous venons de les présenter, sont de la forme : « C_i de X est f_α m_β P_{ik} » où P_{ik} est une propriété de base floue ne faisant pas référence de manière explicite à une valeur du domaine. Or, il arrive que dans une description le locuteur donne une propriété de base faisant référence à des valeurs du domaine à la place des propriétés de base définies dans le chapitre précédent.

Définition I.2.3 : Les *propriétés de base paramétrées* sont des propriétés de base définies à partir de valeurs du domaine appelées *valeurs de référence*.

Les propriétés de base paramétrées les plus courantes sont celles construites à partir d'une ou deux valeurs du domaine. Ce sont des énoncés comme :

- « *La longueur du segment est vraiment entre 3 et 4 unités* » où « entre 3 et 4 unités » est une propriété de base paramétrée et « 3 » et « 4 » des valeurs de référence ;
- « La longueur de la configuration est au moins de 10 unités » ;
- « Le degré de recouvrement est très supérieur à 5 » ;
- « Le nombre de segments est d'environ 10 »…

Définition I.2.4 : Les propriétés élémentaires construites à partir de propriétés de base paramétrées sont appelées *propriétés élémentaires paramétrées*.

Nous pouvons distinguer deux grandes classes de propriétés de base paramétrées : les intervalles et les comparaisons élémentaires.

Définition I.2.5 : Les propriétés élémentaires paramétrées de classe *intervalle* ont généralement une des trois formes suivantes :

1. « C_i de X est f_α de V » ;
2. « C_i de X est f_α entre U et V » ;
3. « C_i de X est au moins de V » ou « C_i de X est au plus de V ».

où f_α est un opérateur flou.

Dans ces énoncés, les valeurs de référence U et V sont des valeurs du domaine $D_i=[Bm,BM]_u$ de C_i. On note que les énoncés de la troisième forme n'acceptent aucun opérateur (modificateur ou opérateur flou) alors que les deux autres acceptent seulement des opé-

rateurs flous. Ces propriétés sont construites à partir des propriétés de base paramétrées « Entre U et V » et « De V ».

Pour la suite, posons les hypothèses suivantes :

- « De V » est équivalent à « Entre V et V » ;
- « C_i de X est au moins de V » est équivalent à « C_i de X est Entre V et $+\infty$ » ou « C_i de X est Entre V et BM » ;
- « C_i de X est au plus de V » est équivalent à « C_i de X est Entre $-\infty$ et V » ou « C_i de X est Entre Bm et V ».

Par conséquent, en plus des propriétés de base classiques, tout concept, ayant une relation d'ordre sur ces valeurs, possède un ensemble de propriétés de base paramétrées PI_p tel que :

$$PI_p = \{PI_{pi} : \forall (x,y) \in D_i^2, x \leq y, PI_{pi} = \text{« Entre x et y »}\}.$$

Exemple : « *Le nombre de segments de la scène est entre 10 et 50* », « *La hauteur de l'immeuble est plus ou moins entre 20 et 30 mètres* », « *Cet homme mesure vaguement 1m80* »…

Définition I.2.6 : Les *propriétés de comparaison élémentaires* sont de la forme :

- « C_i de X est f_α m_β supérieur à U » ;
- « C_i de X est f_α m_β inférieur à U ».

Comme pour les intervalles, en plus des propriétés de base classiques, tout concept, ayant une relation d'ordre sur ces valeurs, possède un ensemble de propriétés de base paramétrées PC_p tel que :

$$PC_p = \{PCs_{pi} : \forall x \in D_i, PC_{pi} = \text{« supérieur à x »}\} \cup \{ PCi_{pi} : \forall x \in D_i, PC_{pi} = \text{« inférieur à x »}\}.$$

Exemple : « *La longueur du navire est vraiment très supérieure à 12 mètres* ».

Les propriétés élémentaires paramétrées seront étudiées plus en détail au chapitre I.3. Par ailleurs, les propriétés élémentaires sont souvent utlilisées de la même façon qu'elles soient ou non paramétrées. Par conséquent, dans la suite de ce travail (sauf dans les chapitres I.3 et I.4), « propriété élémentaire » sera équivalent à « propriété élémentaire (paramétrée ou non) ».

2.1.3 Propriétés élémentaires et quantificateurs

Avec certains concepts, la propriété élémentaire ne s'applique plus forcément sur un seul objet mais sur un ensemble d'objets. Il est alors nécessaire de préciser un quantificateur. Comme en logique du second ordre, il est possible de trouver les deux quantificateurs classiques : « il existe » ($\exists$) et « pour tout » ($\forall$). Entre ces deux extrêmes, toute une gradation de quantificateurs précis ou imprécis est possible. Le quantificateur existentiel permet d'énoncer une propriété d'un unique élément (plusieurs si on prend le sens « au moins un »).

L'utilisation du quantificateur universel permet de décrire une propriété que doit respecter chacun des objets de la scène (ou d'une partie de la scène). Par exemple, « *La longueur d'un segment est très importante* » est interprétable comme « *Il existe un segment dont la longueur est très importante* » et « *La pente de tous les segments est assez faible* » comme « *Pour tout segment, la pente est assez faible* ». Entre les deux quantificateurs classiques existe tout un ensemble de quantificateurs indiquant une quantité ou une proportion plus ou moins importante d'objets vérifiant la propriété. Ces quantificateurs peuvent être précis (« un », « 12 »…), imprécis (« entre 10 et 15 ») ou flous (« une dizaine », « quelques », « beaucoup », « au moins »…). « *La hauteur d'une dizaine de maisons est très supérieure à 5 mètres* » et « *La longueur de beaucoup de segments n'est pas très importante* » sont des exemples de propriétés quantifiées utilisant des quantificateurs flous.

Définition I.2.7 : Une *propriété élémentaire quantifiée* est un énoncé de la forme :

« C_i de Q_t X est f_α m_β P_{ik} ».

où C_i est un concept portant sur une caractéristique d'un objet et Q_t est un quantificateur pris parmi les valeurs suivantes : ($\exists$), « un », « au moins un », …, « une dizaine », …, « le tiers », …, « quelques », …, « beaucoup », …, « tous », ($\forall$).

Notons que les propriétés commençant par « Le nombre de X ayant C_i… » constituent une forme de propriétés quantifiées. Par ailleurs, les quantificateurs semblent se diviser en trois catégories :

- les quantificateurs proportionnels qui font référence à une proportion de propriétés vérifiées (« 1/3 », « la moitié », « la plupart », « tous », « aucun »…) ;
- les quantificateurs énumératifs qui donnent une idée du nombre de propriétés vérifiées (« un », « une douzaine », « au moins trois », « au plus 20 »…) ;
- les quantificateurs ambigüs comme « un tout petit peu », « un peu », « quelques », « Ø », « beaucoup », « énormément » qui, suivant le contexte, sont classables dans l'une ou l'autre des catégories précédentes.

Une solution partielle pour traiter les propriétés élémentaires quantifiées, proposée dans [BLP96], sera présentée au chapitre I.5.

2.1.4 Propriétés élémentaires modifiables

Parmi les propriétés élémentaires (paramétrées ou non) d'un concept, certaines n'acceptent que le modificateur vide. En effet, on ne peut pas dire « *Le nombre de segments est très premier* » ou « *Le nombre de segments est très de 10* ». Par contre, on peut dire « *Le nombre de segments est très important* ».

Définition I.2.8 : Les propriétés acceptant d'autres modificateurs que « Ø » sont appelées des *propriétés modifiables*.

Pour comprendre, il faut se rappeler que l'homme ne peut appréhender une situation que de façon simple et globale en fonction d'un seuil de perception ([Jim97]). Chaque terme du langage (propriété de base) est donc défini exclusivement sur un sous-domaine cohérent. Les propriétés associées aux termes linguistiques ne peuvent être définies que sur un bloc à support compact sur le domaine. Tous les phénomènes physiques ne rentrent pas dans ce cadre mais cette explication est valide du point de vue du langage. Il est donc courant que les propriétés basées sur ces phénomènes ne soient pas modifiables.

Nous posons donc le postulat suivant : une propriété modifiable est toujours définie à l'aide d'un intervalle flou (à support compact). Nous pouvons même ajouter que celles représentées par un intervalle classique ne sont pas modifiables.

L'explication est la suivante : les modificateurs ne peuvent s'appliquer qu'à des propriétés utilisant des termes linguistiques, c'est-à-dire des propriétés qualitatives et non quantitatives. Ils permettent de « glisser » l'intervalle compact sur les valeurs du domaine. On ne peut pas les appliquer à des propriétés précises ou même, la plupart du temps, à des propriétés paramétrées. L'intérêt de ces modificateurs est en effet de manipuler une propriété dont on ne connaît pas les valeurs exactes et qui ne peut donc pas être manipulée autrement. Par contre, les opérateurs flous s'appliquent à une propriété quelconque. Sur des propriétés qui ne sont pas modifiables, comme les propriétés paramétrées, ils permettent d'introduire de l'imprécis (par exemple : « *La hauteur des plafonds est approximativement de 3 mètres* »).

2.1.5 Classification structurelle des propriétés élémentaires

Outre l'aspect modifiable de la propriété, sa manipulation varie en fonction de la forme (voir Annexe 1), de la position et de la méthode de construction de la fonction d'appartenance de la propriété élémentaire. Un des éléments constitutifs d'une propriété élémentaire est la propriété de base. La forme de cette propriété est déterminée par la forme intuitive de la propriété élémentaire possédant l'opérateur flou par défaut et le modificateur par défaut (forme de P_{ik} = forme de $Ø_f Ø P_{ik}$). Dans certains cas, il est indispensable de calculer à nouveau les bornes du domaine (dépendant par exemple d'un paramètre de l'univers). Il faut alors se poser la question suivante : quelles seront la forme et la position de la propriété de base après cette modification ?

Définition I.2.9 : Une *propriété à valeur absolue* est définie indépendamment des bornes du domaine, car elle est construite autour d'une valeur caractéristique.

Elles ne sont pas concernées par les modifications des bornes du domaine. Par exemple, si on considère la taille d'un individu, les tailles minimales et maximales sont susceptibles d'être modifiées plus souvent que la taille moyenne qui servira de référence pour la propriété de base « moyen ». Il en est de même pour les propriétés paramétrées sauf dans le cas particulier des comparaisons élémentaires où c'est plutôt la place relative qui importe.

Définition I.2.10 : Une *propriété à valeur relative* est dépendante des bornes.

Elles sont relatives à leur position dans le domaine. Dans ce cas, ce n'est plus une valeur qui importe mais la place et la forme de la propriété sur le domaine et, en particulier, par rapport aux bornes. Il en est ainsi des propriétés comme « important » et « faible » dans la plupart des concepts. Les propriétés de cette catégorie sont souvent des propriétés floues (à l'exception parfois de « moyen ») ou des propriétés paramétrées de la forme comparaison élémentaire.

2.1.6 Propriétés élémentaires globales et propriétés élémentaires locales

Les propriétés portant sur l'ensemble de la scène sont parfois appelées « propriétés globales » et celles portant sur une partie de la scène (sous-ensemble d'une scène structurée ou élément de base) « propriétés locales » ([Col90], [Ple91]…). Cependant, cette classification ne nous semble pas totalement satisfaisante. La séparation entre ces deux ensembles n'est pas nette. Une propriété globale est aussi une propriété locale sur l'objet (unique) qu'est la scène. Notre définition des propriétés élémentaires (éventuellement quantifiées), nous amène à donner une nouvelle définition pour « locale » et « globale ». Selon le concept, la présence des quantificateurs n'est pas toujours indispensable. Nous proposons donc de baser nos définitions sur cette utilisation des quantificateurs.

Définition I.2.11 : Un *concept local* est un concept dont les propriétés utilisent de façon systématique un quantificateur. Ce dernier est défini par le locuteur et peut être précis, imprécis ou flou. Une *propriété élémentaire locale* est une propriété d'un concept local.

Souvent, un concept local est construit à partir :
- d'un paramètre d'un objet de la scène ;
- d'une relation entre deux (ou plus) objets de la scène ;
- d'un concept spécifique sur un objet ou un sous-ensemble d'objets.

Par exemple, les propriétés de base « longueur » et « pente » pour un segment, « teinte » pour la couleur d'une maison ou « vitesse » pour une voiture sont locales. Le locuteur ne peut utiliser cette propriété qu'avec un quantificateur explicite ou implicite si l'objet sur lequel elle porte est nommé.

Définition I.2.12 : Par opposition, un *concept global* est un concept dont les propriétés n'acceptent pas de quantificateurs. Une *propriété élémentaire globale* est associée à un concept global.

Un concept global fait généralement référence à un objet unique de la scène ou à la scène elle-même. Il peut être :

- une statistique ou une quantification sur une propriété locale ;
- le dénombrement d'objets ;
- un concept spécifique.

Par exemple, les propriétés « longueur de la configuration » et « degré de recouvrement » pour une scène composée de segments, « type de quartier » pour la disposition de maison, « la hauteur du toit » pour un modeleur de maison ou « teinte » pour un modeleur de couleurs (cf. ChromoFormes dans la seconde partie) sont globales. Mais alors, comment classer une propriété telle que : « *La longueur des segments est importante* » ? Comme elle sous-entend un quantificateur universel, elle est classée comme une propriété locale.

Exemple : Si on considère un logement composé d'une maison, d'un garage et d'un jardin, nous pouvons donner la propriété globale suivante : « *La hauteur de la maison est assez importante* ». Par contre, si on considère un lotissement composé d'un certain nombre de maisons, nous pouvons donner la propriété locale suivante : « *La hauteur d'une maison est assez importante* » étant sous-entendu « *Il existe une maison dont la hauteur est assez importante* ».

Il existe un cas particulier de propriété globale : les statistiques sur une propriété locale (« *La surface moyenne d'une propriété est très importante* »).

2.1.7 Propriétés anonymes et propriétés nommées

Dans certaines descriptions, le concepteur veut pouvoir nommer des objets afin de pouvoir leur attribuer plusieurs propriétés. En première approche, il semblerait se dégager trois sous-catégories de propriétés locales : les propriétés anonymes (« *La hauteur d'une maison est assez importante* »), les propriétés pseudo-nommées (« *La maison la plus haute est rouge* » ou « *La troisième maison [décrite] est longue* ») et les propriétés nommées (« *La hauteur de la maison M est assez importante* »). Cependant une propriété anonyme ou pseudo-nommée peut être considérée comme une propriété dont l'objet est nommé par défaut (« *La hauteur [d'une maison] de la maison M102 est assez importante* »).

2.1.8 Propriétés de concepts n-aires

Dans certains cas d'application, il existe des domaines dont les mesures ne se font pas par rapport à un élément mais par rapport à plusieurs éléments. Ces concepts représentent des relations entre des objets. Ce sont des concepts n-aires (relation avec n objets).

Définition I.2.13 : Une *propriété relative* est une propriété élémentaire associée à un concept n-aire.

En dehors de la référence aux objets, cette forme de propriété est de même nature que celle des propriétés élémentaires. D'abord, intéressons nous plus précisément aux propriétés relatives de concepts binaires. Les concepts permettant de tenir compte de relations (d'interactions) entre deux objets (concepts binaires) sont les plus courants et les plus faciles à manipuler. Par exemple, si on considère des relations spatiales, nous aurons des relations de placement relatif (« à gauche de », « devant », « sur »…) et de distance (« loin de », « proche de »…).

Définition I.2.14 : Une *propriété relative binaire* (souvent appelée par abus de langage « propriété relative ») est un énoncé de la forme :

« C_i entre X_p et X_q est f_α m_β P_{ik} ».

Exemples : « *A est très loin de B* » (l'objet "A" est en relation avec l'objet "B" par la propriété « très loin de »), « *La distance entre la maison M1 et la maison M2 est très faible* », « *La maison M1 est à gauche de la maison M2* », « *A est très proche de B* », « *Les segments S1 et S3 se coupent* »…

Ces propriétés sont associées à un concept binaire (définition I.1.6) comme par exemple, le concept des distances entre deux objets. Cependant, elles sont traitées comme des propriétés élémentaires à concept unaire (définition I.1.5). Le domaine et les propriétés de base sont identiques à ceux des concepts unaires. Les propriétés relatives ont donc les mêmes caractéristiques que les propriétés élémentaires et sont gérées de la même manière (chapitre I.3), en particulier par rapport à l'application des opérateurs ([Des95b]).

Remarque : Dans [GAT96], les propriétés relatives (concernant uniquement des relations spatiales comme dans [Chau94]) sont représentées par des faits logiques en Prolog et sont organisées en graphe. Leur généralisation à des relations quelconques semble assez difficile. De plus, aucune nuance, aucun modificateur n'est proposé. Le système semble assez figé.

A partir de ces propriétés de concepts binaires, on peut construire des propriétés quantifiées faisant intervenir plus de deux objets.

Définition I.2.15 : Une *propriété relative binaire quantifiée* est un des deux énoncés suivants :

1. « C_i entre X_p et Q_t X est f_α m_β P_{ik} » ou « C_i entre Q_t X et X_p est f_α m_β P_{ik} » ;
2. « C_i entre Q_t (X et Y) est f_α m_β P_{ik} ».

Exemples : « *La distance entre la maison M1 et les autres maisons est très importante* » (cas 1), « *La distance entre chaque maison est assez faible* » (cas 2).

Par extension de la notion de concept binaire, on peut imaginer d'avoir des concepts n-aires (n>2). Cependant, à notre connaissance, ce type de concept n'existe pas ou peu car probablement trop complexe à manipuler par l'être humain. Ceux qui existent sont souvent équivalents à une combinaison de concepts unaires ou binaires. Mais, ce n'est évidemment pas toujours le cas. Certaines propriétés ne sont pas faciles à définir autrement (ou très difficilement). Nous aurons par exemple une propriété comme « *Quatre maisons forment un carré* » (ou « *La forme géométrique des maisons M1, M2, M3 et M4 est un carré* ») ou « *Le cube C3 est placé entre le cube C1 et le cube C2* ».

2.1.9 Propriétés de relations n-aires

Ces propriétés mettent en jeu soit au moins deux objets de la scène par rapport à une propriété soit deux propriétés sur un ou deux objets (ou sur la scène). Les éléments des domaines sont nécessairement comparables au niveau sémantique. Les propriétés utilisées sont des propriétés élémentaires ou relatives.

Une propriété de relation binaire permet de manipuler un ou deux objets de la scène suivant une ou deux propriétés de base pas forcément du même concept. Les concepts manipulés sont unaires.

Définition I.2.16 : Une *propriété de comparaison* est un énoncé de la forme :

« C_i de X_p est f_α k_β s_δ P_{ik} que C_j de X_q ».

avec i et j quelconques et les valeurs de D_i et D_j comparables au niveau arithmétique et sémantique. k_β est un *opérateur de comparaison* qui est flou (« un tout petit peu », « un peu », « Ø », « beaucoup », « extrêmement ») ou précis (« 2 fois », « de V unités », « Entre U et V unités », « d'au moins V unités », « d'au plus V unités »). s_δ, *l'opérateur de direction*, indique la direction de comparaison (« moins », « aussi » et « plus ») par rapport à la direction de la *propriété de référence* P_{ik}. Le traitement de cette propriété sera détaillé dans le chapitre I.6.

Par exemple :

- « *La hauteur de la maison M1 est beaucoup plus importante que la hauteur de la maison M2* » ;
- « *La largeur de la maison M1 est plus importante que sa hauteur* », « *La longueur de S1 est beaucoup moins importante que celle de S2* » ;
- « *La taille de la personne A est plus importante que celle de la personne B* ».

Définition I.2.17 : Une *propriété de comparaison homogène* est une propriété de comparaison où $C_i = C_j$ et $X_p \neq X_q$. Une *propriété de comparaison hétérogène* est une propriété de comparaison où $C_i \neq C_j$.

Par extension, nous pouvons construire des propriétés mettant en relation un objet avec plusieurs. Ces propriétés sont construites comme une relation binaire à l'exception d'un des deux objets qui devient un groupe d'objets. Nous pouvons alors avoir des énoncés selon les deux formes suivantes :

1. « C_i de X_p est f_α k_β s_δ P_{ik} que C_j de Q_t X » ;
2. « C_i de Q_t X est f_α k_β s_δ P_{ik} que C_j de X_p ».

Exemples : « *La maison M1 est beaucoup plus grande que les autres* » (cas 1), « *Une dizaine de maisons ont une largeur plus importante que celle de la maison M1* » (cas 2).

Définition I.2.18 : Lorsque la propriété de référence n'est plus associée à un concept unaire mais à un concept binaire, nous avons des *propriétés de comparaison relatives*. Ce sont généralement des comparaisons homogènes. Nous avons des énoncés de la forme :

« C_i entre X_p et X_q est f_α k_β s_δ P_{ik} que C_i entre X_r et X_s ».

Par exemple : « *La maison M1 est beaucoup plus loin de la maison M2 que la maison M3* » (avec X_q=X_r), « *La maison M1 est beaucoup plus loin de la maison M2 que de la maison M3* » (avec X_p=X_r) ou « *La maison M1 est beaucoup plus loin de la maison M2 que la maison M3 de la maison M4* ».

Nous pouvons étendre ces propriétés de comparaison relatives à l'aide de quantificateurs de la même manière qu'avec les propriétés de comparaison simples.

2.2. Propriétés constructives

Lorsque l'utilisateur d'un modeleur déclaratif désire décrire une scène, il veut pouvoir décrire ses composants. Pour cela, il utilise un certain nombre de propriétés spécifiques indiquant de quoi est composée la scène. Ces composants sont les éléments de base de la scène mais aussi des objets « de niveau supérieur » construits à partir d'autres objets.

Définition I.2.19 : Une *propriété constructive* est une propriété décrivant les objets présents dans la scène et leurs relations.

Nous avons des descriptions comme « *La scène est composée de cercles et de rectangles* », « *Les cercles forment une zone A* », « *Les rectangles forment une zone B* » et « *Les zones A et B forment une zone C* »… Ces propriétés constructives se classent en deux catégories :

- les propriétés déterminant des composants de base de la scène selon le schéma « La scène est composée de X, Y… » appelées *propriétés de construction descendante* ;
- les propriétés déterminant les objets de la scène selon le schéma « Les objets X, Y… forment un objet Z » appelées *propriétés de construction ascendante*.

Ces propriétés ne manipulent pas de concepts élémentaires mais seulement des éléments (concepts complexes) qui seront caractérisés par des propriétés descriptives. Nous aurons, par exemple, dans la suite de la description ci-dessus la propriété « *La zone C est très large* ».

Cette construction hiérarchique de la scène est soit donnée en bloc au début de la génération soit construite en ajoutant au fur et à mesure les éléments, seulement lorsque ceux de niveau supérieur sont convenablement construits. Cette dernière méthode correspond à une construction par ébauches successives ([CDMM97c]). Pour revenir à notre première description, cette méthode construit d'abord une zone schématique C possédant les propriétés voulues. Ensuite, les zones A et B sont construites dans la zone C selon leurs propres caractéristiques. Enfin, les cercles et les rectangles sont placés dans leurs zones respectives en fonction des contraintes qui leur sont appliquées. Avec une telle technique, chaque ébauche d'un niveau est une description pour le niveau suivant.

Entre les propriétés constructives et les propriétés descriptives, il existe un cas particulier de propriétés descriptives indiquant la quantité d'objets d'un type donné. Nous avons alors un énoncé de la forme : « Le nombre d'objets X est f_α m_β P_{ik} ». Par exemple, nous aurons « *Le nombre de rectangles est assez important* » ou « *Le nombre de cercles est très faible* ».

2.3. Propriétés modificatrices

Lorsque le concepteur prend connaissance d'une solution à sa description, il peut lui arriver de vouloir l'ajuster en modifiant une propriété. Ces propriétés sont donc relatives à une forme solution donnée.

2.3.1 Propriétés modificatrices

Définition I.2.20 : Une *propriété modificatrice* est un énoncé de la forme :

« C_i [de X] est f_α k_β s_δ P_{ik} ».

avec les mêmes opérateurs que ceux définis pour les propriétés de comparaison. L'action de ces propriétés dépend de l'opérateur s_δ. Pour ces opérateurs tels que « plus » et « moins », la solution attendue sera différente de celle proposée. Par contre, avec l'opérateur « aussi », l'utilisateur fige la valeur du concept à la valeur actuelle.

Exemples : « *La longueur de la rue est beaucoup plus importante* », « *La hauteur de la maison M est environ de 2m plus importante* », « *La longueur de la rue est entre 2 et 3 Km plus faible* » ou « *La maison M1 est aussi grande* » (ce qui signifie « *La taille attendue de la maison M1 est la même que la taille actuelle* » et donc implicitement « *ne plus bouger la taille de la maison !* »).

Ce type de propriété est à rapprocher des propriétés élémentaires (ou relatives) paramétrées en considérant le paramètre comme étant la valeur de la scène courante. Ainsi, « *La longueur de la rue est beaucoup plus importante* » est équivalent à « *La longueur attendue de la rue est très supérieure à la longueur actuelle de la rue* ». Elles portent principalement sur des concepts unaires ou binaires. Bien sûr, elles sont aussi quantifiables.

2.3.2 Propriétés modificatrices quantifiées et relatives

Définition I.2.21 : Une *propriété modificatrice quantifiée* est un énoncé de la forme :
« C_i de Q_t X est f_α k_β s_δ P_{ik} ».

Sauf avec les deux quantificateurs classiques ($\exists$ et $\forall$), il ne semble pas que ce genre de propriété soit souvent utilisé. Exemples : « *La longueur de tous les segments est plus importante* », « *La pente du segment S est beaucoup moins faible* ».

Définition I.2.22 : Une *propriété modificatrice relative* est un énoncé de la forme :
« C_i entre X_p et X_q est f_α k_β s_δ P_{ik} ».

Exemple : « *La distance entre les maisons A et B est beaucoup plus importante* ».

Définition I.2.23 : Une *propriété modificatrice relative quantifiée* est un énoncé de la forme :

1. « C_i entre X_p et Q_t X est f_α k_β s_δ P_{ik} » ou « C_i entre Q_t X et X_p est f_α k_β s_δ P_{ik} » ;
2. « C_i entre Q_t (X et Y) est f_α k_β s_δ P_{ik} ».

Exemple : « *La distance entre chaque maison est beaucoup plus importante* ».

3. Classification sémantique

Lorsqu'on parcourt les différents travaux effectués en modélisation déclarative, très vite on se demande s'il ne serait pas possible de créer une bibliothèque de concepts voire de pouvoir en construire certains automatiquement.

3.1. Les concepts « courants »

Comme le domaine privilégié de la modélisation déclarative en synthèse d'images est la modélisation géométrique, certains concepts ou classes de concepts se retrouvent assez souvent dans un modeleur. Plus généralement, il serait intéressant de disposer de tous les concepts faisant partie de *l'ontologie* (chapitre II.1, §2.2.5 ; [CBB96], [Gru93a], [Gru93b], [GuG95]) en modélisation déclarative. Nous distinguerons deux catégories de concepts :

- ceux liés à un objet ;
- ceux liés à un ensemble d'objets.

Les concepts permettant de décrire un objet sont des concepts liés essentiellement à sa géométrie (forme géométrique, topologie, dimensions, orientation…), à son placement dans la scène (souvent mesuré par rapport à la boîte englobante qui lui est associée) et à ses caractéristiques de rendu (couleur, texture, luminosité…). Dans certains domaines d'application, la scène comporte plusieurs objets éventuellement de types différents. Il existe un ensemble de concepts liés aux relations entre ces objets (propriétés inter-objets) avec, en particulier, les concepts de positionnement relatif, de répartition et de dénombrement (nombre d'objets de chaque type).

Remarque : L'ontologie est une notion à laquelle nous attachons beaucoup d'importance. Elle occupe une place importante dans notre plate-forme (chapitre II.1 ; [DeT97]).

3.2. Les concepts construits de manière automatique

Nous venons de montrer que certains concepts se retrouvent très souvent dans les modeleurs. En fait, aux différents objets ainsi qu'à la scène, on peut associer des ensembles de paramètres. Il est alors envisageable de construire des concepts se basant sur ces paramètres. Tel ou tel paramètre ne donnant pas de concept « représentatif » peut éventuellement être éliminé. Par exemple, si on désire décrire des segments dans un plan (projet LinéaFormes), la scène sera paramétrée par les dimensions de la grille sur laquelle seront « posés » les segments. Un segment (objet de la scène) se paramètre par son origine, sa pente et sa longueur. Ainsi, nous pouvons construire automatiquement, dès la définition de ces paramètres, des concepts comme « *La taille verticale de la scène* », « *La pente d'un segment* »…

Généralement, nous rencontrons deux types de concepts dans une application déclarative :

- les concepts liés à une caractéristique d'un objet (de base ou complexe) qui servent éventuellement à générer l'objet ;
- les concepts liés à des relations entre les objets qui se contentent de vérifier la scène construite.

4. Relations entre propriétés

Dans une description, les propriétés énoncées ne sont pas forcément indépendantes les unes des autres. Nous allons maintenant examiner les différentes relations possibles entre elles. Cela nous permettra d'améliorer la construction du modèle de description et la gestion de la cohérence.

4.1. Niveau d'un concept

Un concept peut être mis en relation avec un autre concept aussi bien au niveau syntaxique que sémantique.

Nous aurons trois *niveaux de concepts* :

- Niveau 1 : Ce sont des concepts dont la mesure prend son argument dans la scène (objet, ensemble d'objets ou scène entière). Nous avons par exemple toutes les propriétés élémentaires (non quantifiées) sur des propriétés de base, les propriétés relatives (non quantifiées)…
- Niveau 2 : Il concerne les concepts prenant comme paramètres les mesures d'autres concepts. Par exemple, nous avons les propriétés de comparaison, les propriétés concernant des statistiques sur une propriété locale, celles de dénombrement de propriétés locales…
- Niveau 3 : Il s'agit ici des concepts construits à partir de la composition de propriétés (ils prennent comme paramètres les degrés d'appartenance d'autres propriétés) comme les propriétés quantifiées, les disjonctions ou les conjonctions de propriétés…

Ces niveaux de concepts permettent de construire des propriétés complexes à partir de plusieurs concepts ou d'autres propriétés.

4.2. Dépendance d'instanciation

Dans la description, des propriétés peuvent être liées par la référence à un objet explicitement désigné par un nom. Prenons la description suivante : « *La hauteur de la maison M1 est assez importante* (P1). *La surface de la maison M2 est plus importante que celle de la maison M1* (P2). *La couleur de la maison M2 est blanche* (P3) ». Les propriétés P1 et P2 sont liées par la référence à l'objet M1. Ce lien peut être étendu à la propriété P3 car les objets M1 et M2 sont explicitement en relation grâce à la propriété P2.

Définition I.2.24 : Deux propriétés sont liées par une *dépendance d'instanciation* si elles font explicitement référence à un même objet. Ce lien peut être direct ou, éventuellement, indirect.

Sur la Figure 12, les propriétés P1, P2 et P3 sont liées (en dépendance d'instanciation) car elles font référence toutes les trois à « Objet1 » (idem pour les propriétés P2, P4 et P5 avec « Objet3 »).

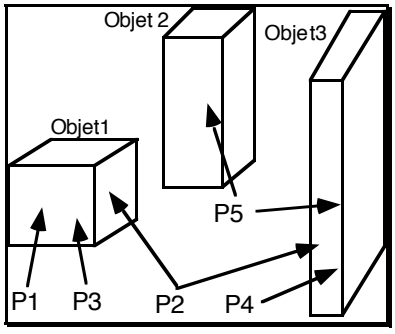

Figure 12. Dépendances entre propriétés

Par contre, les trois propriétés « *La hauteur d'une maison est assez importante* (P1). *La surface de la maison M2 est plus importante que celle de la maison M3* (P2). *La couleur de la maison M1 est blanche* (P3) » ne sont pas liées par une dépendance d'instanciation (même s'il peut arriver que, lors d'une évaluation, la maison de P1 soit M1, M2 ou M3).

4.3. Dépendance sémantique

Lors d'une description, il peut arriver que plusieurs propriétés fassent référence au même concept. Pour s'assurer de la cohérence, il faut s'intéresser aux liens sémantiques entre les propriétés. Prenons par exemple la description suivante : « *La hauteur de la maison M1 est assez importante. La hauteur de toutes les maisons n'est pas extrêmement importante…* ». Ces deux propriétés sont liées sur la maison M1 qui doit pouvoir vérifier au moins un peu des deux. Tout dépendra de l'interprétation qu'on fera de la négation (chapitre I.4).

Définition I.2.25 : Deux propriétés sont liées par une *dépendance sémantique* si elles font référence à un même concept.

Ce type de lien entre propriétés peut être direct comme nous venons de le voir mais aussi plus ou moins indirect par l'intermédiaire d'autres propriétés. Sur la Figure 12, les propriétés P1, P2 et P4 sont liées (dépendance sémantique) si elles font référence toutes les trois au concept « Hauteur » par exemple.

4.4. Dépendance de construction

Généralement, la description n'est pas fournie complètement en une seule fois. Les propriétés sont énoncées les unes après les autres. Le traitement d'une propriété donnée ne sera donc pas le même suivant sa place dans la description. Par exemple, si on donne la propriété « *Les maisons ne sont pas très larges* », l'interprétation linguistique de cette négation (§2.1.1 et chapitre I.4) sera différente selon qu'on la place avant ou après la propriété « *La largeur des maisons est moyenne* ». En effet, si elle est placée avant, les propriétés candidates à la négation seront uniquement celles ayant une intersection avec elle. Par contre, si elle est placée après, toutes les propriétés suffisamment différentes seront candidates.

Certaines méthodes de construction traitent une description de manière incrémentale. Une solution est alors produite après chaque ajout de propriété. Ainsi, la propriété « *La prochaine maison est à droite de celles déjà faites* » dépend de manière évidente de la description précédente.

4.5. Propriétés de composition

Parfois, une propriété ne peut pas s'exprimer à l'aide d'une propriété élémentaire. Il est alors nécessaire de combiner des propriétés (élémentaires, relatives, de comparaison…) à l'aide d'opérateurs de composition.

Définition I.2.26 : Les propriétés définies à l'aide d'une composition de propriétés s'appellent des *propriétés de composition*. Les énoncés permettant de définir une propriété de composition sont de la forme :

« P signifie : "P1" OP "P2" OP "P3"… ».

où OP est un opérateur de composition.

Nous distinguerons deux classes d'opérateurs de composition :

- les *opérateurs conjonctifs* ("Et", ".") pour obtenir une vérification de chacune des propriétés de la composition ;
- les *opérateurs disjonctifs* ("Ou", "Ou Bien") pour construire des propriétés alternatives.

Cette propriété de composition est utilisée dans une description comme une propriété élémentaire ou relative. Elle permet de construire des énoncés comme « X est f_α m_β P » (il n'y a pas de concept explicitement utilisé avec de telles propriétés). Il est en effet tout à fait possible de lui appliquer un modificateur et un opérateur flou. Cependant, l'application de tels opérateurs ne sera pas identique à celle effectuée sur une propriété de base d'un concept et dépendra de l'opérateur utilisé.

4.5.1 L'opérateur conjonctif "."

Une composition de propriétés, dont l'opérateur est ".", est une conjonction de propriétés. Celles-ci sont considérées comme totalement indépendantes les unes des autres. Cet opérateur permet de construire une énumération de propriétés demandées simultanément, c'est-à-dire qu'elle n'est vérifiée que si toutes ses propriétés le sont. La description « *La maison M1 est très haute. La couleur des maisons est beige. La hauteur des maisons n'est pas extrêmement importante…* » comporte un certain nombre de propriétés indépendantes les unes des autres en particulier par rapport à d'éventuels modificateurs. Ces propriétés sont éventuellement liées implicitement par une dépendance sémantique. L'application d'un opérateur à une conjonction P sera alors équivalente à l'application de l'opérateur sur chacune des propriétés composant P (distributivité des opérateurs). Si on considère que « Trapu signifie : *La hauteur est faible. La largeur est importante.* » alors « Très trapu » sera équivalent à « *La hauteur est très faible. La largeur est très importante.* ».

4.5.2 L'opérateur conjonctif "Et"

Une composition de propriétés, dont l'opérateur est "Et", est aussi une conjonction de propriétés. Cependant, son utilisation fait intervenir une ou plusieurs relations implicites entre les opérandes de cette conjonction. Ces relations ne sont possibles que si les domaines de ces opérandes sont comparables au niveau sémantique. Dans le cas contraire, "Et" est totalement équivalent à ".". Les relations qui lient les différentes propriétés sont implicites, ce qui signifie qu'il est assez difficile de les déterminer automatiquement. Comme pour la négation (§2.1.1 et chapitre I.4), il faut faire préciser par le locuteur les éventuelles propriétés implicites de la description. Ainsi, « Trapu signifie : *La hauteur est faible Et la largeur est importante*. » peut être équivalent à « Trapu signifie : *La hauteur est faible*. *La largeur est importante*. *La hauteur est beaucoup plus faible que la largeur*. ». Par conséquent, « Très Trapu » sera équivalent à « *La hauteur est très faible*. *La largeur est très importante*. *La hauteur est beaucoup beaucoup plus faible que la largeur*. » (« très beaucoup » étant considéré comme équivalent à « beaucoup beaucoup »).

4.5.3 L'opérateur disjonctif "Ou"

Une composition de propriétés, dont l'opérateur est "Ou", est une disjonction de propriétés. Ces dernières sont considérées comme totalement indépendantes les unes des autres. De toute manière, il est difficile d'imaginer une relation, même implicite, entre deux propriétés qui ne sont pas forcément vérifiées en même temps. Cet opérateur permet de construire une énumération de propriétés alternatives. Elles peuvent être éventuellement vérifiées simultanément, c'est-à-dire que la composition est vérifiée si au moins une de ses propriétés l'est. Du fait de l'indépendance totale des propriétés les unes par rapport aux autres, l'application d'un opérateur sur une propriété de composition avec cet opérateur sera identique à celle effectuée pour l'opérateur ".".

4.5.4 L'opérateur disjonctif "Ou Bien"

Une composition de propriétés, dont l'opérateur est "Ou Bien", est une disjonction de propriétés. Son utilisation et son comportement sont identiques à ceux de l'opérateur "Ou" à ceci près qu'il faut nécessairement qu'une seule propriété soit vérifiée.

5. Conclusion

Nous avons mis en évidence quelques méthodes de classement des propriétés composant une description. En particulier, deux axes sont ressortis :

- le structurel qui permet de comprendre une description et de construire un modèle de description interne cohérent et fidèle pour la phase de génération ;
- le sémantique pour construire et manipuler automatiquement certains concepts courants.

Nous avons également mis en évidence les relations qu'il peut y avoir entre les propriétés d'une description.

Ce travail ouvre la porte à un vaste champ d'investigations, en particulier sur l'axe comportemental [DeM97a]. Plusieurs travaux ont été effectués sur ce sujet. Cependant, ils ne portent que sur certains points précis et ne sont pas développés dans le cadre général de la modélisation déclarative. De même, il conviendrait d'affiner la formalisation de l'ensemble des classes syntaxiques proposées afin de travailler dans un cadre formel unique et cohérent. Il faudrait aussi recenser beaucoup plus précisément les concepts formant l'ontologie de la modélisation déclarative.

Détaillons maintenant de manière plus précise le traitement des propriétés élémentaires et des propriétés élémentaires paramétrées (chapitre I.3). Nous verrons ensuite une proposition pour le traitement de la négation de ces propriétés élémentaires à l'aide de notions de linguistique (chapitre I.4). Puis, nous proposerons des idées de traitement pour les propriétés élémentaires quantifiées, les propriétés modificatrices et les relations de comparaison (chapitres I.5 et I.6).

CHAPITRE I.3 : LES PROPRIÉTÉS ÉLÉMENTAIRES

1. Introduction

On peut maintenant préciser le cadre formel de représentation des propriétés élémentaires (définition I.2.2), forme de description la plus simple.

Nous présenterons d'abord les propriétés élémentaires (non paramétrées) de la forme « C_i de X est m_α P_{ik} » (sections 2 à 4). Nous étudierons les modificateurs m_α choisis (section 2) et les caractéristiques d'une propriété élémentaire (section 3). Nous verrons ensuite comment traiter l'application d'un modificateur (section 4) ou éventuellement de plusieurs modificateurs (section 5) sur une propriété de base. Puis nous étudierons la forme finale des propriétés élémentaires « C_i de X est f_α m_β P_{ik} », c'est-à-dire celle comportant un opérateur flou et un modificateur (section 6). Nous examinerons ensuite des méthodes pour construire automatiquement les propriétés de base en fonction de critères standard (section 7). Enfin, nous étudierons le cas des propriétés élémentaires paramétrées (section 8).

2. Les modificateurs retenus

Dans la propriété élémentaire « C_i de X est m_α P_{ik} », le modificateur ou opérateur de modification, porte sur la sémantique de la propriété de base. En effet, dire « *Le cube est assez grand* » et « *Le cube est très grand* » change l'idée de taille qu'on a du cube. La seconde assertion, fait référence à une plage de tailles plus importantes que la première.

Les modificateurs possibles étant très nombreux, nous avons restreint notre choix à un ensemble de P modificateurs, noté $M_P = \{m_\alpha \mid \alpha \in [1..P]\}$, pour lesquels existe la relation d'ordre total suivante : $m_\alpha \leq m_\beta \Leftrightarrow \alpha \leq \beta$. M_P comporte un modificateur particulier noté « normalement » ou « Ø ». C'est l'opérateur par défaut appelé aussi *modificateur vide*. Ainsi, un énoncé de la forme « x est P_{ik} » est équivalent à l'énoncé d'une propriété élémentaire « x est Ø P_{ik} » ou « x est normalement P_{ik} ». Il est donc implicite dans l'énoncé « x est important ». Nous avons arbitrairement utilisé l'ensemble suivant : M_7={extrêmement peu, très peu, assez peu, normalement, assez, très, extrêmement}.

Remarque : Pour les modificateurs comme pour les autres opérateurs, les termes sont choisis arbitrairement. Le formalisme que nous proposons n'est pas dépendant de ces choix et ne peut que bénéficier d'une éventuelle étude linguistique sur ce sujet. Les modifications ne seraient que mineures et très faciles à effectuer (changement des termes et éventuellement des coefficients). L'important pour ce formalisme est le principe et la forme des fonctions

d'application des opérateurs. Les termes, la valeur des coefficients et des constantes des fonctions présentées sont choisis parce qu'ils sont *intuitivement corrects* et qu'ils *donnent des résultats satisfaisants*.

3. Caractérisation d'une propriété élémentaire

3.1. Représentation des propriétés de base

Dans tout ce qui suit, nous supposerons toutes les propriétés modifiables (définition I.2.8). Une propriété de base est donc représentée par un sous-ensemble flou compact du domaine. Pour faciliter le traitement, la fonction d'appartenance $\mu_{P_{ik}}$ d'une propriété de base modifiable P_{ik} d'un concept C_i définie sur un domaine D_i ($[Bm, BM]_u$) est une fonction L-R (Figure 13), à support compact, définie par $<\alpha, a, b, \beta>$ et deux fonctions L et R ([DuP80a], [Bou93], [Ton95] et Annexe 2) telle que (Définition I.1.8) :

$\mu_{P_{ik}} : D_i \rightarrow [0,1]$

$d \rightarrow \mu_{P_{ik}}(d) = \mu_{<\alpha_{ik}, a_{ik}, b_{ik}, \beta_{ik}>, L_{ik}, R_{ik}}(d)$

P_{ik} est donc définie par $\{[Bm,BM]_u, \mu_{P_{ik}}\}$ ou, dans notre cas, par $\{[Bm,BM]_u, <\alpha_{ik}, a_{ik}, b_{ik}, \beta_{ik}>, L_{ik}, R_{ik}\}$. « m_α P_{ik} » est représentée par un ensemble flou dont la fonction d'appartenance sera celle de la propriété de base modifiée par l'opérateur de modification.

Pour nous, la *sémantique d'une propriété* concerne les valeurs du domaine qu'elle recouvre, la forme de la fonction d'appartenance qui lui est associée et son comportement vis-à-vis des opérateurs susceptibles d'être utilisés.

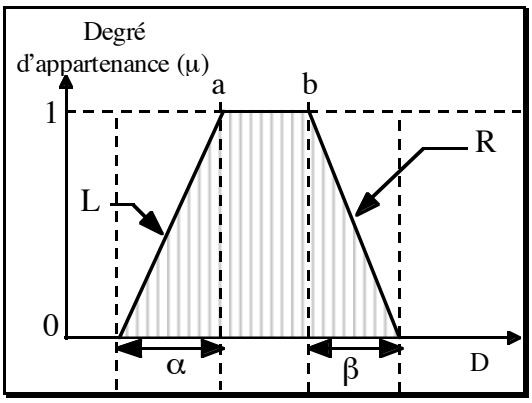

Figure 13. Une fonction d'appartenance < α, a, b, β > LR

Exemple : Étant donné le concept « nombre d'intersections », l'application des modificateurs à la propriété de base « important » donne les sous-ensembles flous rassemblés dans la Figure 14.

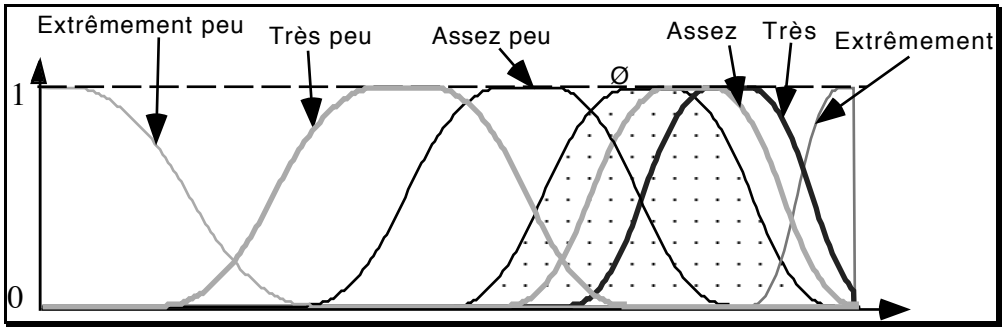

Figure 14. Modificateurs de la propriété de base « important »

Remarque : Dans cette figure, on peut noter un espace entre « très important » (resp. « très peu important ») et « extrêmement important » (resp. « extrêmement peu important »). Ceci rend compte du fait que « très très important » (resp. « très très peu important ») se place entre les deux. Cette propriété élémentaire est construite par application du modificateur « très » sur la propriété élémentaire « très important » (resp. « très peu important ») (C*f.* §5).

4. Traitement d'une propriété élémentaire

Étant donné l'énoncé « x est $m_\alpha\ P_{ik}$ », la solution retenue consiste à déterminer de façon simple et systématique la fonction d'appartenance de la propriété $m_\alpha\ P_{ik}$ en fonction de celle de P_{ik} par une opération simple de translation et de contraction (ou dilatation suivant les conditions). Deux notions doivent alors être précisées, à savoir :

1. la direction de la translation ;
2. l'amplitude de la modification qui va dépendre : de l'amplitude de la translation et de l'amplitude de la contraction.

4.1. Propriétés d'un opérateur

Avant de détailler l'application d'un modificateur, nous posons deux définitions concernant, pour un ensemble donné d'opérateurs, les caractéristiques d'un opérateur par rapport aux autres.

Définition I.3.1 : La *distance* (ou *éloignement*) $d_{\alpha\beta}$ d'un opérateur o_α par rapport à l'opérateur o_β dans un ensemble d'opérateurs *E* est définie par : $d_{\alpha\beta} = |\alpha - \beta|$.

Définition I.3.2 : Deux opérateurs o_α et o_β sont *symétriques* par rapport à l'opérateur o_ϕ dans un ensemble d'opérateurs *E* si ($\alpha = \beta = \phi$) ou ($\alpha \neq \beta$ et $d_{\alpha\phi} = d_{\beta\phi}$).

4.2. Direction de la translation

La direction de la translation dépend du *signe de la propriété* (comportement de la fonction d'appartenance face aux modificateurs) et du *signe du modificateur* (sa place dans M_P).

Définition I.3.3 : Le *signe de* P_{ik} (+1, -1 ou 0) est celui de la direction suivie par les fonctions d'appartenance de $m_\alpha\ P_{ik}$ où $m_\alpha \in M_P$ et α allant de 1 à P par rapport à la variable du domaine.

Définition I.3.4 : Une propriété P_{ik} est dite *positive* si son signe est +1. Elle est dite *négative* si son signe est -1 et *neutre* si le signe est 0.

Exemple : Pour le concept concernant la longueur de la configuration, nous avons : « important » $\Rightarrow$ +1 (positive), « faible » $\Rightarrow$ -1 (négative) et « moyen » $\Rightarrow$ 0 (neutre).

Définition I.3.5 : Le *signe d'un modificateur* m_α, noté signe(m_α), est défini dans M_P comme-suit : signe(m_a) = 0 si $m_a = \emptyset$, signe(m_α) = +1 si $\alpha > a$ et signe(m_α) = -1 si $\alpha < a$.

Définition I.3.6 : On peut maintenant définir la *direction de translation d'une propriété* $m_\alpha\ P_{ik}$ par rapport à P_{ik}, notée Direction(P_{ik}) et calculée selon les règles classiques de l'arithmétique : Direction($m_\alpha\ P_{ik}$) = signe(m_α) * signe(P_{ik}) $\in \{-1, 0, +1\}$.

Exemples :

- « x est très important » : Direction(très important) = signe(très) * signe(important) = +1 * +1 = +1 ;
- « x est très faible » : Direction(très faible) = signe(très) * signe(faible) = +1 * -1 = -1.

Remarques :

- Le signe de la propriété élémentaire est celui de la direction de translation. Autrement dit, le signe de « très important » est +1.
- Ces définitions sont fidèles aux travaux linguistiques sur les catégories sémantiques présentées au chapitre I.1.

4.3. Amplitude de la modification

Après la direction de la modification, nous allons déterminer l'amplitude de la modification de la propriété élémentaire $m_\alpha\ P_{ik}$ par rapport à P_{ik}. Elle comprend une amplitude de translation et une amplitude de contraction ou de dilatation. Ces modificateurs dépendent d'une caractéristique du concept (plutôt des propriétés qui sont associées) que nous nommerons la symétrie linguistique.

4.3.1 Symétrie linguistique

Définition I.3.7 : La *symétrie linguistique* d'une propriété est déterminée par son comportement vis à vis de l'application de deux modificateurs symétriques par rapport à « Ø » dans M_P.

Il existe deux symétries linguistiques pour une propriété :

- la symétrie (comme la propriété du concept de placement « à gauche », Figure 15a) ;
- l'asymétrie (comme la propriété du concept de taille « grand », Figure 15b).

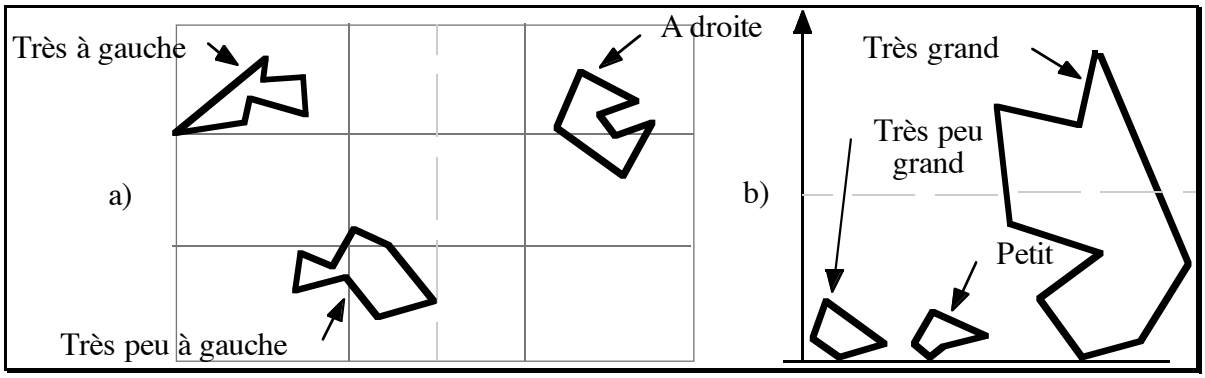

Figure 15. Symétrie et asymétrie des propriétés

Définition I.3.8 : Une *propriété symétrique* (Figure 16) est une propriété dont les amplitudes de contraction et de translation sont identiques pour deux modificateurs dont la position est symétrique par rapport à « Ø » dans M_P.

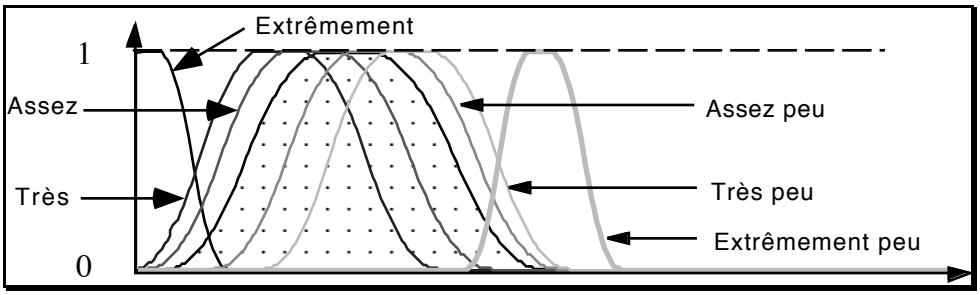

Figure 16. Modificateurs appliqués de la propriété de base symétrique « faible »

Définition I.3.9 : Une *propriété asymétrique* (Figure 17) est une propriété dont les amplitudes de contraction et de translation ne sont pas identiques pour deux modificateurs dont la position est symétrique par rapport à « Ø » dans M_P.

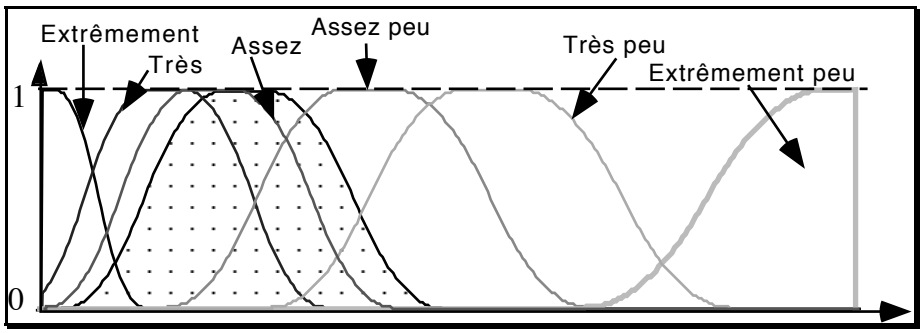

Figure 17. Modificateurs de la propriété de base asymétrique « faible »

Remarques :

- En particulier, pour celui placé avant le modificateur vide, on opère une dilatation. De plus, la translation est beaucoup plus importante que pour celui placé après (Cf. Figure 17).
- [DeP96a] propose seulement des solutions symétriques alors que [DeP97b] propose des solutions asymétriques. En fait, nous venons de montrer qu'il est possible de rencontrer les deux situations. Notons qu'ici, et contrairement à [DeP97b], l'asymétrie est due à la sémantique de la propriété utilisée et non aux modificateurs.

4.3.2 Détermination des amplitudes

Définition I.3.10 : *L'amplitude de translation de la propriété élémentaire* $m_\alpha\ P_{ik}$ par rapport à P_{ik} vaut 0 pour $m_\alpha = m_\varnothing = Ø$ et croît avec la distance $d_{\alpha\varnothing}$ de m_α par rapport à Ø dans M_P. Sa valeur dépend aussi de la sémantique de la propriété P_{ik} (en particulier de sa symétrie).

Définition I.3.11 : *L'amplitude de la contraction ou de la dilatation de la propriété élémentaire* $m_\alpha\ P_{ik}$ par rapport à P_{ik} est nulle pour $m_\alpha = m_\varnothing = Ø$ et croît en fonction de la distance $d_{\alpha\varnothing}$ de m_α par rapport à Ø dans M_P. Lorsque la propriété est symétrique, il y a contraction. Par contre, quand elle est asymétrique, il y a dilatation d'un côté et contraction de l'autre. De plus, cette amplitude est proportionnelle à la taille du support de P_{ik}.

Remarques :

- En général, la dilatation a lieu pour les modificateurs m_α négatifs et la contraction pour les positifs.
- Contrairement à [Chau94b] pour qui les propriétés neutres ne subissent aucune modification de la part des opérateurs, nous considérons qu'elles subissent malgré tout une contraction. En effet « très moyen » indique des valeurs plus proches de la valeur centrale que « moyen » si, bien sûr, cette dernière est considérée comme modifiable.

4.4. Bilan sur le traitement d'une propriété élémentaire

Nous avons vu que le modificateur m_α et la propriété de base P_{ik} interviennent aussi bien dans la direction de translation que dans l'amplitude de la modification. Afin de représenter ces participations, nous avons introduit un *coefficient de modification*, un *coefficient de translation élémentaire* et un *coefficient d'asymétrie*.

Définition I.3.12 : Le *coefficient de modification* k_α d'un modificateur m_α est un entier relatif tel que : signe(k_α) = signe(m_α), k_α=0 pour m_α=Ø et $|k_\alpha|$ est proportionnelle de la distance $d_{\alpha\varnothing}$ de m_α par rapport à Ø dans M_P.

En pratique, pour M_7={extrêmement peu, très peu, assez peu, Ø, assez, très, extrêmement}, nous avons choisi $K(M_7)$ ={-6, -2, -1, 0, 1, 2, 6}. Ces valeurs semblent assez conformes aux sémantiques relatives des différents modificateurs (définitions I.3.5, I.3.10, I.3.11 et I.3.12). Elles permettent une distribution satisfaisante des propriétés $m_\alpha P_{ik}$ sur le domaine.

Définition I.3.13 : Le *coefficient de translation élémentaire* τ_{ik} d'une propriété P_{ik} est un réel tel que : signe(τ_{ik}) = signe(P_{ik}), τ_{ik}=0 pour une propriété neutre et $|\tau_{ik}|$ est liée à la sémantique de la propriété.

Définition I.3.14 : Le *coefficient d'asymétrie linguistique* γ_{ik} d'une propriété P_{ik} est un réel tel que :

- $\gamma_{ik} = 0$ si la propriété est symétrique ;
- $\gamma_{ik} \neq 0$ si la propriété est asymétrique avec signe(γ_{ik}) indiquant la direction sur laquelle s'applique ce coefficient par rapport au signe de la propriété de base ;
- $|\gamma_{ik}|$ est liée à la sémantique de la propriété.

Ce coefficient d'asymétrie linguistique s'applique avec le coefficient de modification seulement lorsqu'il est différent de 0.

Concernant les valeurs $|\tau_{ik}|$ et $|\gamma_{ik}|$, l'annexe 3 propose des règles pour aider à leur détermination en fonction des caractéristiques attendues de la propriété et un exemple de calcul pour quelques cas de propriétés de base.

Sachant que $P_{ik} = \{D_i, <\alpha_{ik},a_{ik},b_{ik},\beta_{ik}>, L_{ik}, R_{ik}, \tau_{ik}, \gamma_{ik}\}$, pour « x est $m_\alpha P_{ik}$ », on a :

- la direction de translation : signe(k_α) * signe(τ_{ik}) (définition I.3.6) ;
- l'amplitude de translation : $|k_\alpha*\gamma_{ik}*\tau_{ik}|$ ou $|k_\alpha*\tau_{ik}|$ (définition I.3.10) ;
- l'amplitude de la contraction : $|k_\alpha|$ * (b_{ik}-a_{ik})*10% sur le noyau et $|k_\alpha|$*10% sur la partie floue (définition I.3.11) ;
- l'amplitude de la dilatation : $\gamma_{ik}*k_\alpha*(b_{ik}-a_{ik})*10\%$ sur le noyau et $\gamma_{ik}*k_\alpha*10\%$ sur la partie floue (définition I.3.11).

Remarque : Nous appelons *partie floue* de P_{ik} l'ensemble défini à partir du noyau et du support telle que PartieFloue(P_{ik})=Support(P_{ik})-Noyau(P_{ik}) soit $\{d \in D_i : 0 < \mu_{P_{ik}}(d) < 1\}$.

Nous avons donc : $m_\alpha P_{ik} = \{[Bm,BM]_u, <\alpha'_{ik}, a'_{ik}, b'_{ik}, \beta'_{ik}>, L_{ik}, R_{ik}, \tau_{ik}, \gamma_{ik}\}$ Avec :

si ($\tau_{ik} \neq 0$) et ($\gamma_{ik} \neq 0$) et (Signe(k_α) = Signe(γ_{ik})) {fortes translation et dilatation} alors	sinon {translation classique et contraction }
$\alpha'_{ik} = \alpha_{ik}*(1 + \gamma_{ik}*k_\alpha * 10\%)$;	$\alpha'_{ik} = \alpha_{ik} * (1 - \|k_\alpha\| * 10\%)$;
$\beta'_{ik} = \beta_{ik}*(1 + \gamma_{ik}*k_\alpha * 10\%)$;	$\beta'_{ik} = \beta_{ik} * (1 - \|k_\alpha\| * 10\%)$;
$a'_{ik} = a_{ik} + k_\alpha*\|\gamma_{ik}\|*\tau_{ik} - \gamma_{ik}\, k_\alpha *10\%(b_{ik} - a_{ik})/2$;	$a'_{ik} = a_{ik} + k_\alpha*\tau_{ik} + \|k_\alpha\| *10\%(b_{ik} - a_{ik})/2$;
$b'_{ik} = b_{ik} + k_\alpha*\|\gamma_{ik}\|*\tau_{ik} + \gamma_{ik}\, k_\alpha *10\%(b_{ik} - a_{ik})/2$;	$b'_{ik} = b_{ik} + k_\alpha*\tau_{ik} - \|k_\alpha\| *10\%(b_{ik} - a_{ik})/2$;

Remarques :

- Sur l'ensemble des exemples traités, cette transformation rend compte de façon satisfaisante de la modification d'une propriété positive (« très important ») ou négative (Figure 18 : « très faible ») par une translation, de l'asymétrie linguistique des propriétés. Il traite aussi l'application d'un modificateur sur une propriété neutre comme « très moyen » où seule la contraction est présente.
- Le coefficient de 10% utilisé pour ces transformations a été choisi intuitivement en fonction des résultats obtenus.

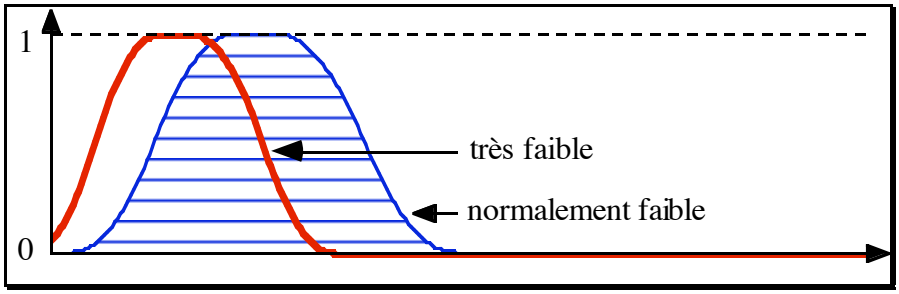

Figure 18. Application du modificateur « très » à la propriété de base « faible »

Exemple : La Figure 19 montre deux solutions possibles lorsqu'on applique successivement les descriptions suivantes :

(a) « *Le nombre de verticales est faible* » ;

(b) « *La longueur de recouvrement est très importante* » (alors que nous constatons qu'en (a) elle est « extrêmement importante ») + « *Le nombre de verticales est faible* ».

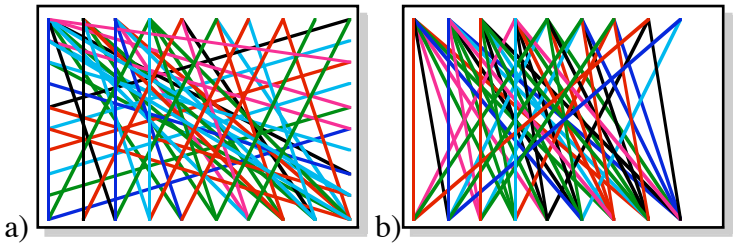

Figure 19. Scènes vérifiant d'abord une (a) puis deux (b) propriétés élémentaires demandées par l'utilisateur

Rappels ([Paj94]) : Le degré de recouvrement en un point d'une ligne de projection est « égal au nombre de segments pour lesquels le point est inclus dans la projection ». La longueur de recouvrement est « le nombre de degrés de recouvrement supérieurs ou égaux à 1 ».

5. Composition de modificateurs sur une propriété de base

L'application d'un modificateur m_α sur une propriété modifiable P_{ik} donne une propriété modifiable $m_\alpha\ P_{ik}$. Il est donc possible d'employer à nouveau un modificateur m_β. Il suffit de postuler l'équivalence suivante : « x est $m_\beta\ m_\alpha\ P_{ik}$ » ⇔ « x est $m_\beta\ (m_\alpha\ P_{ik})$ ». On peut donc

construire des descriptions comme : « *La longueur de la configuration est très très importante* », « *La scène est très très peu remplie* »… En pratique, les possibilités d'applications successives de modificateurs sur une propriété élémentaire sont assez restreintes dans la langue française. Nous proposons donc les quatre règles suivantes :

1. Seuls les opérateurs « très » et Ø peuvent être répétés plusieurs fois dans un énoncé ;
2. « très » porte sur une propriété de dernier modificateur « Ø », « très » ou « très peu » ;
3. On peut appliquer n'importe quel modificateur sur la propriété élémentaire Ø P_{ik} ;
4. « très » est le seul modificateur s'appliquant à une propriété élémentaire autre que Ø P_{ik}.

Remarques :

- Pour appliquer un modificateur sur une propriété de base, il est nécessaire de connaître le signe de cette propriété. Lorsque la propriété de base est une propriété élémentaire (composition de modificateurs), quel sera le signe de cette propriété ? Le signe de m_α P_{ik} sera celui de la direction de translation. Afin de garder le formalisme cohérent, il convient alors de remplacer τ_{ik} par τ'_{ik} dans la formule de m_α P_{ik} du paragraphe précédent, ce qui donne : $\tau'_{ik} = \text{signe}(k_\alpha)*\tau_{ik}$.
- En pratique, la seule répétition possible est « très très ». Il est donc envisageable d'intégrer « très très » et « très très peu » dans la liste des modificateurs possibles en leur attribuant respectivement les coefficients 4 et -4. Ainsi, il n'est pas nécessaire de gérer les répétitions. Nous avons alors : M_9={extrêmement peu, très très peu, très peu, assez peu, normalement, assez, très, très très, extrêmement} et $K(M_9)$ ={-6, -4, -2, -1, 0, 1, 2, 4, 6} (Figure 20).

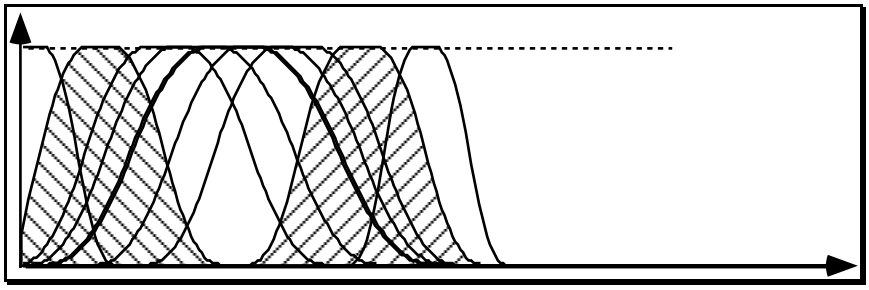

Figure 20. Ensemble des modificateurs (y compris « très très » et « très très peu »)

6. Opérateurs flous sur une propriété

Il s'agit de gérer des opérateurs augmentant la précision ou l'imprécision d'une propriété élémentaire. Le locuteur utilise alors un énoncé de la forme « C_i de x est f_α m_β P_{ik} » où f_α est un opérateur flou (cf. définition I.2.2). Pour traiter un tel énoncé, il suffit de postuler l'équivalence suivante : « x est f_α m_β P_{ik} » ⇔ « x est f_α (m_β P_{ik}) ». Nous allons nous intéresser à l'application de cet opérateur flou sur la propriété élémentaire « m_β P_{ik} ».

6.1. Les opérateurs flous retenus

Les opérateurs flous possibles étant très nombreux, nous avons restreint notre choix à un ensemble noté $F_Q=\{f_\alpha \mid \alpha \in [1..Q]\}$ muni de la relation d'ordre total : $f_\alpha \leq f_\beta \Leftrightarrow \alpha \leq \beta$. F_Q comporte un opérateur par défaut noté « $Ø_f$ ». Nous avons choisi l'ensemble F_6={précisément, vraiment, $Ø_f$, approximativement, plus ou moins, vaguement}. Par exemple, pour le concept « nombre d'intersections », l'application des opérateurs flous à la propriété de base « important(e) » produit les sous-ensembles flous rassemblés dans la Figure 21.

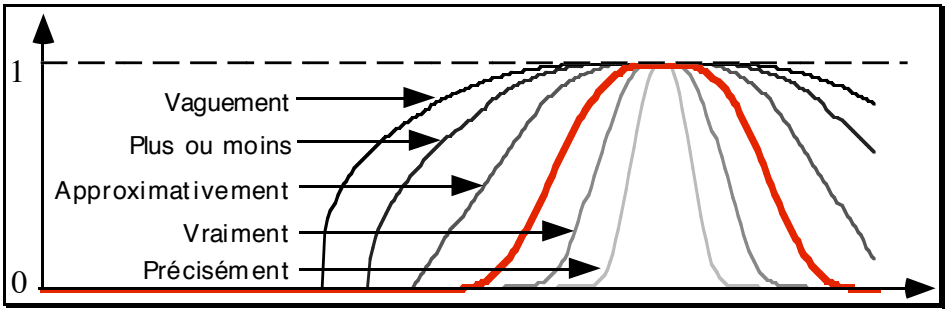

Figure 21. Opérateurs flous appliqués à la propriété élémentaire « Ø important(e) »

6.2. Traitement d'un opérateur flou sur une propriété élémentaire

Étant donné l'énoncé « x est f_α m_β P_{ik} », la solution retenue consiste à déterminer de façon simple et systématique la fonction d'appartenance de la propriété « f_α m_β P_{ik} » en fonction de celle de m_β P_{ik} par une opération simple de contraction ou de dilatation (*Cf.* Figure 21).

Remarque : Les opérateurs flous sont issus des opérateurs de contraction et de dilatation classiques en logique floue. Cependant, contrairement à la définition des modificateurs choisie par d'autres auteurs, dans notre approche la modification s'effectue non seulement sur la partie floue mais aussi sur le noyau. Ils ne dépendent pas de la sémantique de la propriété P_{ik} sur laquelle ils portent, c'est-à-dire de τ_{ik}, de son signe ou de sa place et de sa forme dans le domaine.

6.2.1 Contraction ou dilatation

Définition I.3.15 : *L'ordre* d'un opérateur flou f_α, noté ordre(f_α), est défini dans F_q comme suit : ordre(f_a) = 0 si $f_a = Ø_f$, ordre(f_α) = +1 si $\alpha > a$ et ordre(f_α) = -1 si $\alpha < a$.

Donc, si ordre(f_α) = +1, il y a dilatation, et si ordre(f_α) = -1, il y a contraction. Si ordre(f_α) est nul, la propriété élémentaire reste inchangée.

6.2.2 Amplitude de la modification

Définition I.3.16 : *L'amplitude de la modification* d'une propriété élémentaire m_β P_{ik} par un opérateur flou f_α vaut 1 pour $f_\alpha = Ø_f$ et croît avec la distance $d_{\alpha ø}$ de f_α par rapport à $Ø_f$ dans

F_Q. Il y a contraction lorsque l'opérateur se trouve avant $Ø_f$ et dilatation lorsqu'il se trouve après.

6.2.3 Bilan sur l'application d'un opérateur flou

Un opérateur flou contracte ou dilate la fonction d'appartenance de la propriété élémentaire. Afin de représenter cette action de modification, nous avons introduit un *coefficient de flou*.

Définition I.3.17 : Le *coefficient de flou* j_α associé à un opérateur flou f_α est un réel tel que : $j_\alpha = 1$ si ordre(f_α) = 0, $j_\alpha > 1$ si ordre(f_α) = +1 et $0 < j_\alpha < 1$ si ordre(f_α) = -1. De plus, $j_\alpha^{\text{ordre}(f_\alpha)}$ est proportionnel à la distance $d_{\alpha ø}$ de f_α par rapport à $Ø_f$ dans F_Q.

En pratique, pour F_6={précisément, vraiment, $Ø_f$, approximativement, plus ou moins, vaguement}, nous avons choisi $J(F_6)= \{4^1, 2^1, 1, 2^{-1}, 4^{-1}, 6^{-1}\}$. Ces valeurs semblent assez conformes aux sémantiques relatives des différents modificateurs (définitions I.3.15, I.3.16 et I.3.17). Elles permettent une distribution satisfaisante des propriétés $f_\alpha\ m_\beta\ P_{ik}$ par rapport à $m_\beta\ P_{ik}$.

La propriété élémentaire $m_\beta\ P_{ik}$ étant définie par $\{D_i, <\alpha_{ik}, a_{ik}, b_{ik}, \beta_{ik}>, L_{ik}, R_{ik}, \tau_{ik}\}$, l'application d'un opérateur flou f_α sur cette propriété permet d'obtenir un énoncé « x est $f_\alpha\ m_\beta\ P_{ik}$ » qui a pour fonction d'appartenance :

$$f_\alpha\ m_\beta\ P_{ik} = \{[Bm,BM]_u, <\alpha'_{ik}, a'_{ik}, b'_{ik}, \beta'_{ik}>, L'_{ik}, R'_{ik}, \tau_{ik}\}$$

Avec :

$L'_{ik} = L_{ik}^{j_\alpha}$;

$\alpha'_{ik} = \alpha_{ik} * (1 - j_\alpha * 10\%)$ si $j_\alpha > 1$,
$\alpha_{ik} * (1 + 1/j_\alpha * 10\%)$ si $0 < j_\alpha < 1$,
α_{ik} si $j_\alpha = 1$;

$\beta'_{ik} = \beta_{ik} * (1 - j_\alpha * 10\%)$ si $j_\alpha > 1$,
$\beta_{ik} * (1 + 1/j_\alpha * 10\%)$ si $0 < j_\alpha < 1$,
β_{ik} si $j_\alpha = 1$;

$R'_{ik} = R_{ik}^{j_\alpha}$;

$a'_{ik} = a_{ik} + (b_{ik} - a_{ik}) * j_\alpha * 10\%$ si $j_\alpha > 1$,
$a_{ik} - (b_{ik} - a_{ik}) * 1/j_\alpha * 10\%$ si $0 < j_\alpha < 1$,
a_{ik} si $j_\alpha = 1$;

$b'_{ik} = b_{ik} - (b_{ik} - a_{ik}) * j_\alpha * 10\%$ si $j_\alpha > 1$,
$b_{ik} + (b_{ik} - a_{ik}) * 1/j_\alpha * 10\%$ si $0 < j_\alpha < 1$,
b_{ik} si $j_\alpha = 1$.

Remarques :

- Le coefficient de 10% utilisé pour ces transformations a été choisi intuitivement en fonction des résultats obtenus.
- Sur l'ensemble des exemples traités, cette transformation rend compte de façon satisfaisante de la modification d'une propriété à l'aide d'un opérateur flou. Par exemple, l'application successive du modificateur « assez » et de l'opérateur flou « plus ou moins » sur la propriété de base « important » donne la transformation proposée par la Figure 22.

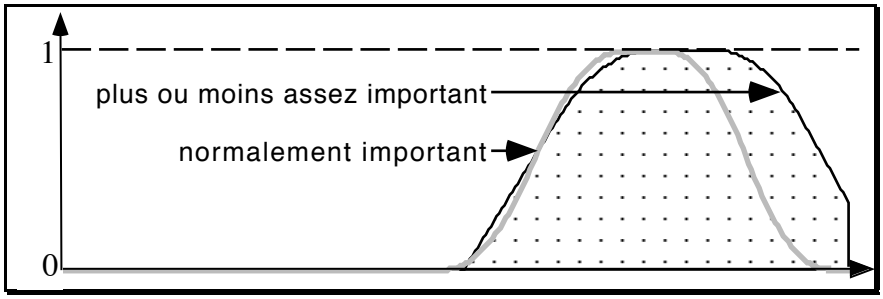

Figure 22. Application de l'opérateur « plus ou moins » à la propriété élémentaire « assez important »

Exemple : A l'aide des propriétés suivantes, on obtient des scènes plausibles du type de celles présentées dans la Figure 23 :

a) « *La longueur de recouvrement est très importante. Le nombre de verticales est faible. La proportion de segments parallèles est vraiment faible* » ;

b) « *Le degré moyen de recouvrement est plus ou moins assez important* » + a).

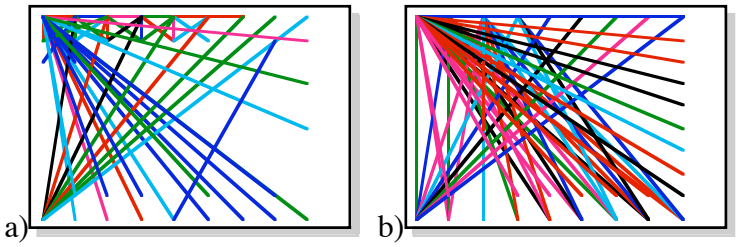

Figure 23. Scènes vérifiant des propriétés élémentaires avec des opérateurs flous

Remarques :

- Il est évident que le nombre de combinaisons d'opérateurs et de modificateurs peut être élevé et que certaines ne sont pas pertinentes dans le langage naturel. Ce point n'est pas abordé dans le cadre de ce travail. Nous pouvons cependant noter que les combinaisons retenues se limitent à deux termes linguistiques, restriction en accord avec les linguistes pour qui l'homme ne maîtrise assez bien que ce cas maximal. De plus les « ø », « øø »… sont ramenés au mot vide. Ces limitations pratiques réduisent déjà énormément le nombre de combinaisons possibles.
- Le concepteur d'un modeleur déclaratif peut modifier les valeurs des différents coefficients selon les critères du domaine d'application. Par contre, l'utilisateur n'en a pas conscience. Nous verrons en conclusion de ce document qu'un module d'apprentissage permettra d'apprendre leur sens et de les adapter à son langage personnel. Pour ajuster ces coefficients, ce module utilise principalement la description et les commentaires de l'utilisateur vis-à-vis des solutions proposées. Ainsi, les effets des opérateurs s'adaptent aux attentes de l'utilisateur tout au long des utilisations successives du modeleur.
- Les transformations effectuées sur « moyen » par les opérateurs « très » et « vraiment » sont similaires. La différence se situe dans les fonctions L et R de la propriété élémentaire obtenue. Contrairement à l'opérateur flou, le modificateur ne les change pas.

- Lorsqu'on applique un modificateur sur une fonction d'appartenance d'une propriété modifiable, il change la forme de la courbe de la fonction par une translation et une contraction/dilatation. Si la courbe de départ est "collée" à une borne du domaine et que la translation se fait vers la borne opposée, la forme de la courbe obtenue n'est pas triviale. Plusieurs politiques sont envisageables ([Des95b]). Cependant, le plus simple est de définir la fonction même en dehors du domaine. Elle est alors utilisée comme les autres, il n'y a pas de traitement spécifique.

7. Construction automatique des propriétés de base d'un concept

L'objectif est de proposer des méthodes de calcul automatique des propriétés de base d'un concept (Figure 24) permettant d'obtenir un premier ensemble. Si les coefficients proposés ne conviennent pas, le concepteur pourra éventuellement les ajuster en fonction des caractéristiques du concept dans le domaine d'application. Nous supposerons par la suite que les valeurs du domaine sont des valeurs numériques (sachant que, dans notre étude, on peut toujours s'y rapporter). Plus précisément, il s'agit de déterminer les fonctions d'appartenance des propriétés de base ainsi que les coefficients de translation élémentaire et d'asymétrie. Les calculs, les hypothèses et les critères de construction sont détaillés en Annexe 3. Les résultats en sont les coefficients figurant dans les tableaux qui suivent. Dans la pratique, ils permettent d'obtenir des propriétés de base satisfaisantes pour la plupart des cas.

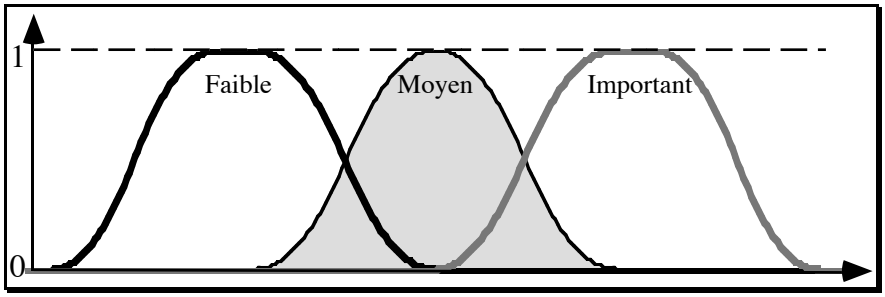

Figure 24. Propriétés de base d'un concept.

Soit $[Bm,BM]_u$ le domaine, on a alors δ la taille du domaine telle que δ = BM-Bm. Soit M une valeur de ce domaine. On a alors : δ'=M−Bm et δ''=BM−M. Le Tableau 1 propose une première définition des propriétés. La propriété « moyen » est centrée dans le domaine.

Tableau 1. Définition automatique des fonctions d'appartenance

nom	α	a	b	β	τ_{ik}	γ_{ik}
faible	20% * δ	Bm + 23% * δ	Bm + 28% * δ	22% * δ	-4.25% * δ	-3
moyen	22% * δ	Bm + 50% * δ	Bm + 50% * δ	22% * δ	0	0
important	22% * δ	Bm + 72% * δ	Bm + 77% * δ	20% * δ	4.25% * δ	-3

Nous pouvons noter que ces valeurs de coefficient ont permis d'obtenir en particulier les résultats de la Figure 14 et de la Figure 16 de ce chapitre.

Lorsqu'on désire paramétrer la valeur M sur laquelle sera centrée « moyen », nous avons alors les propriétés décrites dans le Tableau 2.

Tableau 2. Définition automatique des fonctions d'appartenance en fonction d'une valeur moyenne

nom	α	a	b	β	τ_{ik}	γ_{ik}
faible	40% * δ'	Bm + 46% * δ'	Bm + 56% * δ'	44% * δ'	-8.5% * δ'	$\frac{50}{27}\frac{\delta}{\delta'} - \frac{28}{27} \le \lvert\gamma_{ik}\rvert \le \frac{25}{12}\frac{\delta}{\delta'} - \frac{23}{24}$
moyen	44% * δ'	M	M	44% * δ''	0	0
important	44% * δ''	M + 44% * δ''	M + 54% * δ''	40% * δ''	8.5% * δ''	$\frac{50}{27}\frac{\delta}{\delta''} - \frac{28}{27} \le \lvert\gamma_{ik}\rvert \le \frac{25}{12}\frac{\delta}{\delta''} - \frac{23}{24}$

Remarque : si nous posons δ'= δ''= δ/2, nous obtenons les résultats du Tableau 1.

Parfois, le concept ne comporte qu'une seule propriété qu'on nommera « vérifié ». Nous obtenons la propriété du Tableau 3 quand cette propriété est centrée sur la valeur M.

Tableau 3. Définition d'une propriété de base unique

nom	α	a	b	β	τ_{ik}	γ_{ik}
Vérifié	20% * δ	M - 5% * δ	M + 5% * δ	20% * δ	$\tau_{ik} = \pm Min(\frac{M-Bm}{6}, \frac{BM-M}{6})$	$\lvert\gamma_{ik}\rvert = \frac{Max(\frac{M-Bm}{6}, \frac{BM-m}{6})}{Min(\frac{M-Bm}{6}, \frac{BM-m}{6})}$

Le signe de γ_{ik} est donné par le Tableau 4 en fonction du domaine et de τ_{ik}.

Tableau 4. Détermination de γ en fonction de la position de la propriété

	δ' < δ''	δ' = δ''	δ' > δ''
τ > 0	γ > 1	γ = 0	γ < 1
τ < 0	γ < 1	γ = 0	γ > 1

Le coefficient de translation élémentaire τ_{ik} pour une propriété positive sur la valeur médiane (BM-M = M-Bm = δ/2) est de (8.333% * δ). Nous obtenons ainsi la Figure 25.

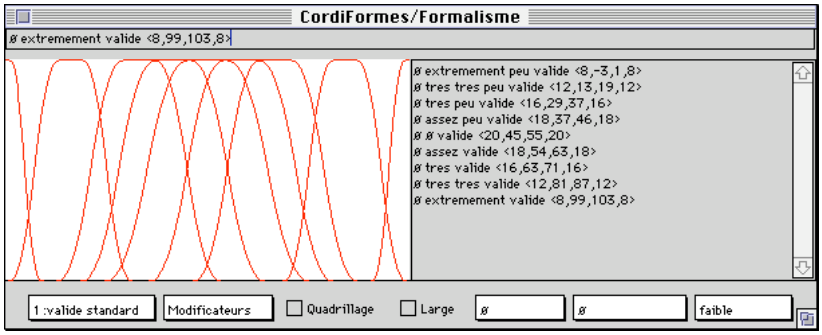

Figure 25. Modificateurs sur la propriété de base « valide », seule propriété de base du concept

Remarques :

- Cette étude est valable pour la définition de toute propriété modifiable.
- Dans certains cas, cette unique fonction peut être définie comme une propriété de base positive ou négative. Nous retrouvons alors les définitions et les coefficients présentés au Tableau 2. Dans ce cas, la propriété sera alors forcément linguistiquement asymétrique afin de parcourir tout le domaine. La Figure 26 illustre ce cas.

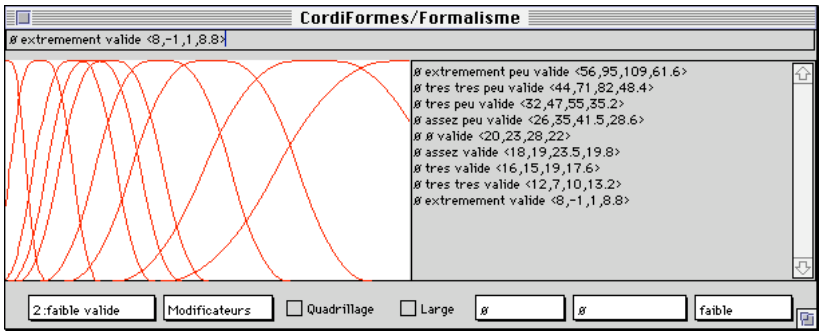

Figure 26. Modificateurs sur « valide » comme « faible »

Dans certaines situations, le concepteur veut pouvoir construire un concept comportant trois propriétés de base telles que chacune des propriétés signées soit définie uniquement sur son demi-domaine. Elles sont donc totalement disjointes des autres (par exemple le concept de placement avec les propriétés « à gauche » et « à droite » comme le souligne la Figure 15 du paragraphe 4.2.1). Ces propriétés peuvent être définies comme la propriété que nous venons de spécifier au paragraphe précédent en remplaçant $[Bm,BM]_u$ par $[Bm, M]_u$ pour « faible » et $[M, BM]_u$ pour « important » (M étant la valeur médiane). Nous aurons alors : τ_{ik} =8.333%*δ' avec δ' = δ/2. Notons qu'il sera nécessaire d'ajouter un filtre afin que les propriétés construites à partir de « extrêmement peu » n'empiètent pas sur l'autre demi-domaine. Pour cela, il suffit d'ajouter un test à la fin de la fonction de modification tel que si la valeur n'appartient pas au domaine de validité de la propriété de base ($[Bm, M]_u$ pour « faible » et $[M, BM]_u$ pour « important ») alors elle n'appartient à aucune propriété issue de celle-ci.

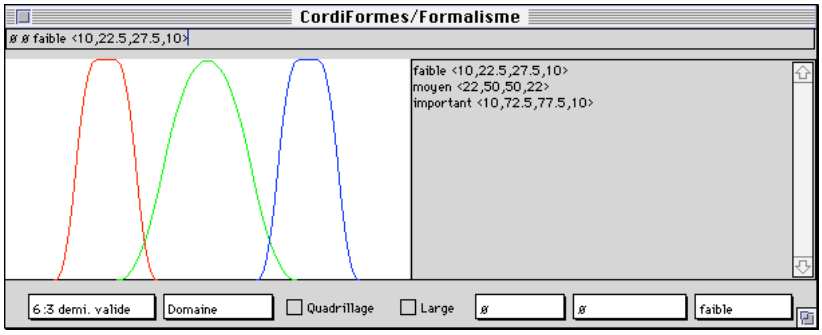

Figure 27. Domaine à base de « valides »

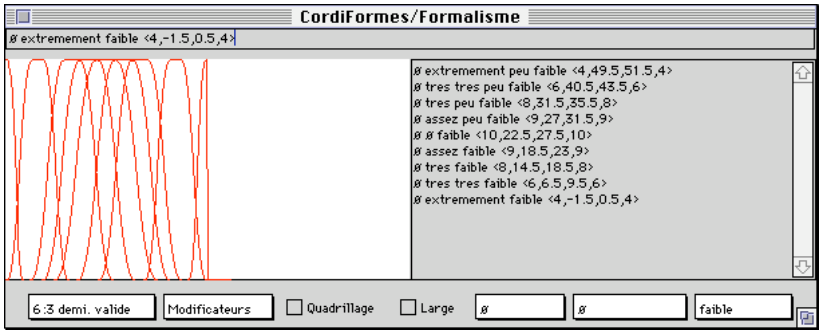

Figure 28. Application des modificateurs sur "faible"

8. Cas particulier des propriétés élémentaires paramétrées

Nous allons nous intéresser ici au cas des propriétés paramétrées (définition I.2.4). Plus précisément, nous étudierons le traitement des deux classes de propriétés de ce type : les intervalles et les propriétés de comparaison élémentaires.

8.1. Les intervalles

Une propriété élémentaire paramétrée de la classe intervalle (définition I.2.5) est un énoncé de la forme : « C_i de X est f_α entre U et V » où U et V sont deux valeurs de D_i, domaine du concept C_i.

Une propriété élémentaire P_{ik} d'un concept C_i est caractérisée par une fonction d'appartenance LR, un coefficient de translation élémentaire τ_{ik} et un coefficient d'asymétrie linguistique et s'écrit : $\{[Bm, BM]_u, <\alpha_{ik}, a_{ik}, b_{ik}, \beta_{ik}>, L_{ik}, R_{ik}, \tau_{ik}, \gamma_{ik}\}$. Une propriété élémentaire paramétrée P_{pi} est, elle aussi, caractérisée par une fonction d'appartenance LR. La difficulté est de déterminer les paramètres de cette fonction. Quatre solutions sont envisageables (Figure 29) avec P_{pi} = « Entre U et V » = $<\alpha,U, V, \beta>$:

- intervalle classique [U, V] qui s'écrit aussi <0, U, V, 0> (Figure 29a) ;
- intervalle flou $<\alpha,U, V, \beta>$ (Figure 29b) ;
- intervalle flou $<\alpha,U+\alpha, V-\beta, \beta>$ (Figure 29c) ;
- intervalle flou $<\alpha,U+\alpha', V-\beta', \beta>$ (Figure 29d).

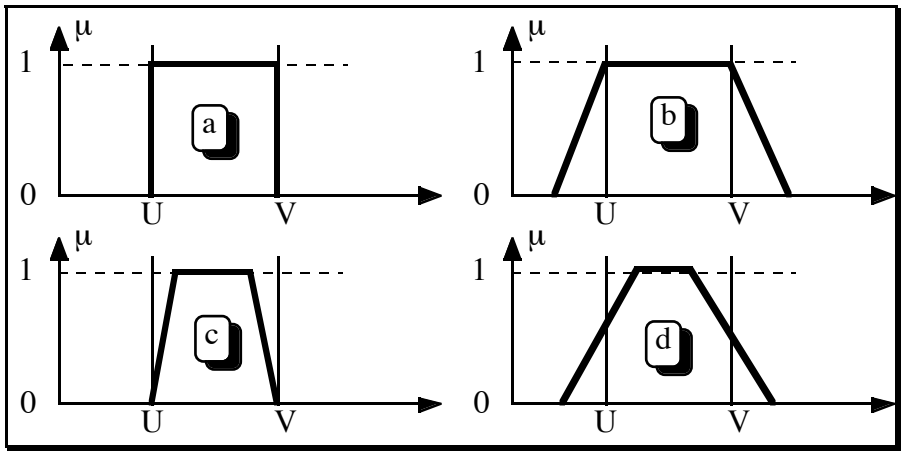

Figure 29. Différentes caractérisations possibles de « Entre U et V »

Cependant, lorsqu'un locuteur propose une propriété paramétrée, il s'attend souvent à ce que toute valeur entre les bornes énoncées vérifie totalement la propriété. Par conséquent, les deux dernières solutions ne sont pas à retenir dans ce cas. Il reste donc deux manières de représenter une propriété paramétrée : l'intervalle classique ou l'intervalle flou (Figure 29a, Figure 29b).

Remarque : En logique floue, la propriété « au moins de V » (resp. « au plus de V ») est souvent représentée par un intervalle flou défini par une fonction vérifiant : <α, V, +∞,0> (resp. <0, -∞, V ,β>). Cependant, il reste toujours à déterminer une bonne valeur pour α (resp. β).

8.1.1 La propriété paramétrée comme un intervalle flou

D'un point de vue pratique, la seconde solution (Figure 29b) est intéressante car les opérateurs flous habituels sont utilisables (ce qui n'est pas le cas lorsqu'on utilise l'intervalle classique). Cependant, il reste à déterminer les valeurs de α et β.

Avec les formules d'application des opérateurs flous (§7.3.7), nous constatons que pour les opérateurs les plus faibles, la contraction ou la dilatation sur α et β est de 20% ($j_\alpha = 2$ ou 1/2). Or lorsque le locuteur énonce la propriété suivante « X est vraiment entre 2 et 3 » (et plus généralement pour tous les opérateurs provoquant une contraction), il s'attend à n'obtenir que des solutions strictement entre ces bornes. Par conséquent, nous proposons de déterminer α et β tels que : Supp(« X est vraiment entre U et V ») ⊆ [U, V] avec « Supp(A) » le support de la fonction d'appartenance de A (Annexe 1). Nous en déduisons que : α =β = j(b-a)/(10-j). D'où, comme j= $j_{vraiment}$ = 2, nous aurons α = β ≤ (b-a)/4. Par exemple, « entre 2 et 4 » serait représentée par la fonction LR <.5, 2, 4,.5 > (Figure 30). Il nous reste le problème de déterminer les fonctions L et R de la fonction d'appartenance. Logiquement, lorsqu'on énonce une propriété paramétrée telle que « entre 2 et 3 », on s'attend à trouver les plus forts degrés (en dehors de ceux entre les deux valeurs) vraiment très proches des bornes. Par conséquent, la fonction la mieux adaptée, aussi bien pour L que pour R, est une fonction tendue comme L_3-R_3 (voir Annexe 2 et Figure 30).

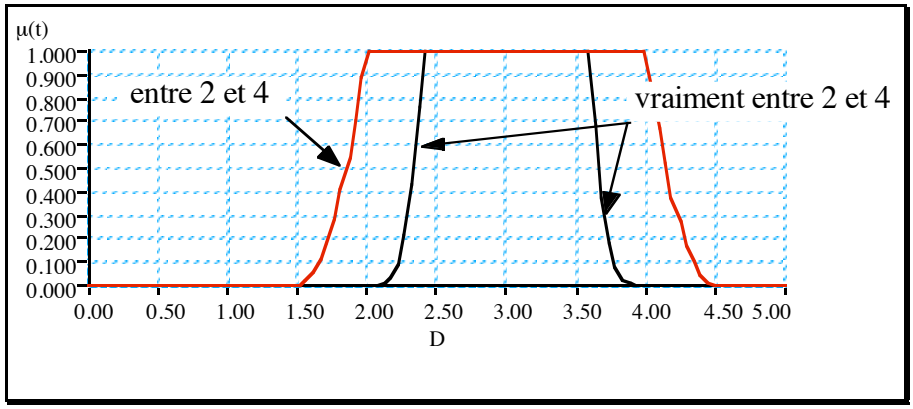

Figure 30. Fonctions d'appartenance pour « entre 2 et 4 » et « vraiment entre 2 et 4 »

Ainsi, il est possible d'appliquer aussi bien des opérateurs de contraction que des opérateurs de dilatation comme « plus ou moins »… Par exemple, l'énoncé « X est plus ou moins entre 2 et 4 » se traite de façon classique.

Éventuellement, si pour un concept spécifique la propriété paramétrée doit être imprécise mais pas floue, nous pouvons imaginer un opérateur spécifique permettant de la transformer en intervalle classique. Par exemple, nous pouvons choisir « strictement » pour jouer ce rôle. Ainsi, « strictement entre 2 et 4 » sera représenté par [2,4].

Il reste encore un autre problème à gérer : la propriété précise. En effet, comment représenter « de 2 » ? Nous avons vu que l'énoncé « x est de U » est équivalent à « x est entre U et U ». Si on garde la méthode de représentation que nous avons présentée dans ce paragraphe, cet énoncé sera représenté par <0, 2, 2, 0>. Cela ne pose pas de problème pour l'application des opérateurs contractants (« vraiment » et « précisément ») car l'énoncé normal ne peut pas être plus précis. Cependant, l'énoncé « X est approximativement de 2 » ne peut pas être traité convenablement puisqu'il donnera toujours <0, 2, 2, 0>. Il est donc nécessaire d'effectuer un traitement spécifique pour un énoncé précis. L'objectif est de déterminer α et β sans autres informations sur la propriété.

Dans le cas où le domaine associé au concept est borné (cas le plus courant) par $[Bm, BM]_u$, nous pouvons imaginer par exemple que : $\alpha = \beta = p\%*(BM-Bm)$ avec p égal à 1 ou 10 (mais cela reste à déterminer). Plus généralement, connaissant uniquement Bm (resp. BM) nous avons au moins : $\alpha = p\%*(U-Bm)$ (resp. $\beta = p\%*(BM-U)$). Enfin, une autre solution est possible quand le domaine est discret, c'est-à-dire possède une unité notée u. Nous avons alors : $\alpha = \beta = u$. Notons que ces deux solutions ne sont pas exclusives. Nous pourrions avoir par exemple : $\alpha = \beta = Max(u, p\%*(BM-Bm))$.

Par contre, si aucune de ces situations n'est possible, il n'y a pas de solutions calculables. Il est cependant possible de se baser sur des raisonnements « inverses » en se posant la question suivante : ayant une idée de la forme de « plus ou moins de X », en appliquant la fonction inverse de l'opérateur « plus ou moins », est on capable d'en déduire la forme de « de X » ? Connaissant l'opérateur utilisé, cette fonction est facile à déterminer pour une propriété de la classe intervalle « entre U et V » (et l'est encore plus pour une propriété « de V » car elle suppose a'=b'=a=b). De plus, inverser la fonction d'application d'un opérateur flou (de la forme $\alpha'=k\alpha$, $\beta'=k\beta$, $a'=a-l(b-a)$, $b'=b+l(b-a)$ avec k et l deux constantes déterminées à partir de l'opérateur utilisé) ne pose pas de problèmes particuliers. Cependant, dans les conditions où le domaine n'est que partiellement borné et ne possède pas d'unité, l'énoncé « approximativement de X » n'a pas réellement de sens (« *La distance de la planète est d'approximativement 2 années lumière* » ne donne pas d'idées sur les bornes autorisées, car les bornes de l'univers ne sont pas connues). Il est donc envisageable de considérer l'énoncé « approximativement de U » comme ambigu. Plutôt que d'essayer de calculer arbitrairement une valeur floue, il faut alors demander à l'utilisateur de préciser sa pensée en lui proposant de fournir plutôt un énoncé de la forme « Ø entre U et V ».

8.1.2 La propriété paramétrée comme un intervalle classique

Avec cette vision des choses, plutôt que de faire un traitement spécifique pour les énoncés précis, nous proposons d'étendre cela à toutes les propriétés paramétrées. L'énoncé « entre U et V » est alors un intervalle classique représenté par <0, U, V, 0> (ou [U, V]). L'objectif est de concevoir une fonction d'application des opérateurs flous spécifique pour cette catégorie d'énoncés (Figure 31). Une solution découle directement de celle proposée au paragraphe précédent. Avant d'appliquer effectivement l'opérateur, il suffit de calculer α et β « virtuels » selon la formule proposée ($\alpha = \beta = (b-a)/4$). Une autre solution est de déterminer α et β en fonction de la modification de l'intervalle du noyau. En effet, la fonction habituelle effectue aussi une réduction ou une dilatation du noyau de la fonction. Par conséquent, nous pouvons imaginer d'affecter tout ou une partie seulement de la variation du noyau.

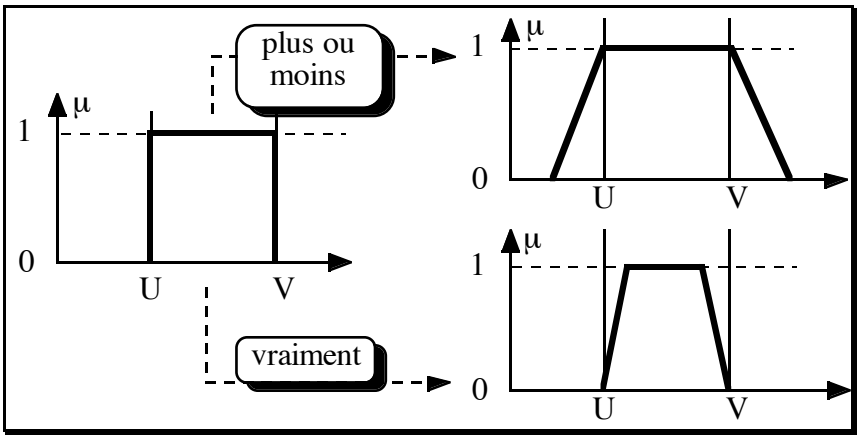

Figure 31. Opérateurs flous sur un intervalle classique

Notons cependant que ces méthodes ne sont pas non plus utilisables sur les valeurs précises. En effet, comme le noyau est réduit à une valeur, selon les formules, il ne sera pas modifié et (b-a)=0. Il faut donc, là encore, appliquer les techniques présentées au paragraphe précédent pour déterminer α et β. Ainsi, quand on applique un opérateur flou, il y a un traitement pour les propriétés élémentaires et un pour les propriétés paramétrées.

8.2. Les comparaisons élémentaires

Une propriété élémentaire paramétrée de la classe comparaison élémentaire (définition I.2.6) est un énoncé de la forme : « C_i de X est f_α m_β supérieur à U » ou « C_i de X est f_α m_β inférieur à U » où U est une valeur de D_i, domaine du concept C_i. Dans ce cas, il est possible d'appliquer aussi bien les modificateurs que les opérateurs flous. La propriété de base « supérieur à U » (resp. « inférieur à U ») est donc une propriété à laquelle est associé un intervalle flou (propriété modifiable). La difficulté essentielle est de déterminer la fonction d'appartenance d'une telle propriété de base. Elle sera définie comme les propriétés de base

classiques (« faible », « important »), en particulier avec un coefficient de translation élémentaire dépendant du concept.

Nous pouvons malgré tout décrire un certain nombre de caractéristiques (Figure 32). Le domaine sur lequel sera définie la propriété de base « supérieur à » (resp. « inférieur à ») sera $[U, BM]_u$ (resp. $[Bm, U]_u$). Nous appellerons ces domaines des pseudo-domaines. D'un point de vue linguistique, ces propriétés sont asymétriques ($\gamma_{ik} \neq 0$). La translation est de plus en plus importante au fur et à mesure qu'on s'éloigne de la valeur de référence U, c'est-à-dire l'application de modificateurs positifs augmente la translation. On en déduit que $\gamma_{ik} > 1$ pour les deux propriétés.

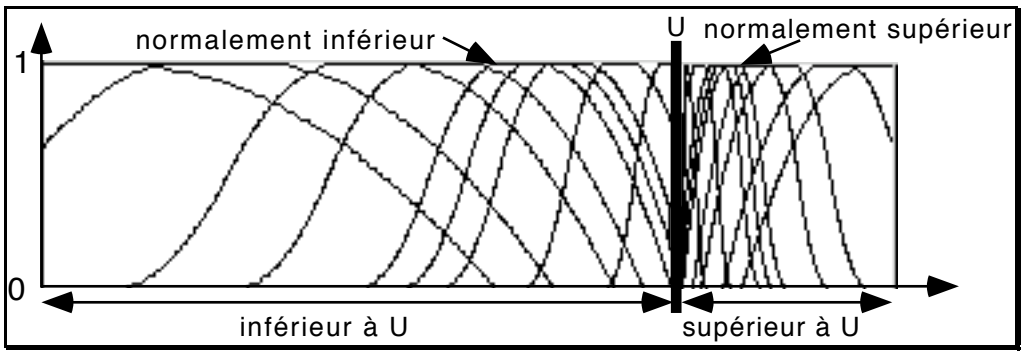

Figure 32. Formes attendues des propriétés issues des propriétés de base « supérieur à » et « inférieur à »

En posant des hypothèses et des contraintes similaires à celles utilisées pour calculer automatiquement les propriétés de base d'un domaine, nous pouvons proposer un calcul automatique de ces propriétés paramétrées. Ce calcul, les hypothèses et les critères de construction sont détaillés en Annexe 3. Les résultats en sont les coefficients figurant dans le tableau qui suit. Dans la pratique, ils permettent d'obtenir des propriétés de comparaisons élémentaires satisfaisantes pour la plupart des cas.

Soit $D_i=[Bm, BM]_u$ le domaine, et $D'_i=[Bm,U]_u$ et $D''_i=[U, BM]_u$ les pseudo-domaines engendrés par la valeur de référence U. On a alors δ la taille du domaine telle que $\delta = BM-Bm$ et δ' et δ'' les tailles des pseudo-domaines telles que $\delta' = U-Bm$ et $\delta'' = BM-U$. Nous proposons alors de définir les fonctions d'appartenance des propriétés de base comme indiqué dans le Tableau 5. Ces formules permettent d'obtenir les résultats de la Figure 33.

Tableau 5. Définition automatique des fonctions d'appartenance pour les propriétés de comparaison élémentaires

nom	α	a	b	β	τ_{ik}	γ_{ik}
supérieur à U	22% * δ''	U + 22% * δ''	U + 27% * δ''	23% * δ''	-4% * δ'	3
inférieur à U	23% * δ'	Bm + 73% * δ'	Bm + 78% * δ'	22% * δ'	4% * δ''	3

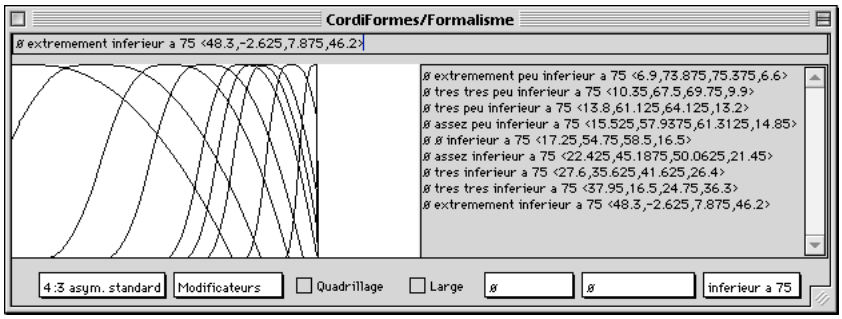

Figure 33. Modificateurs sur la propriété « inférieur à 75 »

Remarques :

- Le coefficient de translation élémentaire est proportionnel à la taille du pseudo-domaine.
- Comme dans le paragraphe 7, on retrouve le besoin d'ajouter un filtre pour éviter de « déborder » en appliquant certains opérateurs (en particulier « extrêmement peu »).

Ces nouvelles constructions automatiques permettent de proposer une nouvelle méthode pour construire les propriétés de base d'un concept en posant : « faible » ≡ « inférieur à M » et « important » ≡ « supérieur à M » avec M la valeur moyenne (Figure 34 et Figure 35). Ces propriétés ont la caractéristique d'être définies chacune sur une moitié du domaine (comme celles définies au paragraphe 8).

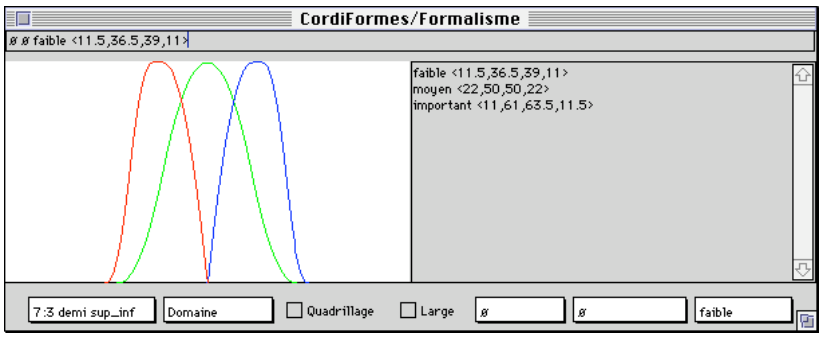

Figure 34. Domaine à base de « supérieur à » et « inférieur à »

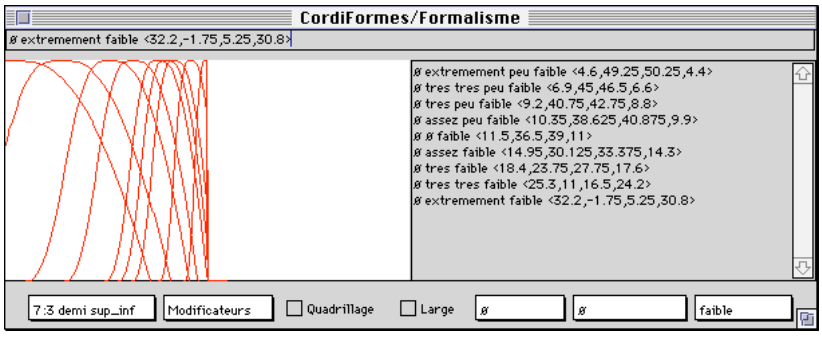

Figure 35. Application des modificateurs sur « faible » comme « inférieur à »

9. Conclusion

Dans ce chapitre, nous avons montré comment formaliser, traiter et manipuler les propriétés élémentaires, paramétrées ou non. Ces propriétés sont construites à partir de propriétés de base, sous-ensembles flous de l'ensemble des valeurs possibles du concept. Sur ces propriétés de base, nous avons appliqué deux types d'opérateurs génériques : les modificateurs (ou opérateurs de modification) ajustant la sémantique de la propriété et les opérateurs flous modifiant son imprécision. Nous avons aussi proposé des méthodes pour construire automatiquement les propriétés de base et déterminer la valeur des différents coefficients utilisés. Rappelons que les valeurs calculées sont éventuellement à modifier en fonction du concept et du domaine d'application.

Le modèle présenté semble indépendant du domaine d'application et uniforme quant au traitement des différentes catégories de propriétés élémentaires (même si les propriétés élémentaires paramétrées demandent parfois un traitement un peu spécifique).

Les propriétés élémentaires présentées ici sont des énoncés à la forme affirmative. Dans le prochain chapitre, nous étudierons le cas où ils sont à la forme négative.

CHAPITRE I.4 : LA NÉGATION D'UNE PROPRIÉTÉ ÉLÉMENTAIRE

1. Introduction

Alors que dans le chapitre précédent nous avions des énoncés affirmatifs, nous allons désormais nous intéresser à l'interprétation qu'un locuteur souhaite donner à un énoncé comme « *Le nombre d'intersections n'est pas très faible* », soit, de façon plus formelle, un énoncé du type « x n'est pas A » où A est une propriété imprécise ou vague. En effet, au cours d'une description, le locuteur est parfois amené à donner un énoncé négatif, et ceci pour plusieurs raisons, notamment :

- par effet de style, sous-entendant une affirmation, pour donner plus de force à sa description (litote…) ;
- pour souligner une propriété qu'il ne veut surtout pas ;
- par approximation parce qu'il ne maîtrise pas encore suffisamment l'image mentale qu'il a de la scène…

Pour les linguistes, « ne pas être A » n'équivaut pas, en général, à la propriété logique « être non-A » qui en termes de sous-ensembles flous est définie à partir de A. Ceci résulte du fait que la propriété logique « non-A » n'a pas, en général, de traduction dans le langage naturel. Le locuteur veut ainsi traduire le fait qu'un objet satisfait à un certain degré une propriété P autre que A mais du même domaine. Pour notre locuteur l'exemple initial peut signifier que « *Le nombre d'intersections est moyen* » (Figure 36). En d'autres termes, un énoncé de négation équivaut alors à une affirmation se référant à une autre propriété élémentaire. Cependant, il arrive que le locuteur se contente de fournir une négation sans arrière-pensée quant à sa signification. L'énoncé basé sur « …n'est pas… » est alors interprété comme « …est tout sauf… ». Notre étude étant principalement consacrée aux propriétés élémentaires, l'objectif visé est donc la recherche de l'énoncé « x est P » considéré par notre locuteur comme implicitement équivalent à l'énoncé « x n'est pas A ».

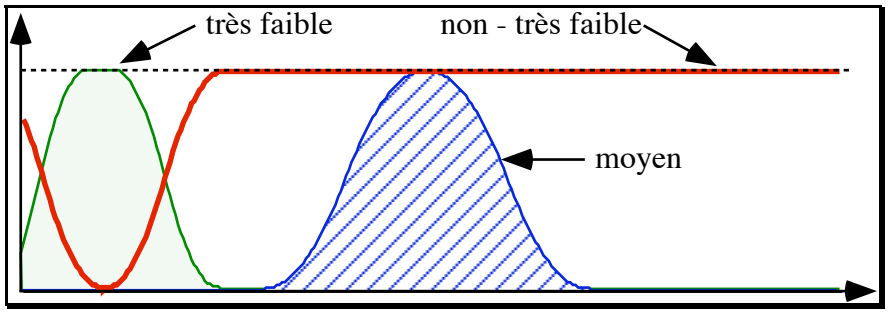

Figure 36. Interprétations possibles de « x n'est pas très faible »

Nous allons donc dans un premier temps rappeler ce qu'est la négation logique (section 2) puis, après avoir présenté la notion d'interprétation linguistique d'une propriété (section 3), nous introduirons la négation linguistique (section 4). Dans le cadre du flou, plusieurs études ont déjà abordé ce problème particulier dans un contexte plus formel ([Tor96] et [Pac97]). L'approche plus pragmatique présentée ici et dans [DeP97a], [PaD97a], [PaD97b] et [DeP97b] s'en différencie en ce sens qu'elle reste placée dans le formalisme envisagé. Ensuite, nous étudierons une première stratégie de choix d'une négation plausible à l'aide de travaux linguistiques basés, en particulier, sur la notion de marquage (section 5). Puis nous présenterons notre méthode basée sur la notion de similarité (section 6). Enfin, nous étudierons le problème particulier de la négation des propriétés paramétrées (section 7).

2. La négation logique d'une propriété élémentaire

Définition I.4.1 : La *négation logique* d'un énoncé du type « x est A », où A est une propriété élémentaire, est l'énoncé « x est non-A », où « non » correspond à l'opérateur de complémentation dans la théorie des ensembles flous. Par abus de langage, on le traduit encore sous la forme « x n'est pas A ».

Du point de vue de la fonction d'appartenance, on sait qu'on a alors ([Zad65]) :

$$\mu_{\text{« x est non-A »}} = 1 - \mu_{\text{« x est A »}} \equiv \mu_{\text{« x n'est pas A »}}.$$

Remarque : La gestion de la négation est très peu étudiée en modélisation déclarative. Seuls [Col90] et [Chau94b] pour les ensembles classiques puis [Des96] pour les ensembles flous ont traité ce sujet en utilisant la négation logique.

Dans la Figure 36, « non-très faible » est la négation logique de « très faible ». D'un point de vue linguistique, la propriété « x est non-très faible » n'a pas de sens. Est-ce que « x n'est pas très faible » ne signifierait pas plutôt « x est faible », « x est extrêmement important » ou « x est moyen » comme le suggère la figure ? Si c'est le cas, du point de vue de la génération des scènes solutions, utiliser le complémentaire « non-très faible » est alors très coûteux. L'emploi de « moyen » permettrait d'explorer moins de valeurs et de proposer des solutions qui ont plus de chance de plaire au locuteur.

Du point de vue de la linguistique et de la génération des solutions, le traitement de la négation par cette méthode n'est donc pas satisfaisant. Nous allons donc étudier une autre manière de traiter cette négation à travers l'utilisation de notions linguistiques.

3. L'interprétation linguistique d'une propriété élémentaire

Définition I.4.2 : *L'interprétation linguistique* d'un énoncé du type « x est A » (Figure 37, fonctions plus claires), où A est une propriété élémentaire, est souvent considérée par les linguistes ([Sch81]) comme la *disjonction des propriétés* :

- « x est $\varnothing_f$ A » ;
- « x est vraiment A » ;
- « x est approximativement A ».

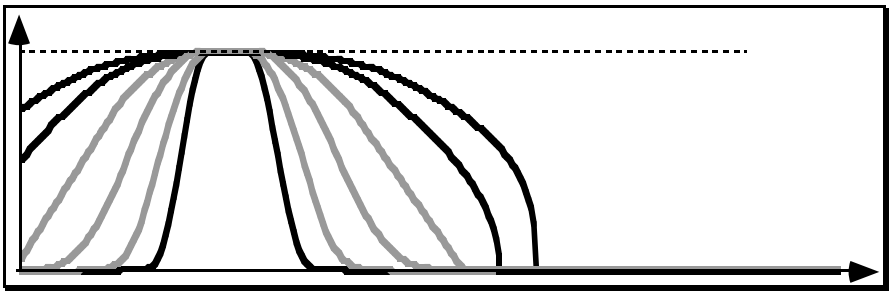

Figure 37. Interprétations linguistiques « fortement similaires » à « x est A »

L'ensemble des *opérateurs flous* se décompose pratiquement en deux ensembles G_1 et G_2 tels que :

- $F_Q = G_1 \cup G_2$;
- G_1 est l'ensemble des opérateurs flous « sous-entendus » par l'énoncé « x est A » ;
- G_2 est le complémentaire de G_1 dans F_Q.

Il en résulte alors que : « x est A » $\Leftrightarrow$ { « x est f_α A » avec $f_\alpha \in G_1$ }.

Par exemple, dans F_6, on obtient : G_1 = {approximativement, vraiment, $\varnothing_f$} et G_2 = {précisément, plus ou moins, vaguement}. Dans ces conditions, l'énoncé « x est faible » a pour traduction linguistique « x est approximativement faible » ou « x est vraiment faible » ou encore « x est $\varnothing_f$ faible ».

4. La négation linguistique d'une propriété élémentaire

Définition I.4.3 : La *négation linguistique* d'un énoncé du type « x est A » dans un domaine donné, notée « x n'est pas A », signifie déjà qu'on rejette toutes les interprétations de « x est A ». On fait ensuite référence à une autre des propriétés élémentaires P de ce même domaine pour signifier que « x n'est pas A » équivaut à « x est P ».

En utilisant les concepts introduits précédemment et en se référant à l'interprétation linguistique donnée à l'énoncé « x est A », *nier* « x est A » dans un domaine donné, signifie que,

tout en excluant les énoncés « x est f_α A » avec $f_\alpha \in G_1$, on retient comme interprétation de l'énoncé « x n'est pas A » *un des énoncés suivants* :

- « x est f_χ B » avec B une propriété élémentaire du même domaine que A, avec B ≠ A (en particulier, la propriété de base est différente) et $f_\chi \in F_Q$ *ou*
- « x est f_β A » avec $f_\beta \in G_2$ *ou enfin*
- « x est m_δ A » avec $m_\delta \neq \varnothing$ et compatible avec les modificateurs de A.

Exemple : Si pour le concept du nombre d'intersections des segments dans le plan, on a les propriétés {faible, moyen, important}, quelques interprétations plausibles de « x n'est pas faible » peuvent être « x est très important » ou « x est vraiment moyen » ou « x est extrêmement faible » ou encore « x est précisément extrêmement faible »…

Remarques :

- Le nombre de propriétés simples susceptibles d'être une interprétation de « x n'est pas A » semble être élevé. Cependant, nous avons déjà remarqué que le nombre de combinaisons d'opérateurs flous et de modificateurs est limité. De plus, des stratégies donnant la direction vers laquelle s'oriente la négation seront appliquées (§5 et §6.3.2). Pour que l'utilisateur accède rapidement à son idée, les solutions seront ordonnées et présentées par ordre croissant de complexité pour respecter un principe de simplicité dans les réponses (§6.3.3). Enfin, un filtrage sera aussi introduit pour ne retenir que des combinaisons pertinentes (§6.3.4).
- La réunion des supports des solutions possibles recouvre souvent celui de l'énoncé « x est non-A ». Par conséquent, l'utilisation de la négation logique pour traduire simplement la négation linguistique d'une propriété élémentaire, peut se justifier même si elle ne reçoit pas de signification sémantique dans un contexte d'imprécision des concepts. Elle donne un sens à la réunion des solutions sans fournir le détail de ses composants et s'approche plus de l'interprétation de « n'est pas » comme étant « est tout sauf ». En réalité, la propriété « tout sauf » sera traduite par le complémentaire du support de la propriété niée. Ainsi, aucune valeur de A ne pourra être solution. Ce traitement semble le plus proche de l'idée de cette propriété.

5. Stratégie d'orientation de « n'est pas »

Toute la difficulté réside dans l'ambiguïté liée à la négation qui, par définition, ne précise pas explicitement la propriété P à laquelle on se réfère. Il est alors difficile de *préciser la direction dans laquelle s'oriente* « n'est pas ». *Une stratégie de recherche utilisant la notion de similarité ou le voisinage* [Pac92] peut être utilisée pour obtenir cette information dans le cas général. Nous aborderons ce thème dans la section suivante. Nous pouvons, indépendamment de cette stratégie, apporter cependant quelques éléments de réponse dans les cas les plus sim-

ples. En effet, hormis les propriétés paramétrées qui sont généralement en nombre infini et donc pratiquement inexploitables, on peut noter que les propriétés d'un concept sont assez peu nombreuses. Examinons les trois cas les plus courants, c'est-à-dire les concepts avec une, deux ou trois propriétés précises, donc de nature booléenne, dans un contexte bien défini.

5.1. Concepts avec une seule propriété

Nous sommes dans le cas où pour un concept C_i, nous avons seulement $P_i=\{P_{i1}\}$. Souvent, la négation peut être nommée en ajoutant au nom de la propriété un des préfixes suivant ([Mul91]) : « im », « in », « dé », « non- », « pas- »... Ceci signifie que la négation linguistique est, en réalité, une affirmation reposant sur une nouvelle propriété qui peut être associée au concept. Cette propriété est souvent caractérisée avec le complémentaire de la propriété d'origine. Par conséquent, P_i s'écrit implicitement $P_i=\{P_{i1}, P_{i2}\}$ avec $P_{i2} \equiv$ Non-P_{i1}. Le traitement de « x n'est pas P_{1i} » donne « x est non-P_{1i} » et réciproquement. Par conséquent, un concept avec apparemment une seule propriété est, en réalité, un concept à deux propriétés.

Exemple : Pour décrire l'organisation des segments dans le plan, on peut définir un concept de densité dont le domaine ne contient à priori que la propriété « dense ». En fait, il existe aussi une propriété qui peut être nommée « pas-dense » ou « non-dense ».

5.2. Concepts avec deux propriétés

Dans certains cas, la direction peut être trouvée à l'aide des notions de *marquage* et *d'antonyme* ([DuS95], [Mul91] et [Hor89]) présentées au chapitre I.1 (§3.4). Supposons qu'un concept C_i soit représenté par deux propriétés $P_i=\{P_{i1}, P_{i2}\}$. Si on suppose que P_{i2} est marqué ou est l'antonyme négatif de P_{i1}, la négation de P_{i1} est alors équivalente à P_{i2}. En ce qui concerne la négation de P_{i2}, on peut utiliser le traitement classique, car la relation n'est pas symétrique. Mais, on peut malgré tout exploiter le marquage et considérer que la négation est alors « dirigée » vers P_{i1} (on retrouve cette idée dans [DuS95] et [Mul91]). On en déduit alors un traitement presque classique à ceci près que les propriétés associées à la négation se trouvent seulement du côté de P_{i1} (Figure 38) par rapport à P_{i2}.

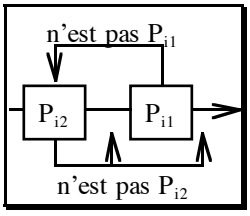

Figure 38. Négation pour un couple de propriétés marquées

Notons que cette solution n'est possible que si on introduit la notion de gradation des propriétés du concept ([DuS95]). Cela revient à supposer une relation d'ordre total dans P_i, c'est-à-dire : $P_{i\alpha} \leq P_{i\beta} \Leftrightarrow \alpha \leq \beta$.

5.3. Concepts avec trois propriétés

Supposons qu'un concept C_i soit représenté par trois propriétés $P_i=\{P_{i1}, P_{i2}, P_{i3}\}$. Si on arrive à déterminer un couple d'antonymes, on les traite comme au paragraphe précédent. La troisième propriété est alors traitée d'une manière standard.

Exemple : Dans le domaine concernant le nombre de segments dans le plan, on définit aisément le triplet de propriétés suivant {faible, moyen, important}. Si on précise que « faible » est l'élément marqué négativement nous avons « n'est pas important » équivalent à « est faible ». Par contre, « n'est pas faible » est une propriété, probablement différente de « est important » mais du même côté par rapport à « est faible ».

6. La négation via la similarité

Une stratégie de recherche utilisant la notion de similarité des sous-ensembles flous ([Tve77], [DuP80b], [Zad87], [ZCB87], [Pac92], [DiK94]) peut être utilisée pour obtenir cette information dans le cas général. Cette méthode est présentée en détail dans [DeP97a], [Pac97], [PaD97a], [PaD97b] et [DeP97b]. Elle permet, pour des propriétés imprécises, de déterminer et d'ordonner l'ensemble des propriétés plausibles comme négation.

6.1. Rappels

Lorsqu'un locuteur énonce une négation :

- il refuse les interprétations linguistiques de la propriété niée ;
- il fait référence à une propriété élémentaire du même domaine.

La recherche de la propriété implicite liée à une négation s'effectue en quatre étapes :

1. la génération des propriétés possibles ;
2. la sélection des propriétés plausibles (à l'aide de la notion de similarité) ;
3. l'ordonnancement de ces propriétés ;
4. le choix effectif.

Nous allons étudier la méthode de sélection des propriétés plausibles puis le critère d'ordonnancement le plus simple.

6.2. Détermination des propriétés plausibles

Les propriétés plausibles pour la négation linguistique de « X est A » sont "calculées", filtrées, à partir de l'ensemble des propriétés possibles grâce à la notion de similarité de deux propriétés selon [Pac92]. Ce choix a été effectué de façon arbitraire. Cette méthode demanderait à être évaluée vis-à-vis d'autres méthodes de calcul.

6.2.1 Voisinage de deux ensembles flous

Le voisinage de deux valeurs est déterminé à partir de l'implication de Lukasiewicz. Cette relation est définie sur [0,1]x[0,1] et à valeurs dans [0,1] telle que :

$$u \rightarrow_L v = 1 \text{ si } u \leq v \text{ sinon } 1 - u + v.$$

Définition I.4.4 : La relation $\mho_\alpha$ de *voisinage à un degré* $\alpha \in [0,1]$ de deux valeurs u et v est une fonction dans [0,1] définie de la manière suivante :

u est *voisin au degré* α de v $\Leftrightarrow$ u est α-*voisin* de v $\Leftrightarrow$ u $\mho_\alpha$ v $\Leftrightarrow$ Min $\{u \rightarrow_L v, v \rightarrow_L u\} \geq \alpha$.

La Figure 39 illustre cette notion. Dans cette figure, le blanc représente l'absence de voisinage et le noir le voisinage total (identité). A partir de cette notion, nous définissons le voisinage de deux sous-ensembles flous.

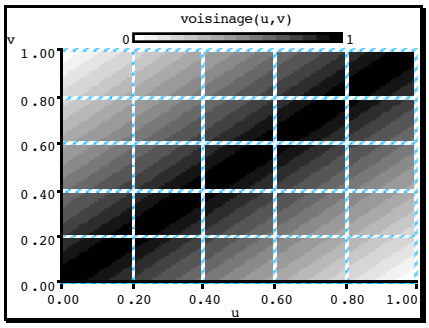

Figure 39. Voisinage à un degré α de deux valeurs

Définition I.4.5 : Étant donnés deux sous-ensembles flous A et B, A est *voisin au degré* α de B, ou encore α-*voisin* de B, si et seulement si l'on a :

A est *voisin au degré* α de B $\Leftrightarrow$ A est α-*voisin* de B $\Leftrightarrow$ A $\approx_\alpha$ B $\Leftrightarrow \forall x, \mu_A(x) \mho_\alpha \mu_B(x)$.

6.2.2 Similarité symbolique

Afin de définir la similarité des ensembles flous, nous utilisons une partition *I* de l'intervalle [0,1]:

$I = \{I_1, ..., I_7\} = [0] \cup]0,0.25] \cup]0.25,0.33] \cup]0.33,0.67[\cup [0.67,0.75[\cup [0.75,1[\cup [1]$.

Nous avons réalisé une bijection de *I* avec l'ensemble totalement ordonné θ de termes linguistiques :

$\theta = \{\theta_1, ..., \theta_7\}$={pas du tout, très peu, assez peu, moyennement (∅), assez, très, tout à fait}.

Définition I.4.6 : Étant donnés deux sous-ensembles flous A et B, ils sont dits θ_i *similaires* si et seulement si : $\theta_i \in \theta$, $I_i \in I$ est tel que $\alpha \in I_i$, sachant que $\alpha = \text{Max}\ \{\delta \mid A \approx_\delta B\}$.

Remarque : Lorsqu'il s'agit de similarité le terme « faiblement » recouve les termes « pas du tout », « très peu » ou « assez peu ». De même, le terme « fortement » recouvre « moyennement », « assez », « très » ou « tout à fait ».

La Figure 40 illustre le calcul de cette similarité. A partir des deux ensembles flous associés aux propriétés A et P, on détermine la courbe de voisinage local définie par le α-voisinage maximum des degrés d'appartenance pour chaque valeur du domaine (courbe épaisse). Cette courbe donne le α-voisinage des deux ensembles flous. Les deux propriétés A et P sont, dans cet exemple, au plus 0,37-voisines. Finalement, la similarité symbolique est donnée par l'échelle permettant d'associer au voisinage maximum la similarité symbolique (flèche pointillée). Les deux propriétés A et P sont « normalement similaires » et donc considérées comme « fortement similaires ».

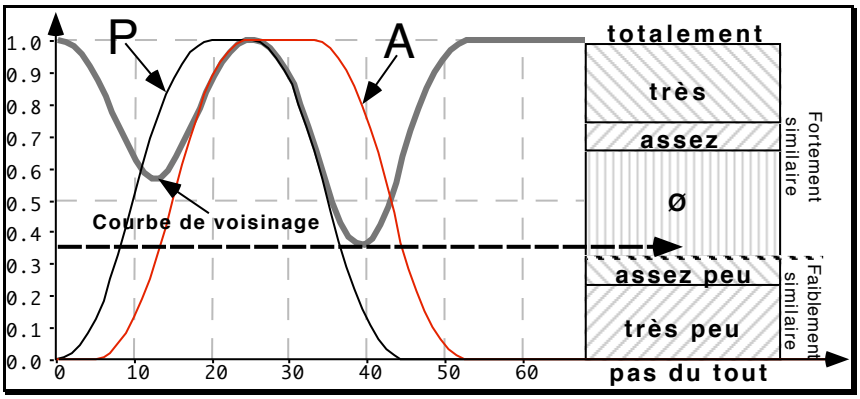

Figure 40. Similarité symbolique de deux propriétés

Remarque : L'énoncé « x est A » recouvre linguistiquement les énoncés « x est f_αA » avec $f_\alpha \in G_1$. On vérifie aisément que les ensembles flous associés aux « f_αA », $f_\alpha \in G_1$, sont *fortement similaires* à A (en clair dans la Figure 37). De même, ceux avec $f_\alpha \in G_2$ sont *faiblement similaires à* A.

6.2.3 Similarité et propriétés plausibles

Intuitivement, nous pouvons dire qu'une propriété P peut être sélectionnée comme propriété plausible pour la négation linguistique de A si elle est « faiblement similaire ». Cependant, ce critère n'est pas suffisant. En effet, dans la Figure 41, P et A (faiblement similaires) sont telles que lorsque les valeurs de l'une sont voisines de 1, les valeurs de l'autre ne sont pas voisines de 0. Elles sont donc encore trop « similaires ». Il est donc nécessaire d'introduire un autre critère permettant de rejeter les propriétés ayant un fort voisinage pour les degrés d'appartenance les plus significatifs.

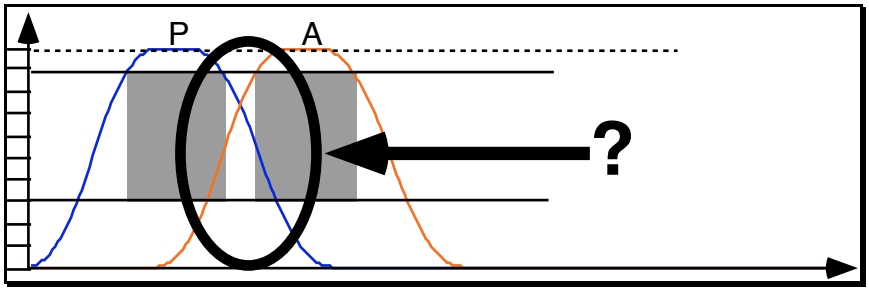

Figure 41. Nécessité de critères supplémentaires

Définition I.4.7 : Soit ρ un réel tel que $0.33 \geq \rho \geq 0$. La propriété élémentaire P, définie sur le même domaine que la propriété niée A, est dite propriété ρ-plausible pour la négation linguistique de A si elle satisfait les conditions suivantes :

1. P et A sont *faiblement similaires*, c'est-à-dire θ_i-similaires avec $\theta_i <$ moyennement ;
2. $\forall x, ((\mu_A(x) \geq 0.67+\rho) \Rightarrow (\mu_P(x) \leq \mu_A(x)-0.67))$;
3. $\forall x, ((\mu_P(x) \geq 0.67+\rho) \Rightarrow (\mu_A(x) \leq \mu_P(x)-0.67))$.

L'expérimentation montre que seulement à partir du seuil $\rho=0.3$ ce filtrage propose comme négations plausibles de « X n'est pas faible » des propriétés dérivées de « faible » en plus de nombreuses nuances des propriétés « moyen » et « important », ce qui semble conforme à ce que peut en attendre l'intuition humaine ([DeP97a]). Par conséquent, P est une propriété plausible pour la négation linguistique de A si elle est 0.3-plausible.

Remarque : Ces deux critères supplémentaires semblent suffisants pour sélectionner, parmi les propriétés « faiblement similaires », les propriétés plausibles (Figure 42 : les deux propriétés P vérifient l'ensemble des critères par rapport à A).

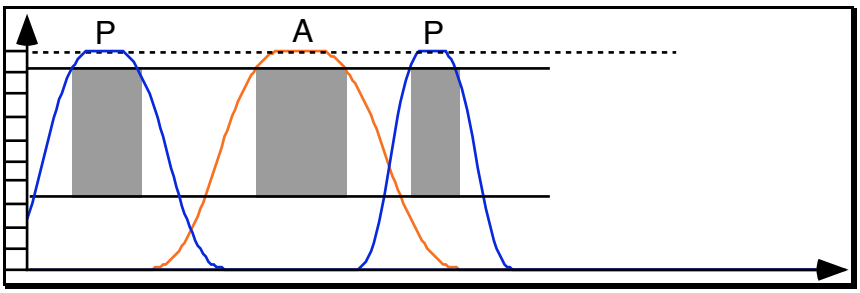

Figure 42. Faible similarité aux valeurs significatives

La détermination des propriétés plausibles est donc effectuée selon les critères suivants :

1. Rejet des interprétations implicites de l'énoncé « X est A » (Cf. §3) ;
2. Sélection des propriétés faiblement similaires à celles rejetées en 1 ;
3. Sélection des propriétés au faible voisinage local par rapport à celles rejetées en 1 pour les degrés d'appartenance les plus significatifs (en particulier ceux proches de 1).

6.3. Ordonnancement des propriétés plausibles

6.3.1 Principe

Le nombre de propriétés sélectionnées par la méthode que nous venons de présenter est très souvent important (Figure 43). En effet, notre formalisme permet d'effectuer un grand nombre de combinaisons d'opérateurs et de modificateurs. Pour que l'utilisateur choisisse rapidement la propriété la plus adaptée, il convient donc d'ordonner les propriétés et, éventuellement, d'effectuer une nouvelle sélection. L'ordonnancement des propriétés plausibles suit les deux règles suivantes :

1. *Principe de simplicité*. Un locuteur pense plutôt à une propriété construite de façon simple à partir d'une propriété du domaine. En pratique, les opérateurs utilisés sont peu nombreux, c'est-à-dire l'opérateur Ø est le plus souvent présent.
2. *Principe de préférence*. Le locuteur pense plus souvent à une propriété dont la propriété de base est différente de A et, de préférence, signée.

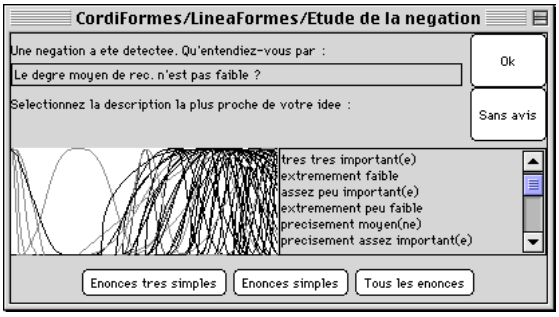

Figure 43. Totalité des propriétés plausibles

6.3.2 Utilisation du marquage

Pour affiner l'ordonnancement de ces propriétés plausibles, il est intéressant d'utiliser les notions linguistiques de marquage présentées au chapitre I.1, paragraphe 3.6. Notons que, selon la méthode exposée, il y a élimination de certaines propriétés (voire toutes sauf une). Ici, nous n'allons pas « éliminer » les propriétés mais seulement faire en sorte qu'elles n'apparaissent qu'en dernier recours. L'ordonnancement présenté au paragraphe précédent sera donc partiellement remis en cause lorsque la propriété de base de la propriété niée fera partie d'un couple de termes marqués. L'ordre des propriétés plausibles sera modifié selon les règles suivantes :

- Lorsque la propriété de base de la propriété niée correspond au terme non-marqué, la propriété plausible dont la propriété de base est le terme marqué et dont les opérateurs sont ceux par défaut est proposée en premier (Figure 44, droite).
- Lorsque la propriété de base de la propriété niée correspond au terme marqué, les propriétés plausibles, dont la propriété de base est le terme non-marqué, ne sont proposées qu'en dernier recours (Figure 44, gauche).

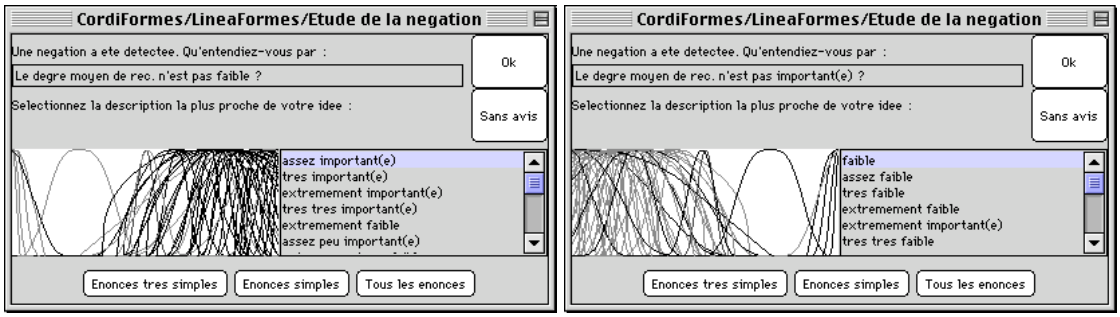

Figure 44. Négation de la propriété marquée et de la propriété non-marquée

6.3.3 Bilan sur la méthode d'ordonnancement

Pour permettre l'ordonnancement des propriétés plausibles, nous allons définir les notions de complexité linguistique des opérateurs et d'une propriété élémentaire.

Définition I.4.8 : La *complexité linguistique d'un opérateur* est une fonction entière cl telle que cl(« Ø ») = 0 et pour tout opérateur "op", cl(op) est le nombre de mots de cet opérateur.

Cependant, nous allons poser le cas particulier suivant : sachant qu'il est naturellement plus facile de donner une description « positive », les opérateurs négatifs seront pénalisés par rapport aux modificateurs positifs. Nous aurons par exemple : $cl_{modificateur}(m_\alpha) = cl(m_\alpha)+1$ si signe(m_α) = -1 et $cl_{modificateur}(m_\alpha)= cl\ (m_\alpha)$ sinon.

Définition I.4.9 : La *complexité linguistique d'une propriété élémentaire* est une fonction entière CL tel que : $CL(« f_\alpha\ m_\beta\ P_{ik} ») = cl(m_\beta) + dc * cl(f_\alpha)$ avec dc un réel.

Cette définition prend en compte le fait que, naturellement, le locuteur donnera plus facilement une propriété avec seulement un modificateur ou bien un opérateur flou (toujours selon le principe de simplicité). De plus, elle considère que l'utilisation d'un opérateur flou est plus complexe que celui d'un modificateur. Le coefficient dc permet de rendre compte de cette différence de complexité. Dans la pratique, 3 semble une valeur satisfaisante pour dc. Nous pouvons maintenant introduire la notion de valeur linguistique d'une propriété P par rapport à la propriété niée A.

Définition I.4.10 : La *valeur linguistique* d'une propriété élémentaire P relativement à la propriété niée A est fonction de la complexité linguistique de P ajustée par les notions de marquage et les principes de simplicité et de préférence.

Les propriétés plausibles sont alors triées selon leur valeur linguistique croissante. L'Algorithme 1 propose le calcul de la valeur linguistique de cette propriété P lorsque A est niée.

Algorithme 1. Valeur d'une propriété

```
Algorithme Valeur_Linguistique( P = « f'α m'β P'ik » ↓,
                                A = « fα mβ Pik » ↓) retourne Entier
 V ← 2 * CL(P) //Princ. de simplicité
 VA ← 2 * CL(A)
 Si A = non-marquée et p = marquée alors
   //solution principale à la négation de la propriété non-marquée
   V ← -1
 Sinon
   Si A = marquée alors
     Si P'ik = non-marquée et m'β = « Ø » alors
       //Il faut éviter les prop. non-marquées
       //comme négation de la prop. marquée
       V ← V + 50
     Sinon
       Si P n'est pas du même côte que non-marquée par rapport à A alors
         //Eviter aussi les prop. qui ne sont par du côte
         //de la non-marquée
         V ← 2 * V
       Fin Si
     Fin Si
   Fin Si
   Si P'ik = Pik alors //Princ. de préférence
     //On défavorise un peu si la propriété de base
     //est la même de celle de A
     V ← V + 1
   Fin Si
   Si Signe(P'ik) = 0 alors //Princ. de préférence
     //On défavorise encore plus les propriétés non modifiables
     V ← V + 2
   Fin Si
 Fin Si
 Retourne V
Fin Algorithme Valeur_Linguistique
```

6.3.4 Suppression des énoncés complexes

Les propriétés étant construites de façon automatique, il n'est donc pas rare d'obtenir des propriétés n'ayant pas de sens dans le langage naturel. Certaines sont beaucoup trop complexes pour avoir une chance d'être retenues. Par exemple, on peut trouver des propriétés de la forme « *Le nombre de segments est vaguement très très peu faible* ». Évidemment, ce sont souvent les propriétés très « compliquées », c'est-à-dire comportant de nombreux modificateurs et ayant une valeur linguistique élevée. Il serait donc intéressant de mettre au point une méthode de classement et surtout de sélection pour en éliminer une grande partie.

Une solution possible consiste à éviter d'utiliser certains opérateurs rarement utilisés ou trop compliqués. En particulier, les opérateurs suivants provoquent souvent des propriétés élémentaires trop complexes : « très très », « très très peu », « extrêmement peu », « plus ou moins »… Ils sont très souvent utilisés seuls (avec l'opérateur par défaut « Ø »). On ne doit

donc les proposer qu'en dernier recours (peut-être même seulement sur demande de l'utilisateur avec, par exemple, un bouton « plus de propriétés »).

En complément de cette solution, et toujours dans l'objectif de respecter le principe de simplicité, il est possible de limiter le nombre de mots utilisés pour caractériser une propriété de base. Il est raisonnable de penser qu'au delà de deux mots qualificatifs, la propriété devient trop complexe pour être manipulée facilement par le locuteur.

Pour simplifier les recherches, le concepteur doit donc pouvoir ne sélectionner que certaines classes de propriétés. Ces sélections sont basées sur le principe de simplicité. Nous pouvons poser trois niveaux de sélection en rapport avec ce principe :

1. la sélection globale où toutes les propriétés plausibles sont produites ;
2. la simplicité élargie où seules les propriétés plausibles possédant l'opérateur vide ou le modificateur vide sont prises en compte, c'est-à-dire les énoncés de la forme « x est Ø Ø P_{ik} », « x est f_α Ø P_{ik} » ou « x est Ø m_β P_{ik} » (le produit des complexités linguistiques des opérateurs est nul) ;
3. la simplicité restreinte où, parmi les propriétés plausibles sélectionnées selon la simplicité élargie, seules sont choisies celles ayant un ou deux mots maximum pour l'ensemble des opérateurs utilisés (le produit des complexités linguistiques des opérateurs est nul et la somme des complexités linguistiques des opérateurs est inférieure ou égale à 1 ou 2).

Exemples :

- Nous aurons dans le dernier niveau des propriétés comme « x est très important », « x est assez faible », « x est vraiment faible » ou « x est très très faible » mais nous n'aurons pas « x est très très peu faible », « x est plus ou moins très important »…
- La négation de « X est faible » a beaucoup plus de chance d'être « X est important » (sauf en cas de marquage, bien sûr) ou « X est extrêmement faible » plutôt que « X est plus ou moins très très peu important » !

La Figure 44 (gauche) propose l'ensemble des propriétés plausibles comme négation linguistique de « X est faible ». La Figure 45 propose seulement celles sélectionnées selon le principe de simplicité (Figure 45, gauche) puis de simplicité restreinte (Figure 45, droite). On constate que le nombre de propriétés proposées est nettement moins important. Le choix en est donc facilité.

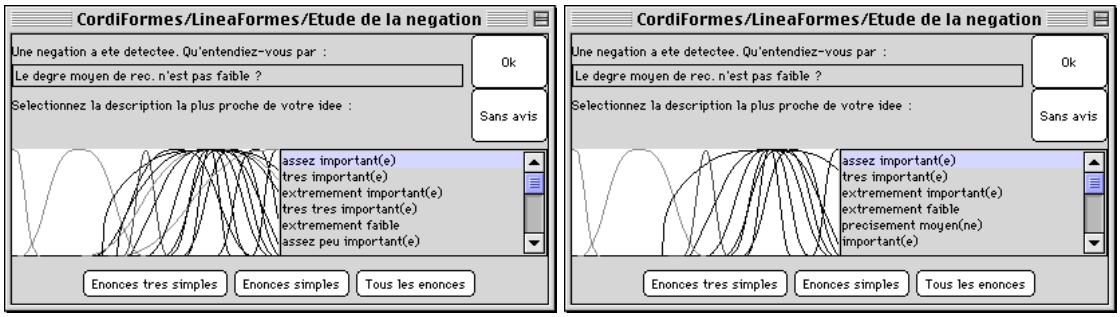

Figure 45. Propriétés plausibles selon la règle de simplicité et de simplicité restreinte

6.4. Algorithme de la négation

Finalement, nous pouvons proposer l'algorithme de gestion de la négation (algorithmes suivants) où :

- « Trier » est une procédure permettant d'arranger les propriétés plausibles selon leur valeur linguistique (voir définition I.4.10) ;
- « α-voisinage » est la fonction décrite au début du paragraphe 6.1.2 (Figure 39) ;
- « Grille » permet de faire l'équivalence entre le voisinage et la similarité symbolique (Figure 40) ;
- « Type_Similarité » est l'ensemble des valeurs symboliques possibles pour la similarité.

Algorithme 2. Gestion de la négation

```
Algorithme Gestion_Négation(A↓) retourne "f m P"
 Interprétations ← {f A : ∀f ∈G_i} ; Propriétés_plausibles ← {}
 Propriétés_possibles ← {f m P : ∀f ∈F et ∀m ∈M et ∀P ∈C_i}
 Pour tout P_i de Propriétés_possibles faire
   Pour tout A_i de Interprétations faire
     Si Est_Plausible(P_i, A_i) Alors
       Propriétés_plausibles ← Propriétés_plausibles + P_i
     Fin si
   Fin pour
 Fin pour
 Propriétés_proposées ← Filtre(Simplicité, Propriétés_plausibles)
 Retourner Selection_utilisateur(Trier(Propriétés_proposées))
Fin Algorithme Gestion_Négation
```

Algorithme 3. La propriété P est-elle une négation plausible ?

```
Algorithme Est_plausible (P↓, A↓) retourne Booléen
 Est_plausible ← FAUX ; Similarite ← Similarité_symbolique(P, A)
 Si Similarité ≤ « assez peu similaire » //Critère de base
   alors Si Faible_Similarité_Locale(P, A) //Critère supplémentaire
     alors Est_plausible ← VRAI
   Fin si
 Fin si
 Retourner Est_plausible
Fin Algorithme Est_plausible
```

Algorithme 4. Étude de la similarité symbolique de deux propriétés

```
Algorithme Similarité_symbolique(P↓, A↓) retourne Type_Similarité
 voisinage ← 1
 Pour toute valeur x du domaine faire
   voisinage ← Minimum(voisinage, α-voisinage(μP(x), μA(x)))
 Fin pour
 Retourner Grille(voisinage)
Fin Algorithme Similarité_symbolique
```

Algorithme 5. Étude de la similarité entre deux propriétés pour les valeurs significatives

```
Algorithme Faible_Similarité_Locale (P↓, A↓) retourne Booléen
 Ok ← VRAI
 ρ ← 0.3
 Pour toute valeur x du domaine ET tant que Ok faire
   Si μA(x)≥ ρ + 0.67
     Alors Ok ← μP(x) ≤ μA(x) - 0.67
   Fin Si
   Si μP(x)≥ ρ + 0.67
     Alors Ok ← Ok ET μA(x) ≤ μP(x) - 0.67
   Fin Si
 Fin pour
 Retourner Ok
Fin Algorithme Faible_Similarité_Locale
```

La Figure 46, la Figure 47, la Figure 48 et la Figure 49 proposent les propiétés élémentaires plausibles comme négation linguistique de la propriété élémentaire « x est très faible » dans différents cas d'organisation des propriétés de base du concept. Chaque ligne de la liste de propriétés commence par la valeur linguistique de cette propriété et termine par les coefficients de la fonction LR qui lui est associée.

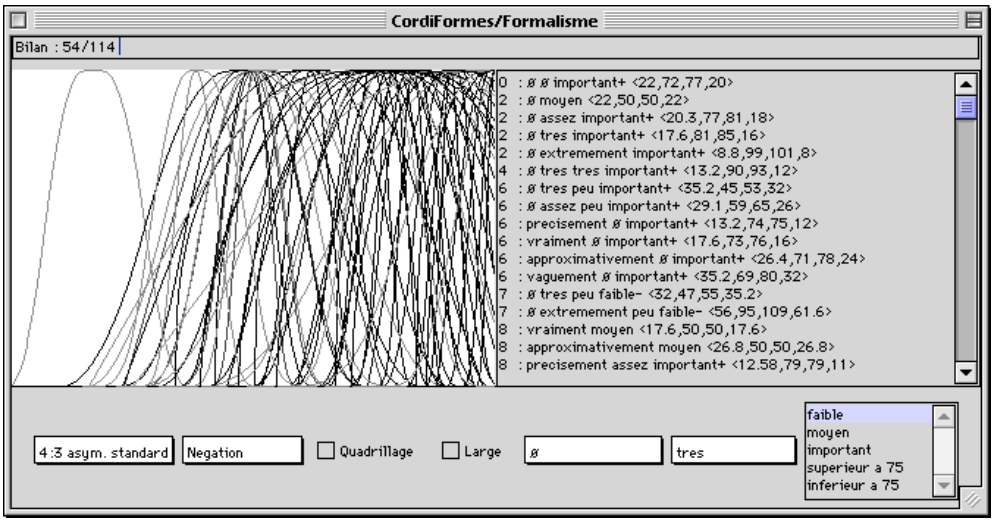

Figure 46. Négation avec des propriétés de base asymétriques sur tout le domaine

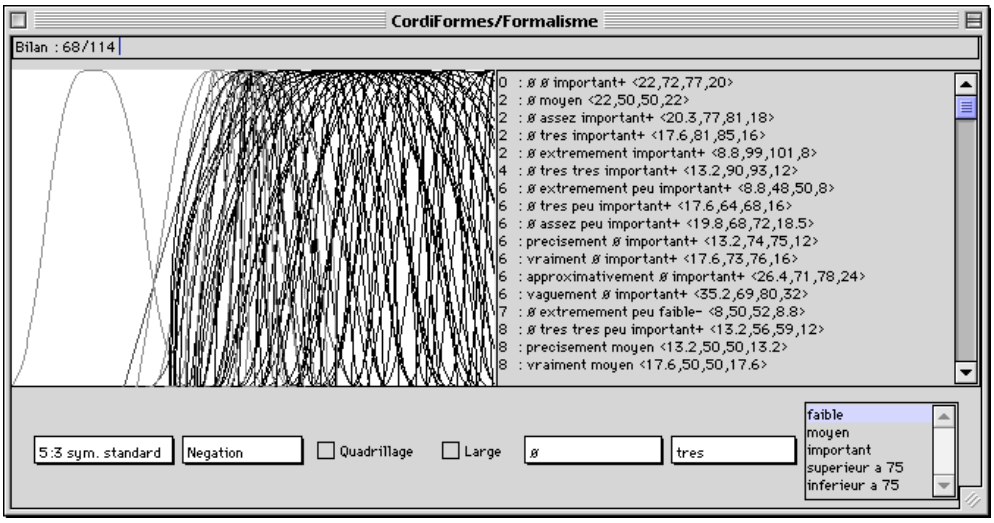

Figure 47. Négation avec des propriétés de base symétriques sur tout le domaine

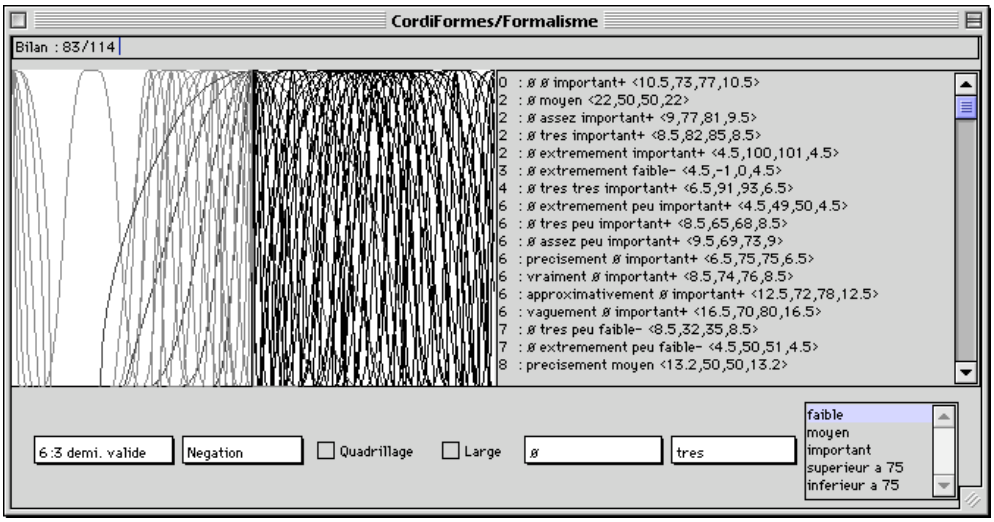

Figure 48. Négation avec des propriétés de base symétriques sur le demi-domaine

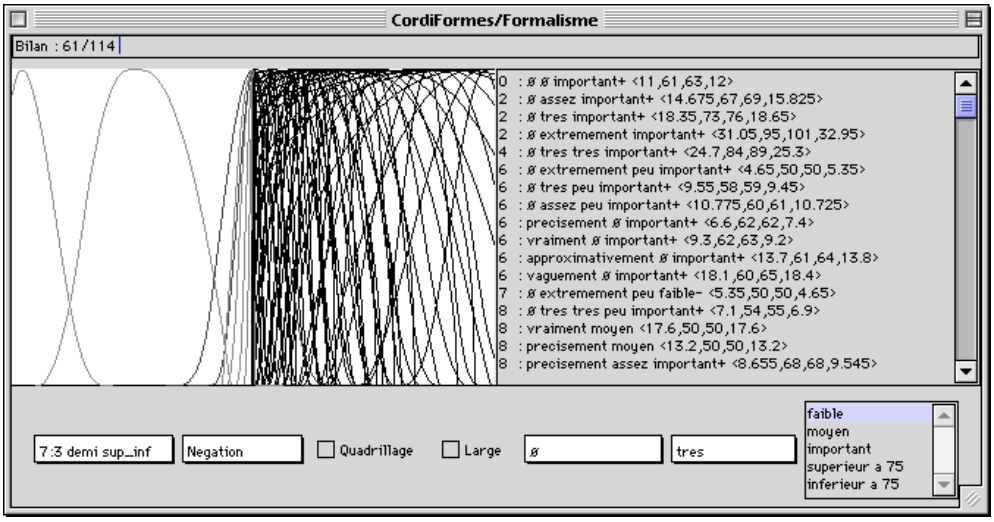

Figure 49. Négation avec des propriétés de base asymétriques sur le demi-domaine

7. Négation et propriétés paramétrées

Au paragraphe précédent, nous avons proposé une méthode permettant de gérer la négation d'une propriété élémentaire. Nous allons nous intéresser ici à la négation d'une propriété paramétrée. Selon la méthode retenue, nier une propriété de la forme « entre U et V » ne peut faire référence qu'à d'autres propriétés car, ne pouvant appliquer de modificateurs, toutes celles issues de cette propriété sont trop similaires. Les propriétés plausibles sont alors uniquement des propriétés élémentaires issues des propriétés de base du concept. Or lorsque le locuteur nie « vraiment entre 2 et 3 », il sous-entend éventuellement « supérieur à 3 » ou « entre 200 et 300 »… Il serait donc intéressant de pouvoir proposer des propriétés paramétrées. La même remarque peut être faite lorsqu'il énonce « n'est pas petit » en ayant en arrière pensée « entre 1m70 et 1m90 ». Nous allons donc nous intéresser à l'introduction des propriétés paramétrées dans la gestion de la négation linguistique.

7.1. Négation d'une propriété paramétrée

Notre objectif est d'introduire des propriétés paramétrées issues plus ou moins directement de la propriété niée comme propriétés plausibles. Considérons que la propriété niée est de la forme « x est f_α m_β P(U,V) » où P est « entre » (alors m_β = « Ø »), « supérieur à » ou « inférieur à », c'est-à-dire les trois formes possibles de propriétés paramétrées. Éventuellement, nous avons U = V.

7.1.1 Négation d'une propriété de comparaison élémentaire

La négation de « f_α m_β supérieur à U » (resp. de « f_α m_β inférieur à U ») proposera bien sûr des combinaisons à partir des propriétés de base mais aussi à partir de « supérieur à U » (resp. de « inférieur à U »), car celle-ci est modifiable. Il est aussi possible de prendre en compte la propriété « symétrique », c'est-à-dire « inférieur à U » si on nie « supérieur à U » et inversement.

7.1.2 Négation d'une propriété intervalle

Nous allons nous intéresser maintenant à la négation d'une propriété paramétrée de la forme « x est f_α entre U et V ».

Les propriétés les plus évidentes à introduire comme propriétés possibles sont les propriétés de comparaison élémentaires vues au chapitre précédent. Nous aurons alors des propriétés de la forme :

- « x est f_α m_β supérieur à U » ;
- « x est f_α m_β supérieur à V » ;
- « x est f_α m_β inférieur à U » ;
- « x est f_α m_β inférieur à V ».

Il reste maintenant à essayer de proposer des propriétés de la forme « x est f_α entre A et B ». La difficulté est de trouver des valeurs pour A et B sachant que le nombre de propriétés de cette forme est très souvent extrêmement important voire infini.

Une solution consiste à considérer la propriété « entre U et V » comme une propriété modifiable. Ainsi, nous allons construire des propriétés de la forme « x est f_α m_β entre U et V ». Il suffit ensuite de les renommer en fonction des nouvelles formes calculées à l'aide des opérateurs et de les rendre à nouveau non-modifiables. Pour ce traitement, il faut déterminer quel type de fonction utiliser. L'objectif étant de parcourir tout le domaine, la propriété doit être asymétrique sur l'ensemble du domaine. Le coefficient de translation élémentaire et le coefficient d'asymétrie sont alors calculés de la même manière que pour les propriétés de base (Chapitre I.3). Il convient juste de poser le signe de ces pseudo-propriétés de base. Arbitrairement, nous les considérerons positives.

Nous pourrions éventuellement ajouter à cette liste des propriétés paramétrées (de la forme « entre U et V ») issues :

- de la forme du domaine comme par exemple des découpages en intervalles réguliers fonction d'un pourcentage du domaine, de valeurs caractéristiques ou d'autres techniques similaires ;
- des supports ainsi que des noyaux des propriétés de comparaison élémentaires et des propriétés engendrées à partir des propriétés de base (Figure 50).

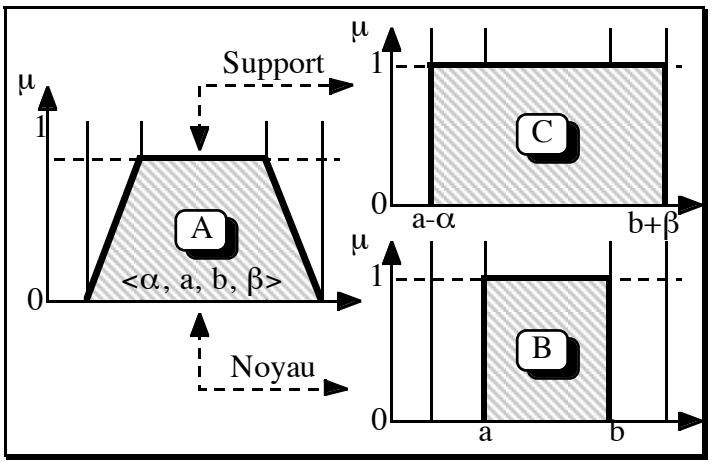

Figure 50. Passage d'un intervalle flou vers un intervalle classique

7.1.3 Bilan

Cependant, aussi bien pour la négation d'une propriété élémentaire paramétrée que pour celle d'une propriété élémentaire non-paramétrée, il ne faut pas non plus proposer trop de propriétés plausibles car l'utilisateur ne s'y retrouverait pas. Il faut aussi prendre en compte l'éventualité qu'il ne trouve pas de propriétés satisfaisantes.

Il aura alors le choix entre trois solutions :

- il prend une propriété relativement proche de ce qu'il pense ;
- il reprend sa description en remplaçant sa négation par la propriété qu'il sous-entend ;
- il considère que sa négation est, en réalité, la traduction du « tout sauf » amenant à une fonction d'appartenance proche du complémentaire logique (comme nous l'avons vu au paragraphe 4).

En tout état de cause, les propositions faites, même si elles ne correspondent pas exactement à son idée, l'amènent à réfléchir et éventuellement à reformuler sa description. Le traitement linguistique de la négation apparaît alors comme participant pleinement au processus de conception de la scène. Si l'utilisateur utilise une négation, c'est parfois qu'il n'a pas une idée précise de la propriété implicite. Il vaut donc mieux donner une préférence aux propriétés issues des propriétés de base ainsi que des propriétés « supérieur à » et « inférieur à » qui sont naturellement floues. Il faut aussi rester vigilant. Avec toutes ces solutions, il arrive certainement que des propriétés proposées soient très similaires. Il convient donc d'effectuer une sélection pour réduire autant que possible cet ensemble de propriétés possibles.

7.2. Négation d'une propriété non-paramétrée

Lorsque l'utilisateur nie une propriété de la forme « x n'est pas f_α m_β A » issue d'une propriété de base A du concept, il sous-entend éventuellement une propriété paramétrée. Il est donc intéressant d'essayer d'introduire des propriétés paramétrées comme propriétés plausibles à une telle négation. Pour cela, il suffit de rechercher les propriétés plausibles de la négation du noyau de cette propriété selon la méthode exposée au paragraphe précédent. Ainsi, nous obtenons des propriétés paramétrées plausibles. Éventuellement, il est aussi possible de le faire avec le support. Il serait aussi intéressant de faire de même avec les autres propriétés de base du concept. Ces propriétés étant basées sur des valeurs significatives (valeurs « normales ») sont susceptibles de générer des intervalles intéressants.

Remarque : Le cas où le locuteur associe un intervalle « entre U et V » à la négation d'une propriété de base non-paramétrée A (de la forme « f m P_{ik} ») peut être interprété comme la définition d'une nouvelle propriété de base pour le concept. Cependant, ce choix est généralement spécifique à la description courante. Pour ajouter effectivement cette propriété, il faudrait mettre en place un système d'apprentissage spécifique. Son fonctionnement serait de proposer l'intervalle sélectionné de manière prioritaire à la prochaine utilisation de cette négation. Au bout d'un certain temps (qui reste à définir), le locuteur serait informé de la déduction et consulté sur la validité et, si la déduction est correcte, sur l'identificateur à utiliser (en lui proposant une liste de termes utilisant la propriété de base préfixée par chacun des préfixes de négation classiques comme non-P_{ik} , pas-P_{ik} , im-P_{ik} …).

7.3. Algorithme de génération des propriétés possibles

En généralisant, il suffit, pour toute propriété niée, de déterminer les propriétés paramétrées possibles à partir du noyau et, éventuellement, du support de cette propriété. Pour une propriété paramétrée, son support et son noyau sont éventuellement confondus. Ainsi, dans l'algorithme 1 sur la gestion de la négation, il faut remplacer la ligne « Propriétés_possibles = {f m P : $\forall$f $\in F$ et $\forall$m $\in M$} » par la ligne suivante : « Propriétés_possibles = Propriétés_possibles(A) » où « Propriétés_possibles(A) » est donné par l'Algorithme 6.

Algorithme 6. Détermination des propriétés possibles

```
Algorithme Propriétés_possibles(A↓) retourne {f m P}
 Possibles ← {f m P : ∀f ∈F_Q et ∀m ∈M_P et ∀P ∈C_i}
 [U,V] ← Noyau(A)
 Si A ≡ "f m inférieur à U" alors
   Possibles ← Possibles + {f m supérieur à U : ∀f ∈F_Q et ∀m ∈M_P }
 Sinon Si A ≡ "f m supérieur à U" alors
   Possibles ← Possibles + {f m inférieur à U : ∀f ∈F_Q et ∀m ∈M_P }
   Sinon Possibles ← Possibles + {f m supérieur à U : ∀f ∈F_Q et ∀m ∈M_P }
                               + {f m supérieur à V : ∀f ∈F_Q et ∀m ∈M_P }
                               + {f m inférieur à U : ∀f ∈F_Q et ∀m ∈M_P }
                               + {f m inférieur à V : ∀f ∈F_Q et ∀m ∈M_P }
   Fin si
 Fin si
 A' ← "entre U et V"
 Possibles ← Possibles +
              {"f m A' " ≡ "f entre U' et V' " : ∀f ∈F_Q et ∀m ∈M_P}
 Retourner Possibles
Fin Algorithme Propriétés_possibles
```

8. Petit exemple de synthèse

Afin d'illustrer le fonctionnement du modèle présenté dans ce chapitre et aux chapitres précédents, nous avons choisi un exemple simple mais significatif de description. L'objectif principal étant de montrer qu'une description linguistique de scène, faisant appel à des informations affirmatives et négatives, est envisageable en modélisation déclarative. Elle reprend les exemples choisis dans le contenu de ces chapitres.

Pour chaque propriété de cette description, nous proposons deux ou trois solutions parmi le grand nombre existant ainsi que les courbes des fonctions d'appartenance de la propriété de base et de la propriété élémentaire finale pour les trois premières ou le dialogue de l'étude de la négation (contenant elle aussi les deux courbes) pour les deux dernières.

La description comporte les énoncés (et les résultats) suivants :

- « *Il y a vraiment peu de segments verticaux* » (le nombre de verticales est vraiment faible) « *et très peu de segments horizontaux* » (le nombre d'horizontales est très faible) (Figure 51 et Figure 52).

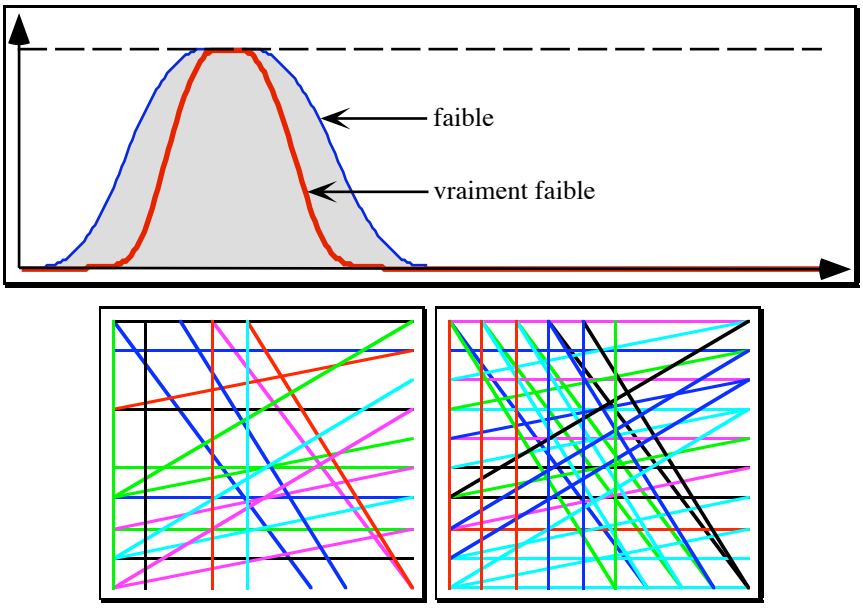

Figure 51. Scènes vérifiant : « le nombre de verticales est vraiment faible »

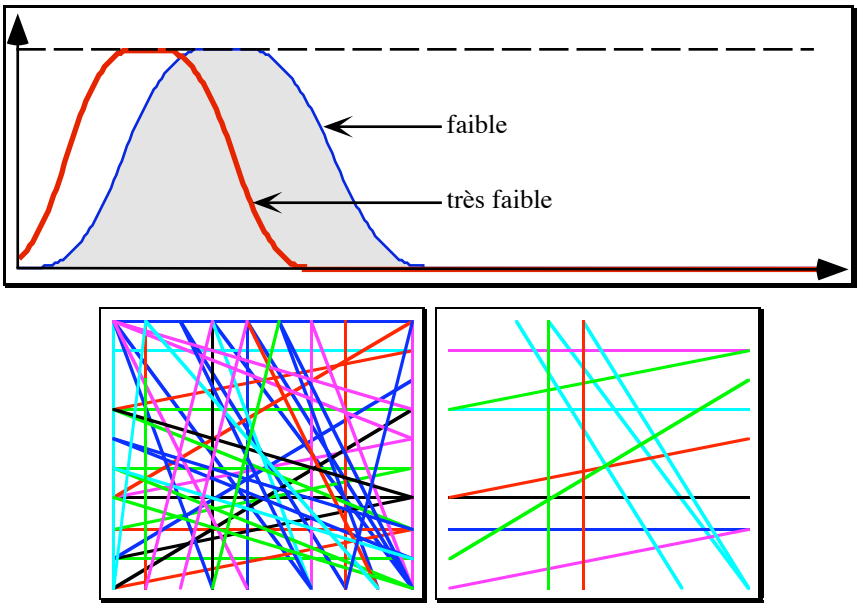

Figure 52. Scènes vérifiant en plus : « Le nombre de segments horizontaux est très faible »

- « *Le recouvrement est assez long* » (la longueur de recouvrement est assez importante) (Figure 53).

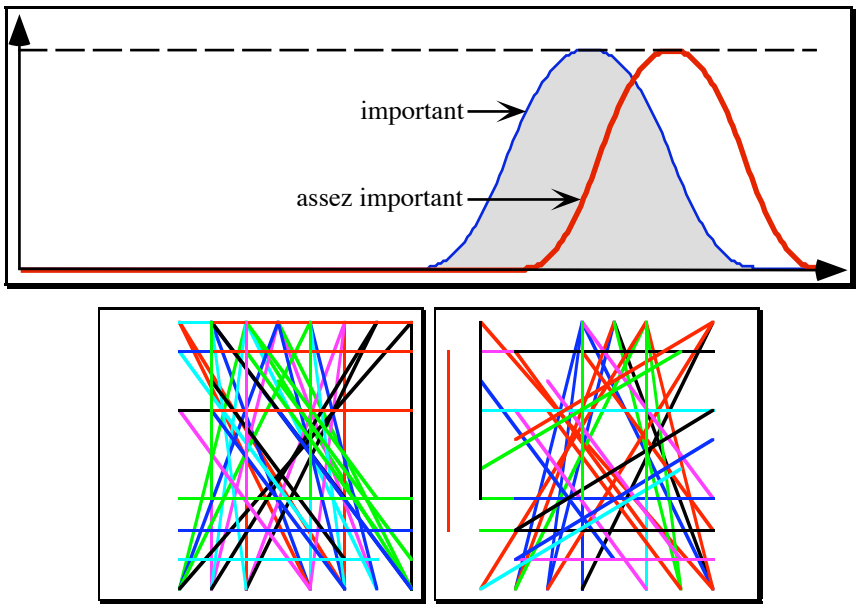

Figure 53. Scènes vérifiant en plus : « La longueur de recouvrement est assez importante »

- « *Le degré moyen de recouvrement n'est pas extrêmement important* » (Figure 55) (je sous-entends : il est moyen (Figure 54)).

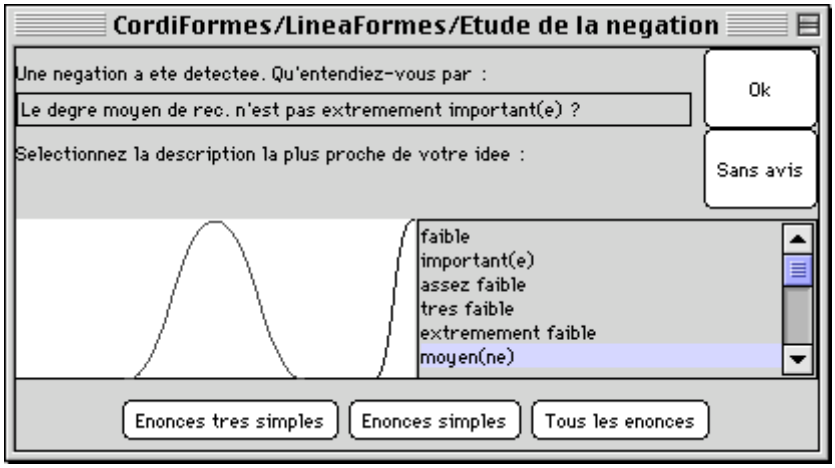

Figure 54. Gestion de la première négation

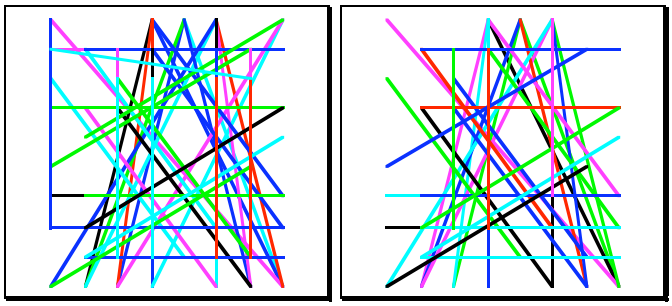

Figure 55. Scènes vérifiant : « Le degré moyen de recouvrement n'est pas extrêmement important »

- « *Il n'y a pas beaucoup de parallèles* » (Figure 57), c'est-à-dire très peu (Figure 56).

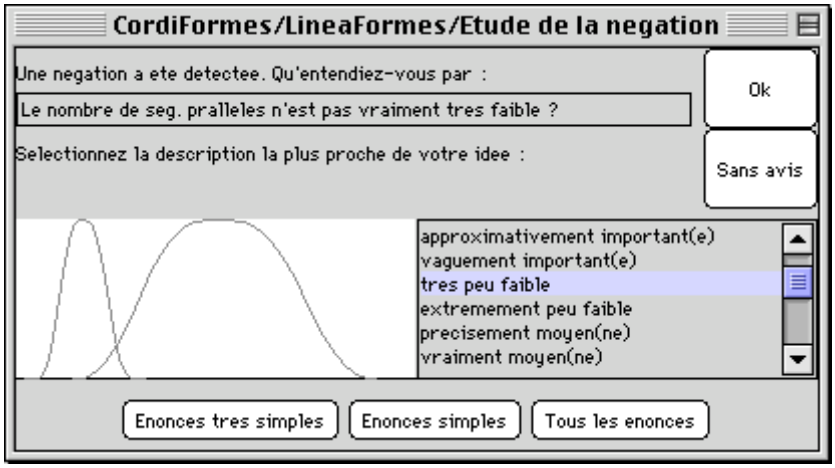

Figure 56. Gestion de la seconde négation

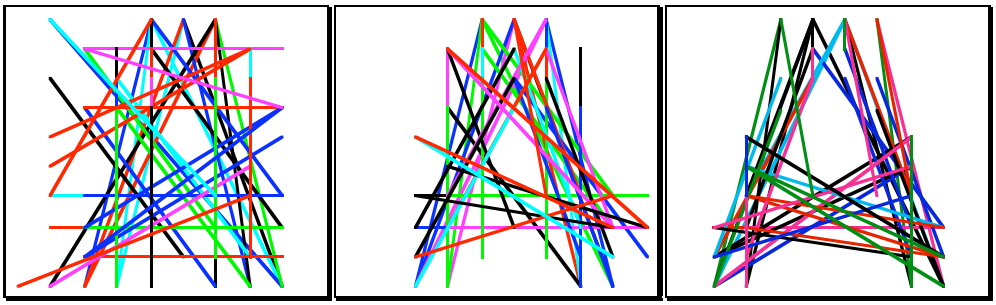

Figure 57. Scènes vérifiant : « La proportion de parallèles n'est pas vraiment très faible »

9. Conclusion

Contrairement à ce qui était habituellement fait, en particulier en modélisation déclarative, la négation d'une propriété ne revient pas à calculer sa négation logique. D'un point de vue linguistique ce n'est pas satisfaisant. Nous avons donc proposé un nouveau traitement basé sur des travaux en linguistique. Selon ces derniers, nier une propriété (ici, une propriété élémentaire) revient souvent à donner (implicitement) une autre propriété élémentaire sur le même concept. Nous avons proposé un ensemble d'outils permettant, dans le cadre de la description de scènes en modélisation déclarative, de déterminer et ordonner les propriétés plausibles comme implicites à la négation d'une propriété élémentaire sur un concept donné. Cette propriété est éventuellement paramétrée. C'est évidemment au locuteur d'avoir le dernier mot. Il peut, s'il n'a pas trouvé de solution convenable proposer une propriété élémentaire (paramétrée ou non) affirmative.

Les propriétés plausibles comme négation linguistique sont sélectionnées à partir des propriétés élémentaires possibles à l'aide d'une notion spécifique de similarité. Cette méthode, choisie arbitrairement, doit être étudiée plus finement et évaluée par rapport à d'autres calculs

de similarité existants (qu'on pourra trouver par exemple dans [Tve77], [DuP80b], [Zad87], [ZCB87], [DiK94]).

Notre méthode de classement et de sélection (selon le principe de simplicité…) permet d'éliminer un grand nombre de propriétés plausibles n'ayant pas de sens ou étant très peu probables parce que trop complexes. Cependant, il risque d'en rester quelques unes. Il serait alors intéressant de déterminer d'autres règles permettant de les supprimer. De plus, il faudrait affiner le traitement de la négation des propriétés élémentaires paramétrées.

Cette étude de la négation linguistique donne des résultats très intéressants dans le cadre de la modélisation déclarative. Tout d'abord, elle permet d'améliorer le processus de conception de la scène en encourageant le concepteur (le locuteur) à affiner sa pensée ou, plus précisément, sa réflexion sur les propriétés qu'il utilise. De plus, elle permet au modeleur déclaratif, une fois la description terminée, de ne traiter que des affirmations. Cela permet de mettre en œuvre des systèmes raisonnant sur des propriétés pour la gestion des ambiguïtés ou de la cohérence de la scène et ne traitant pas les formes négatives ([ADF83], [ADG84], [DAG86], [GFT92] et [GAT96]). Enfin, comme nous l'avons déjà vu, la détermination de la propriété élémentaire implicite à une négation permet une optimisation de la phase de calcul des solutions par rapport à la négation classique en réduisant l'espace des solutions à explorer et en permettant de ne proposer à l'utilisateur que des solutions proches de sa pensée.

Nous allons, dans le prochain chapitre, proposer des propriétés liées aux propriétés élémentaires : les propriétés élémentaires quantifiées, les propriétés statistiques sur des propriétés élémentaires et les propriétés modificatrices (de forme similaire aux propriétés élémentaires).

CHAPITRE I.5 : PROPRIÉTÉS LIÉES A LA PROPRIÉTÉ ÉLÉMENTAIRE

1. Introduction

Après avoir recensé un certain nombre de nouvelles propriétés, dans les deux chapitres précédents, nous avons présenté les propriétés élémentaires. L'objectif de ce chapitre est d'analyser les différentes propriétés dérivant de ces dernières. Nous allons étudier, en particulier, la gestion des :

- propriétés élémentaires quantifiées ;
- propriétés statistiques sur des propriétés élémentaires ;
- propriétés modificatrices.

D'un point de vue bibliographique, ces propriétés (et leur traitement) ont été assez peu étudiées en modélisation déclarative.

2. Les propriétés élémentaires quantifiées

Au chapitre I.3, nous avons étudié la représentation et la manipulation des propriétés élémentaires. Au chapitre I.2, nous avons noté que ces propriétés peuvent être classées en deux catégories : les propriétés locales et les propriétés globales. Les secondes sont en fait les propriétés élémentaires classiques. Par contre, les premières sont des propriétés élémentaires quantifiées (définition I.2.7) de la forme : « C de Q_t X est A » où Q_t est un quantificateur. La quantification indique une « quantité » d'objets vérifiant une propriété élémentaire donnée. Nous aurons alors des propriétés comme : « *La hauteur de tous les cubes est très très faible* », « *La longueur d'un segment est extrêmement faible* », « *La couleur de quelques maisons est ocre* »…

Évaluer une propriété élémentaire correspond à donner le degré d'appartenance de la scène à cette propriété. Cependant, comment pouvons-nous évaluer une propriété élémentaire quantifiée ?

Soit A la propriété locale demandée, Q_A la propriété quantifiée, $O=\{o_i\}$ l'ensemble des objets de la scène et $O_C=\{o_{ic}\}$ le sous-ensemble des objets concernés par le concept C sur lequel porte A ($O_C \subseteq O$ et nous supposerons $O_C \neq \varnothing$). Pour chacun des objets de O_C, il est facile de déterminer son degré d'appartenance à la propriété A : $\mu_A(o_{ic})$. Cependant, la difficulté réside dans la détermination de μ_{Q_A} le degré d'appartenance de la scène (ou plus précisément de O_C) à Q_A, car il dépend du degré d'appartenance à la propriété A (propriété de niveau 3) de chacun des objets.

Des solutions sont proposées dans [Zad83], [Yag84]… Nous allons nous intéresser ici à celle présentée dans [BLP96]. Une méthode d'évaluation est proposée pour des requêtes de la forme « Q des x A' sont A » avec A et A' deux propriétés éventuellement floues. Ils proposent en particulier un traitement lorsque A' est précise ($\mu_A(x) \in \{0,1\}$). Si nous posons que A' est « x est d'un type possédant C », nous retrouvons la forme de propriété élémentaire que nous avons proposée (« Q des x ayant C est A » ou « C de Q x est A ») .

Posons $O_C = \{x_1, \ldots, x_m\}$ avec $m \leq n = |O|$. Les éléments de O_C sont supposés ordonnés tels que $\mu_B(x_1) \geq \mu_B(x_2) \geq \ldots \geq \mu_B(x_m)$. Soit le quantificateur Q, sa fonction d'appartenance μ_Q est alors définie sur le sous-ensemble d'entiers $\{0, \ldots, m\}$ telle que $\mu_Q(k) \leq \mu_Q(k+1)$ pour k=0,m-1 et $\mu_Q(m)=1$. Il indique le nombre d'éléments à considérer. Avec Q, on associe un sous-ensemble I_Q défini par : $\mu_{I_Q}(k) = 1 - \mu_Q(k-1)$ pour k=1,m et $\mu_{I_Q}(0) = 1$. Il représente les rangs de O_C considérés comme importants dans l'évaluation. Nous aurons alors :

$$\mu_{Q_A}(O_C) = \min_{i=1,m}(\max(\mu_B(x_i), 1 - \mu_{I_Q}(i))).$$

Par exemple, on a les quantificateurs :

- « Tous » tel que $\mu_{tous}(k) = 0$ pour $k \leq m-1$ et $\mu_{tous}(m) = 1$;
- « Au moins U » tel que $\mu_{au\ moins\ U}(k) = 0$ pour k=1,u-1 et $\mu_{au\ moins\ U}(k) = 1$ pour k=u,m ;
- « La plupart » tel que $\mu_{la\ plupart}(k) = 0$ pour $k<p$, $\mu_{la\ plupart}(k) > 0$ pour $p \leq k < q$ et $\mu_{la\ plupart}(k) = 1$ pour $q \leq k \leq m$.

[BLP96] propose aussi des méthodes lorsque A est floue pour interpréter des requêtes de la forme : « Tous les x A sont B », « Pour la plupart des x, les A sont B », « La plupart des x sont A et B » ou « La plupart des x A sont B ».

Remarque : il ne propose pas de traitement pour des quantificateurs comme « aucun », « il existe », « quelques », « une dizaine », « environ 3 »… Cependant, certains sont calculables à partir de ceux proposés :

- $\mu_{il\ existe_A}(O_C) = \mu_{Au\ moins\ 1_A}(O_C)$;
- $\mu_{Aucun_A}(O_C) = 1 - \mu_{il\ existe_A}(O_C) = 1 - \mu_{Au\ moins\ 1_A}(O_C)$.

En effet, la formule proposée ne fonctionne que pour des opérateurs de quantification ayant une sémantique proche du « au moins k » (ou de sa négation) ou « tous sauf k ». Plus généralement, elle ne traite que les modificateurs issus de la relaxation de « tous ».

Pour illustrer cette technique, nous allons chercher à évaluer la propriété « *Au moins 6 segments ont une longueur assez peu importante* » dans la Figure 58.

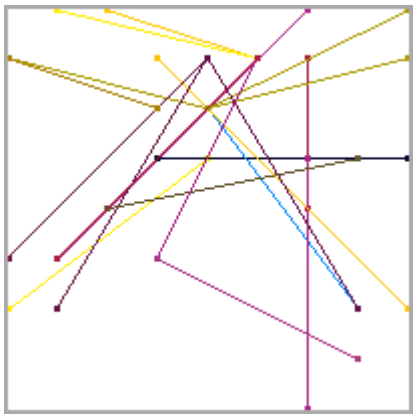

Figure 58. Une scène composée de segments

Les segments de cette figure ainsi que leur longueur et le degré d'appartenance à la propriété « *La longueur est assez peu importante* » sont regroupés dans le Tableau 6.

Tableau 6. Segments et leur degré d'appartenance à « assez peu important » (<2.950,7.188,7.704,2.682,0.438>)

X origine	**Y origine**	**X extrémité**	**Y extrémité**	**Longueur euclidienne**	μ
2	0	2	3	3	0.000
-3	-1	2	4	7.071	0.997
-4	-2	0	1	5	0.133
0	2	4	3	4.123	0.000
-1	1	4	1	5	0.133
1	3	-2	4	3.162	0.000
0	2	-4	3	4.123	0.000
-4	-1	0	3	5.657	0.462
3	-2	0	2	5	0.133
2	-4	2	1	5	0.133
-3	-2	0	3	5.831	0.576
4	-2	-1	3	7.071	0.997
-1	2	-4	3	3.162	0.000
-2	0	3	1	5.099	0.170
1	3	-3	4	4.123	0.000
0	2	4	4	4.472	0.013
-1	-1	1	3	4.472	0.013
3	-2	0	3	5.831	0.576
3	-3	-1	-1	4.472	0.013
-3	-1	1	3	5.657	0.462

En appliquant cette méthode, nous obtenons alors les courbes de la Figure 59 (le mode de calcul est donné au chapitre II.3, §4). Nous pouvons en déduire que la scène vérifie la propriété quantifiée « *Au moins 6 segments ont une longueur assez peu importante* », où « au moins 6 » est défini sur [1,20] par <0,6,20,0>, possède un degré de 0,462. De même, pour la propriété « *La plupart des segments ont une longueur assez peu importante* », où « La plupart » est défini par <5,15,20,0>, possède un degré de 0,133.

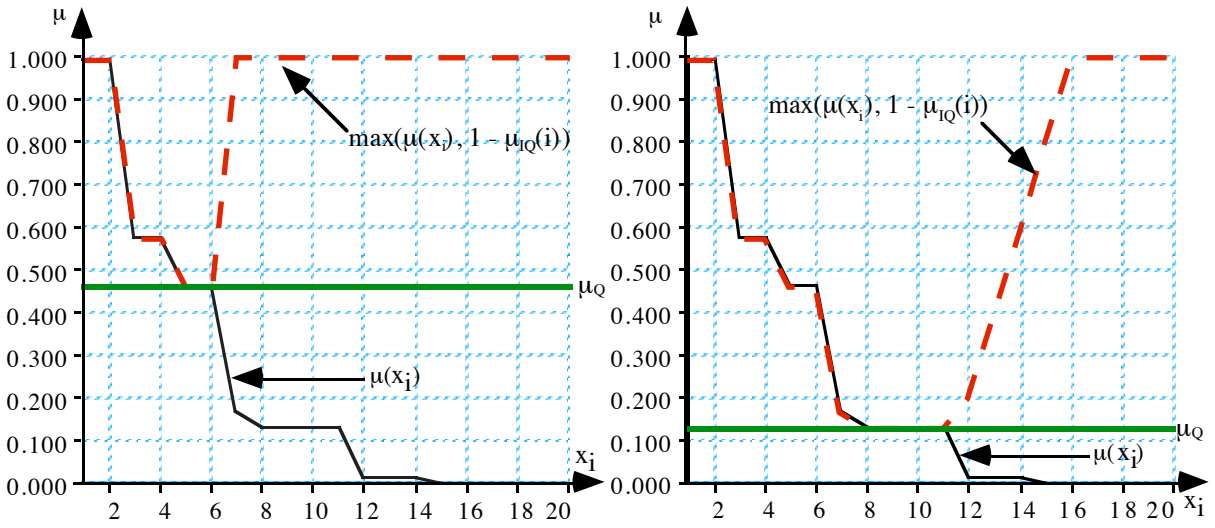

Figure 59. Calcul du degré d'appartenance de la scène à la propriété « Au moins 6 segments sont assez peu longs » et « La plupart des segments sont assez peu longs »

3. Propriétés statistiques sur les propriétés élémentaires

En dehors des propriétés basées sur le nombre d'éléments (quantification vue au paragraphe précédent), il existe d'autres propriétés basées sur des opérations statistiques. Elles sont formées de la manière suivante : « S(C) des x A' est A » où S est une opération statistique (la moyenne, l'écart-type…) basée sur le concept C et A et A' des propriétés élémentaires. Lorsque A' est « x est d'un type possédant C », l'évaluation d'une telle propriété est plus facile que celle d'une propriété quantifiée car elle est fonction d'une seule mesure (propriété de niveau 2). Celle-ci est calculée à partir des mesures des différents objets pris en compte. Nous avons :

$$E = \mu_S(O_C) = \mu_S(f(x_1, \ldots, x_m)).$$

4. Les propriétés modificatrices

Au chapitre I.2, nous avons mis en évidence une autre forme de propriété élémentaire : les propriétés modificatrices. Elles indiquent une propriété en fonction d'une scène, d'une forme courante. Plus précisément, elles font référence à la valeur de la scène selon le concept décrit et indiquent un intervalle (précis ou flou) préféré. La difficulté est de déterminer cet intervalle et de gérer les propriétés de la description sur ce concept ainsi que les éventuelles modifications précédentes. D'un point de vue bibliographique, ces propriétés (et leur traitement) n'ont pas été vraiment étudiées en modélisation déclarative. Les références sur le sujet indiquent qu'elles existent mais elles ne donnent pas précisément (le plus souvent pas du tout) la méthode de traitement. Seuls [Mou94] et [Bour97] ont proposé une solution. [Bour97] traite les propriétés de modification comme des propriétés de comparaison en considérant deux objets : l'objet courant (modifié) pris comme objet de référence et l'objet futur. Nous étudierons sa

méthode en présentant le traitement des propriétés de comparaison au paragraphe 3. La solution de [Mou94] est présentée au paragraphe 2.4.3. Nous allons examiner rapidement les problèmes posés par le traitement de telles propriétés d'abord indépendamment des autres descriptions, puis en fonction de celles-ci, c'est-à-dire en fonction du contexte d'application.

4.1. Application d'une propriété modificatrice

Une propriété modificatrice (définition I.2.20) est, rappelons le, de la forme suivante : « C_i [de X] est f_α k_β s_δ P_{ik} ». k_β est un opérateur de comparaison et s_δ est l'opérateur de direction qui indique la direction de comparaison par rapport à la direction de la propriété de référence P_{ik}. Cette propriété fait donc référence à un intervalle pour lequel les valeurs seraient plus adaptées. Comment pouvons-nous déterminer cet intervalle ? L'objectif est de construire une fonction spécifique permettant de rendre compte de cette propriété. Pour les propriétés précises (k_β est précis, paramétré), cela ne pose pas de problème. L'intervalle est déterminé de la même façon que pour les propriétés paramétrées sachant que le paramètre est une fonction de la valeur courante U. Nous aurons de manière générale une propriété de la forme « f_α entre F(U) et F'(U) » avec F et F' deux fonctions éventuellement identiques. Nous allons maintenant nous intéresser aux cas où k_β est un terme linguistique flou.

4.1.1 Les opérateurs de comparaison retenus

Les opérateurs de comparaison possibles étant très nombreux, nous avons restreint notre choix à un ensemble de R termes, noté : $K_R = \{k_\alpha \mid \alpha \in [1..R]\}$, pour lesquels existe la relation d'ordre total suivante : $k_\alpha \leq k_\beta \Leftrightarrow \alpha \leq \beta$. K_R comporte un opérateur particulier noté « normalement » ou « Ø ». C'est opérateur par défaut appelé aussi *opérateur de comparaison vide*. Dans ce travail, nous avons utilisé l'ensemble suivant : K_7={un tout petit peu, un petit peu, un peu, Ø, beaucoup, énormément, infiniment}.

4.1.2 Première méthode de traitement

Une première méthode pour traiter la propriété « C_i [de X] est f_α k_β s_δ P_{ik} » est de la considérer comme équivalente à la propriété élémentaire paramétrée : « C_i [de X] est f_α m_β PP » où PP est une propriété de base paramétrée de la forme « supérieur à » ou « inférieur à ». Par exemple, soit U la mesure courante du concept C alors la modification « C de x est un peu plus importante » sera considérée comme équivalente à la propriété élémentaire paramétrée « C de x est très peu supérieur à U ».

4.1.2.1 Choix de la propriété paramétrée

Définition I.5.1 : Le *signe de l'opérateur de direction* s_δ, noté signe(s_δ), est tel que : signe(« moins »)=-1, signe(« plus »)=+1 et signe(« aussi »)=0.

Définition I.5.2 : La *direction de modification* dépend du signe de l'opérateur de direction s_δ et de celui de la propriété de référence P_{ik} (définition I.3.6). Cette direction de modification est calculée selon les règles classiques de l'arithmétique :
$Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik}) = signe(s_\delta) * signe(P_{ik})$.

Définition I.5.3 : Le signe de la propriété de base paramétrée PP choisie pour former la propriété équivalente est le même que celui de la direction de modification.

Nous aurons alors les situations suivantes :

- $Signe(s_\delta)=Signe(P_{ik})\neq 0 \Rightarrow Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik}) = signe(PP)>0 \Rightarrow$ PP=« supérieur à U » ;
- $Signe(s_\delta)\neq Signe(P_{ik})\neq 0 \Rightarrow Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik}) = signe(PP)<0 \Rightarrow$ PP=« inférieur à U » ;
- $Signe(s_\delta) = 0 \Rightarrow Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik})=0 \Rightarrow$ PP=« de U ».

Dans le cas où la propriété obtenue est « C de x est $f_\alpha\ m_\beta$ supérieur à U_1 » nous obtiendrons l'intervalle représenté dans la Figure 60A et dans le cas où la propriété obtenue est « C de x est $f_\alpha\ m_\beta$ supérieur à U_2 » celui représenté dans la Figure 60B.

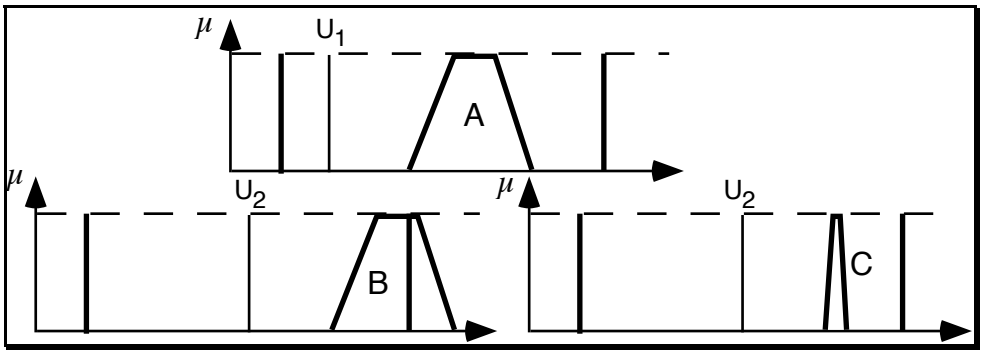

Figure 60. Calcul de la propriété issue d'une propriété comme « beaucoup plus P »

4.1.2.2 Choix du modificateur

Il reste maintenant à déterminer le « m_β ». Nous avions : M_9={extrêmement peu, très très peu, très peu, assez peu, normalement, assez, très, très très, extrêmement}. Il faut alors arriver à déterminer le bon terme en fonction de « k_β ». Nous allons poser que :
$\forall\alpha \in [1..R], \exists\beta \in [1..P] : k_\alpha \leftrightarrow m_\beta$ avec $k_\alpha \in K_R$ et $m_\beta \in M_P$.

Par exemple, nous pouvons poser les équivalences suivantes :

extrêmement peu	↔	un tout petit peu ;
très très peu	↔	un petit peu ;
très peu	↔	un peu ;
ø	↔	ø ;
très	↔	beaucoup ;
très très	↔	énormément ;
extrêmement	↔	infiniment.

Ainsi, si l'utilisateur donne « *La hauteur du cube est un peu plus importante* », en supposant la hauteur courante à 6, nous traduirons par « *La hauteur du cube est très peu supérieure à 6* ».

Remarque : Certaines associations « s_δ P_{ik} » sont plus courantes que d'autres. En particulier, lorsqu'il existe un couple marqué dans les propriétés d'un concept, la propriété non marquée, par définition représentant le concept, est la plus souvent utilisée. Ainsi, pour un concept de taille comportant {petit, moyen grand}, nous aurons plus souvent « moins grand », « plus grand » que « plus petit » et surtout que « moins petit ». Cette dernière propriété n'est sans doute presque jamais utilisée car elle correspond à une double négation assez rare en français.

4.1.2.3 Critique de la méthode

Cette solution est intéressante mais n'est pas totalement satisfaisante. En effet, il serait préférable d'obtenir des modifications à peu près identiques quelle que soit la position de la valeur de référence dans le domaine. Or, avec ce traitement, la modification dépend de la valeur mais aussi d'une borne du domaine. La modification sera d'autant plus importante que la valeur de référence sera loin de la borne. Au contraire, pour que le locuteur puisse manipuler efficacement les termes de modification, il faut qu'il puisse estimer la modification qu'ils impliquent. Pour deux valeurs du domaine ou pour deux applications successives d'une même modification, il faudrait que la différence soit du même ordre (comme entre la Figure 60A et la Figure 60C et non pas comme entre la Figure 60A et la Figure 60B). C'est pour cela que [Bour97] propose une autre solution (§2.4.1.2).

Il faudrait aussi améliorer le traitement des propriétés de modification en prenant en compte la notion de marquage. En effet, l'utilisation d'une modification peut dépendre de la propriété de base utilisée. En particulier, lorsque l'utilisateur décrit sa modification à l'aide d'une propriété marquée, cela peut indiquer que cette propriété est importante et doit donc être un peu vérifiée. Ainsi, pour un concept de taille, il peut donner l'une des deux propriétés suivantes :

- « x est beaucoup moins grand » ;
- « x est beaucoup plus petit ».

Ces deux propriétés sont, en première analyse, équivalentes. Cependant, l'utilisation du terme marqué « petit » semble indiquer qu'il voudrait que X soit plutôt petit alors que l'utilisation du terme non marqué ne donne pas cette information.

4.2. Prise en compte du contexte

Jusqu'ici, nous avons proposé le traitement d'une modification sans prendre en compte le contexte de traitement. Cependant, l'application d'une modification ne se fait pas toujours indépendamment de ce contexte, c'est-à-dire d'une part la description initiale et d'autre part les éventuelles modifications déjà demandées. Bien sûr, seules les propriétés et les modifications concernant le concept associé à la modification sont prises en compte.

4.2.1 Prise en compte des modifications précédentes

Pour construire une scène, l'utilisateur est parfois amené à réaliser plusieurs modifications successives pour un concept qui lui apparaît particulièrement intéressant. Lors de l'application réitérée des propriétés modificatrices soit on ne tient pas compte de ce qui a été fait précédemment soit on essaye d'affiner le traitement pour ne pas risquer de proposer une seconde fois une solution qui a conduit à une modification.

Prenons l'exemple suivant : l'utilisateur veut ajuster la hauteur d'un objet. Cette hauteur est un concept dont le domaine associé est $[1,80]_1$. Partant d'une solution de mesure 5, prenons les modifications successives suivantes :

1. « *La hauteur de l'objet est beaucoup plus importante* » ;
2. pour 60, « *La hauteur de l'objet est plus faible* » ;
3. pour 40, « *La hauteur de l'objet est plus importante* » ;
4. pour 45, « *La hauteur de l'objet est moins faible* »…

La Figure 61 montre les solutions successives que peut voir l'utilisateur au moment de chaque modification.

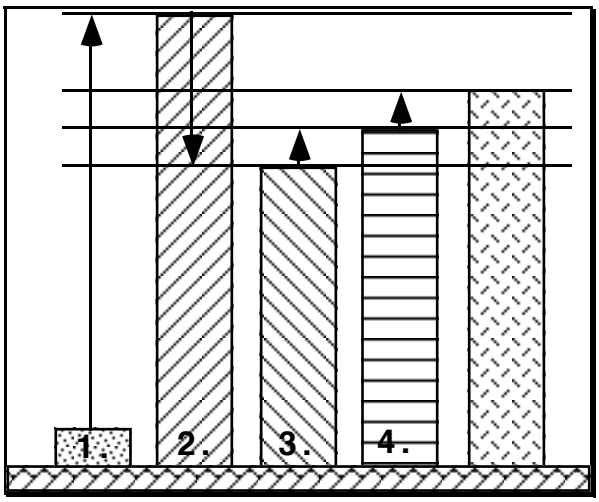

Figure 61. Modifications successives de la hauteur d'une boîte

Avec cet exemple, la première méthode (§4.1.2.), qui ne tient pas compte du contexte, engendre les courbes de gauche de la Figure 62.

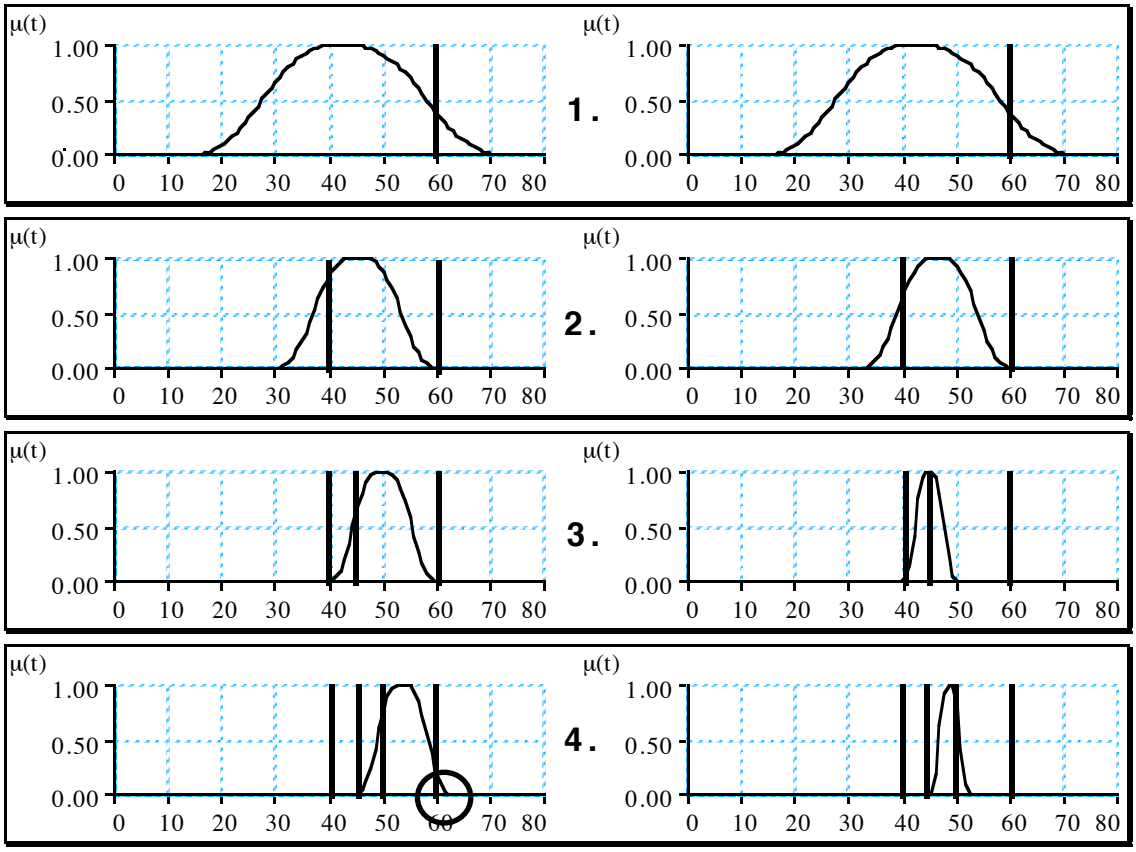

Figure 62. Modifications successives sur le domaine de « hauteur ».

Pour tenir compte des modifications passées, nous allons adapter cette méthode. Son principe est de remplacer la propriété modificatrice par une propriété élémentaire paramétrée en fonction de la valeur courante. Comme nous l'avons vu au paragraphe 8.2 du chapitre I.3, les propriétés paramétrées de type comparaison élémentaire « supérieur à » et « inférieur à » sont calculées à partir de la valeur de référence et respectivement de la borne supérieure et inférieure du domaine. Nous avons alors :

- « supérieur à U » = f(U, BM) ;
- « inférieur à U » = f'(U, Bm).

Nous allons nous intéresser aux bornes sur lesquelles elles sont calculées. Lorsque l'utilisateur énonce une propriété modificatrice correspondant à « supérieur à U » (resp. « inférieur à U »), il indique que toutes les valeurs inférieures (resp. supérieures) à ce U doivent être rejetées. Il faut donc garantir de ne pas y revenir lors des générations futures. Par conséquent, la borne inférieure courante (resp. la borne supérieure courante) du domaine devient U. Cette seconde méthode appliquée à l'exemple donné en début de paragraphe donne les courbes de droite de la Figure 62. En résumé, nous avons selon le cas :

- « supérieur à U » ⇒ f(U, BM) et Bm = U ;
- « inférieur à U » ⇒ f'(U, Bm) et BM = U.

4.2.2 Difficultés posées par ces méthodes

Notons que cette seconde méthode pose malgré tout un problème. Pour deux applications d'une même modification pour deux valeurs très proches, l'intervalle obtenu risque d'être très différent. L'utilisateur risque alors de perdre ses repères et d'avoir du mal à obtenir sa solution. La première méthode, où les bornes restent fixes, ne pose pratiquement pas ce problème et est, de ce point de vue, plus intéressante. En ce qui concerne la première méthode, elle risque de mener à proposer des valeurs rejetées par des modifications précédentes. Nous trouvons cette situation à la Figure 62 (étape 4.). La seconde valeur modifiée (60) peut être à nouveau proposée (partie entourée). Ce défaut peut être vu comme un avantage lorsqu'on autorise l'utilisateur à changer d'avis sur sa valeur idéale (ce qui est un peu plus difficile à faire avec la seconde méthode). La Figure 63 met en évidence les problèmes posés par ces deux méthodes. La première méthode (à gauche) propose des valeurs rejetées par la seconde modification (intervalle en partie dans une zone hachurée, c'est-à-dire interdite). La seconde méthode (à droite) propose à la troisième modification un intervalle beaucoup plus petit que celui proposé à la première modification alors que celle-ci est la moins « forte » des deux d'un point de vue sémantique (courbes entourées).

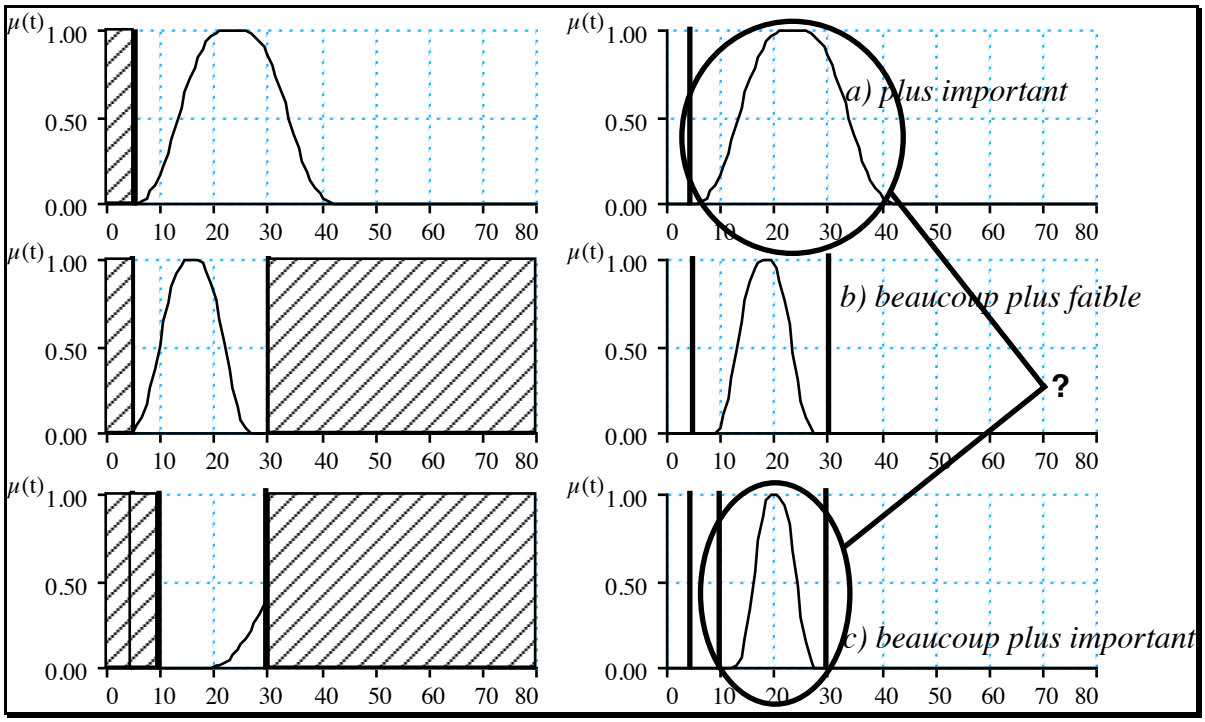

Figure 63. Problèmes posés par les deux méthodes de modification

Il y a encore un autre problème dont nous n'avons pas encore parlé : la gestion de la cohérence des modifications successives. En fait, deux cas se présentent :

- Les modifications sont effectuées les unes après les autres sans s'occuper des précédentes (c'est ce que nous avons fait pour l'exemple de la Figure 62).
- Pour chaque application, la cohérence avec les applications précédentes est vérifiée.

Dans ce dernier cas, il est nécessaire de vérifier l'intersection avec les intervalles précédents. Quand elle existe, l'intervalle final est réduit à cette intersection. Dans le cas contraire, soit la propriété modificatrice est rejetée, soit l'utilisateur prend l'ultime décision pour éventuellement rompre volontairement la cohérence.

4.2.3 Prise en compte de la description initiale

Dans certains cas, il arrive que, sur le concept que l'utilisateur veut modifier, soit demandé une ou plusieurs propriétés. Nous allons supposer ici que ces propriétés sont des propriétés élémentaires. Il faut alors gérer la modification en fonction de la description et éventuellement de l'avis de l'utilisateur sur la solution. Comme seules les scènes vérifiant la description initiale sont proposées à l'utilisateur (à moins que la propriété décrite soit une des alternatives d'une disjonction), ces propriétés sont vérifiées. Lorsque l'utilisateur donne sa propriété modificatrice, nous avons alors deux cas possibles :

- La description et la propriété modificatrice (plus exactement, leurs fonctions d'appartenance) ont une intersection (Figure 64a). La génération peut alors être lancée sur la partie commune sans autre traitement.
- La propriété modificatrice n'a pas d'intersection avec la description. Il faut alors gérer un cas d'incohérence (Figure 64b). Nous retrouvons le problème classique déjà vu au paragraphe précédent qui est soit de rejeter soit de demander à l'utilisateur de trancher.

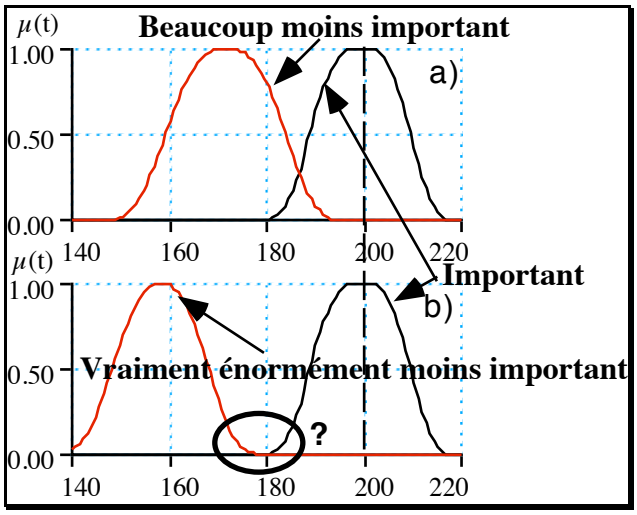

Figure 64. Cas entre description et propriété modificatrice

Du point de vue de la phase de génération en modélisation déclarative, les propriétés modificatrices interviennent au niveau des choix des valeurs à considérer et non dans l'évaluation de la description. Elles donnent la possibilité à l'utilisateur de diriger la recherche vers des valeurs qu'il pense plus intéressantes. Elles n'ont donc pas leur place dans la description. Il faut tenir compte de cela pour la gestion de l'incohérence. Si l'utilisateur insiste sur sa modification, il est nécessaire de l'informer que telle ou telle propriété de sa description ne sera

plus valable (et donc désactivée). Ce point de vue favorise aussi l'idée que la prise en compte des modifications précédentes n'est pas forcément utile. L'utilisateur doit pouvoir changer de plage d'étude pour voir d'autres solutions parfois très différentes.

4.3. Autre méthode de traitement

Le traitement des modificateurs, que propose [Mou94], est original. A l'aide de la propriété, il définit une fourchette de valeurs autorisées. La valeur courante sert de borne supérieure ou inférieure suivant que le signe de la modification est respectivement moins ou plus. Ensuite, il opère un tirage aléatoire sur cet intervalle sachant que la probabilité pour chaque valeur d'être choisie est *répartie suivant une courbe de Gauss*. L'apogée de cette courbe est choisie en fonction de la quantité associée au modificateur. Ainsi, la répartition des valeurs n'est pas équiprobable et le calcul permet de rendre compte plus fidèlement de l'imprécision de la modification. La formule de Gauss choisie est la suivante : $G(x) = K(e^{-x^a} - b)$. Avec cette méthode, les bornes de l'intervalle de validité sont modifiées de la même manière que ce que nous avons vu au paragraphe 2.4.1. Cette méthode, spécifique au type de génération choisi, pose donc des problèmes identiques.

5. Conclusion

Après avoir proposé un modèle cohérent de propriétés élémentaires, nous venons de présenter quelques idées de traitement pour d'autres catégories de propriétés qu'on peut trouver en modélisation déclarative et liées avec les propriétés élémentaires.

La solution proposée pour les propriétés quantifiées, nous l'avons vu, ne couvre pas tous les cas de quantificateurs. Il faudrait donc étudier d'autres solutions en particulier celles proposées dans les travaux sur les requêtes floues en base de données ([Zad83], [Yag84], [DuP90]…). Il en est de même pour les propriétés statistiques. De plus, pour l'instant les objets sont regroupés selon les critères « x est d'un type possèdant C » mais le problème plus ardu lorsque les objets étudiés sont regroupés selon une propriété spécifique par exemple : « *La plupart des segments très long sont horizontaux* » ou « *La pente moyenne des segments verticaux est assez faible* ». Ces traitements, nous l'avons vu, sont donc le plus souvent imparfaits. Il sera donc nécessaire de les reprendre afin de les améliorer.

Dans le prochain chapitre, nous allons présenter une autre catégorie de propriétés mettant en jeu plusieurs objets ou concepts de la scène quelque soit le quantificateur utilisé : les relations.

CHAPITRE I.6 : LES RELATIONS

1. Introduction

Jusqu'ici, nous nous sommes intéressés au traitement et à la manipulation de propriétés portant sur un seul concept, c'est-à-dire ne portant que sur un objet de la scène (éventuellement la scène entière) d'un point de vue conception (avec l'utilisation du quantificateur existentiel). Les propriétés élémentaires quantifiées et statistiques telles que nous les avons présentées sont des extensions à un ensemble d'objets mais toujours en faisant référence à une seule caractéristique de la scène. Seulement, comme nous l'avons vu au chapitre I.2, beaucoup de propriétés portent directement ou indirectement sur plusieurs caractéristiques.

Définition I.6.1 : une *relation* est une propriété d'un concept portant sur plusieurs caractéristiques ou sur plusieurs objets de la scènes.

Il existe trois types de relations :
- les propriétés relatives (de niveau 1) ;
- les propriétés de relation (de niveau 2) et, en particulier, les propriétés de comparaison ;
- les propriétés de compositon (de niveau 3).

Nous verrons d'abord une première idée de traitement des propriétés de comparaison permettant de mettre en relation plusieurs propriétés (section 2). Puis, nous aborderons la gestion des compositions de propriétés (section 4) qui font intervenir plusieurs propriétés. Les propriétés relatives ne sont pas abordées ici, car nous avons constaté au chapitre I.2 que leur traitement se rapporte à celui des propriétés élémentaires.

2. Propriétés de comparaison

2.1. État de l'art

D'un point de vue bibliographique, les relations (et leur traitement) n'ont pas été vraiment étudiées en modélisation déclarative. La formalisation de ce type de propriété a été principalement abordée par [Chau94b] et [Ple91]. Pour eux, les contraintes sont représentées par des contraintes mathématiques sur des paramètres de la scène (mesures de la scène selon certains concepts). Les opérateurs utilisés sont évidemment : $<$ (inférieur), $>$ (supérieur), $\leq$ (inférieur ou égal), $\geq$ (supérieur ou égal), $=$ (égal) et $\neq$ (différent). [Ple91] introduit trois opérateurs originaux : $<<$ (très inférieur), $>>$ (très supérieur) et $\sim$ (presque égal). Cependant, il ne précise

pas la sémantique exacte de ces opérateurs. Il propose aussi des modificateurs sur ses opérateurs mais ne précise pas le traitement. Généralement, c'est au concepteur d'application de spécifier la sémantique de tous ces opérateurs. Il n'existe pas de modèle formel, général et indépendant du domaine.

Seul [Bour97] a, nous le verrons, proposé une solution pour les propriétés de comparaison. [Des95b] propose une première solution pour l'évaluation des propriétés de comparaison. Les relations de base sont binaires (sur deux objets). Les autres relations sont le plus souvent des relations binaires sur lesquelles portent un quantificateur. Nous ne présenterons donc que le traitement de relations portant sur deux objets.

2.2. Types de comparaison

Les propriétés de comparaison (définition I.2.16) sont des énoncés de la forme : « C_i ([de X_p] *ou* [entre X_p et X_r]) est f_α k_β s_δ P_{ik} que C_j ([de X_q] *ou* [entre X_q et X_s]) ». C_i et C_j sont deux concepts (unaires ou binaires) dont les valeurs des domaines (D_i et D_j) sont comparables au niveau arithmétique et sémantique. La propriété de référence P_{ik} est une propriété de C_i ayant forcément un équivalent sémantique dans C_j. De plus, elle ne peut pas être une propriété paramétrée (« *A est beaucoup plus de 2m que B* » ne se dit pas). Les comparaisons hétérogènes (définition I.2.17) posent le problème de garantir l'équivalence des deux domaines. Lorsqu'on énonce la propriété « *La hauteur de A est plus importante que la largeur de B* », il faut être certain que le concept « Hauteur » possède une propriété « importante » (par exemple « Haut ») ainsi que le concept de « Largeur » (par exemple « Large »), c'est-à-dire « Haut » ≡ « Large ».

En réalité, il y a deux types de propriétés de comparaison en fonction du k_α utilisé :

- les propriétés de comparaison basées sur une différence (avec « beaucoup », « un peu »…) ;
- celles basées sur une proportion (avec « 2 fois », « une dizaine de fois »… plus généralement tous les coefficients de la forme « p fois »).

Nous allons considérer que ces propriétés sont chacune associées à un concept particulier :

- le *concept de comparaison par différence* (ou *concept de différence*) ;
- le *concept de comparaison par proportion* (ou *concept de proportion*).

Ces concepts sont des concepts « paramétrés » par deux concepts, c'est-à-dire qu'ils sont fonction de deux concepts. Ils seront notés respectivement $Comp_{C_i,C_j}$ et $Prop_{C_i,C_j}$.

2.3. Le concept de différence

Lorsque le locuteur utilise une propriété basée sur le concept de différence $Comp_{C_i,C_j}$, il entend comparer les valeurs des mesures des deux concepts. Il doit pouvoir donner des énoncés comme : « *A est beaucoup plus grand que B* », « *A est un petit peu moins grand que B* »… Pour un tel concept (de niveau 2), la mesure est une fonction $m_{Comp_{C_i,C_j}}$ basée sur les mesures des concepts paramètres telle que :

$m_{Comp_{C_i,C_j}}$: $D_i \times D_j \rightarrow D_{Comp_{C_i,C_j}}$

$m_i, m_j \rightarrow d = m_i - m_j$

Avec $D_{Comp_{C_i,C_j}}$ tel que $[-\Delta, \Delta]_u$ où Δ correspond à la plus grande différence possible.

Si l'on prend $D_i = [Bm_i, BM_i]_{u_i}$ et $D_j = [Bm_j, BM_j]_{u_j}$ nous aurons :

$\Delta = Max(|Bm_i - BM_j|, |BM_i - Bm_j|)$ et $u = Min(u_i, u_j)$.

Remarque : Le traitement des comparaisons par différence où l'opérateur de comparaison k_α est une valeur précise (par exemple « *A est de 2m plus grand que B* ») est trivial. La différence valide est explicitement donnée par cet opérateur. Dans la suite, nous ne considérerons que des opérateurs de comparaison flous.

Nous allons d'abord exposer une méthode présentée dans [Des95b] puis nous verrons une autre méthode qui nous semble plus intéressante du point de vue linguistique.

2.3.1 Traitement d'une comparaison par différence

Ce concept possède une propriété $P_{Comp_{C_i,C_j}}$ dont nous allons essayer de déterminer la fonction d'appartenance en fonction des opérateurs utilisés et des conditions d'application (caractéristiques des concepts paramètres). Cette propriété est définie par :

$D_{Comp_{C_i,C_j}} \rightarrow [0,1]$

$d = m_i - m_j \rightarrow \mu_{Comp_{C_i,C_j}}(d)$

Remarque : Lorsque l'opérateur de comparaison est précis (de la forme « entre U et V »), $P_{Comp_{C_i,C_j}}$ s'obtient très facilement en utilisant les paramètres ($\mu_{Comp_{C_i,C_j}}(d) = \mu_{<0,U,V,0>,L,R}(|d|)$).

Définition I.6.2 : La *direction de comparaison* dépend du signe de l'opérateur de direction s_δ et de celui de la propriété de référence P_{ik}. Elle est calculée selon les règles classiques de l'arithmétique : $Direction(k_\beta\ s_\delta\ P_{ik}) = signe(s_\delta\ P_{ik}) = signe(s_\delta) * signe(P_{ik})$.

Définition I.6.3 : Le *signe de la différence* est le signe de la différence des mesures des concepts paramètres.

Théorème I.6.1 : La propriété ne peut être vérifiée que si le signe de la différence est le même que la direction de comparaison.

Le Tableau 7 regroupe les différentes situations qu'on peut rencontrer pour évaluer la propriété selon cette dernière définition.

Tableau 7. Première évaluation du degré d'appartenance à la propriété $P_{Comp_{C_i,C_j}}$

Signe(P_{ik})	Signe(s_β)	Signe($m_{C_i} - m_{C_j}$)	Direction($k_\beta\, s_\delta\, P_{ik}$)* Signe($m_{C_i}-m_{C_j}$)	$\mu_{Comp_{C_i,C_j}}$
+1	+1	+1	+1	≥ 0
+1	+1	-1	-1	= 0
+1	-1	+1	-1	= 0
+1	-1	-1	+1	≥ 0
-1	+1	+1	-1	= 0
-1	+1	-1	+1	≥ 0
-1	-1	+1	+1	≥ 0
-1	-1	-1	-1	= 0
±1	0	±1	±1	≥ 0

L'opérateur de direction « aussi » est un cas particulier du point de vue linguistique. En effet, il n'autorise que l'utilisation de l'opérateur de comparaison « Ø » et est indépendant du signe de la propriété de référence. Cependant, cet opérateur n'a aucun effet. Le traitement doit donc prévoir le traitement de cet opérateur où ce sont les différences « autour » de 0 qui seront intéressantes.

Remarque : L'application de l'opérateur flou f_α se fait sur la fonction d'appartenance de $P_{Comp_{C_i,C_j}}$ de la même manière que pour une propriété élémentaire ou une propriété relative puisque cet opérateur est indépendant de la sémantique de la propriété sur laquelle il porte (voir chapitre I.3, §6.3).

La propriété $P_{Comp_{C_i,C_j}}$ est déterminée à l'aide de la direction de comparaison mais aussi de l'opérateur de comparaison et de la sémantique des concepts paramètres. Cette propriété définit un intervalle flou représentant les écarts autorisés des mesures paramètres. La forme de cet intervalle dépend de la sémantique des concepts (« largeur » de l'intervalle) et de la position de la « force » de l'opérateur de comparaison (« translation » de l'intervalle).

Définition I.6.4 : Le *coefficient de différence linguistique* T_{C_i} est un coefficient qui rend compte d'une différence significative liée au concept utilisé.

Pour que les deux concepts soient considérés comme comparables, il faut que leurs coefficients de différence linguistique soient équivalents ($T_{C_i} \approx T_{C_j}$). Généralement, ce coefficient est proportionnel à l'intervalle du domaine. Il représente la translation élémentaire rendant compte de la différence de "force" entre deux opérateurs. En effet, la comparaison dépend des concepts sur lesquels elle porte. « *La boîte A est plus grande que la boîte B* » la différence de

mesure n'est pas du même ordre de grandeur que lorsqu'on dit « *L'immeuble A est plus grand que l'immeuble B* » ou « *La planète Mars est moins éloignée que la planète Pluton* ».

Définition I.6.5 : Le coefficient de comparaison λ_α associé à un opérateur de comparaison k_α est un entier tel que $\lambda_\alpha \leq \lambda_\beta \Leftrightarrow k_\alpha \leq k_\beta$.

Remarques :

- Les opérateurs de comparaison sont ceux définis pour les propriétés modificatrices. Nous avons donc la même définition pour K_R.
- En pratique, pour K_7={un tout petit peu, un petit peu, un peu, ø, beaucoup, énormément, infiniment}, nous avons choisi $L(K_7) = \{1, 2, 3, 4, 5, 6, 7\}$. Ces valeurs semblent assez conformes aux sémantiques relatives des différents opérateurs de comparaison (définitions I.6.5) et permettent une distribution satisfaisante sur le domaine.

La Figure 65 montre les différentes propriétés pour chacun des opérateurs de comparaison choisis.

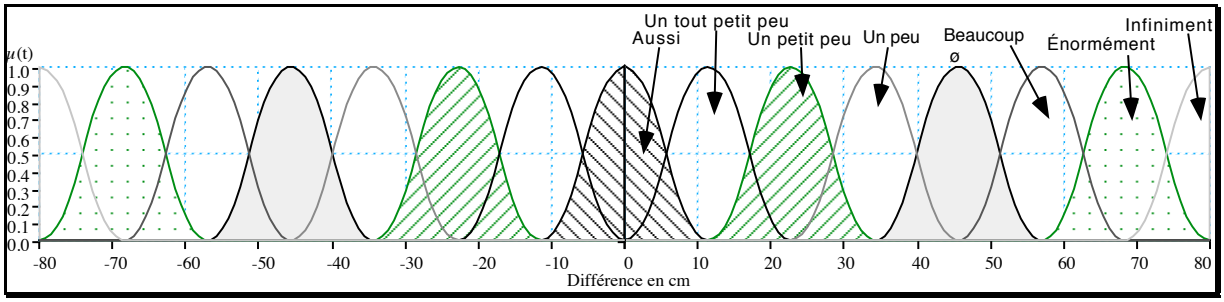

Figure 65. Opérateurs de comparaison pour une propriété du domaine des tailles avec T_{taille}=11.4

2.3.2 Bilan

Si la propriété est $P_{Comp_{C_i,C_j}} = \{D_{Comp_{C_i,C_j}}, \mu_{Comp_{C_i,C_j}}\}$, la fonction d'appartenance associée à la propriété de comparaison est :

$\mu_{Comp_{C_i,C_j}}(d) =$

0 si $Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik}) \neq Signe(m_i - m_j)$ et $Signe(s_\beta) \neq 0$

$\mu_{<T_{C_i}, 0, 0, T_{C_i}>, L_{CompC_i,C_j}, R_{CompC_i,C_j}}(|d|)$ si $Signe(s_\beta) = 0$

$\mu_{<T_{C_i}, \lambda_\alpha * T_{C_i}, \lambda_\alpha * T_{C_i}, T_{C_i}>, L_{CompC_i,C_j}, R_{CompC_i,C_j}}(|d|)$ sinon

Dans la fonction d'appartenance de la comparaison, $\lambda_\alpha * T_D$ représente la différence « idéale » associée au comparateur en fonction du domaine. La Figure 66 et la Figure 67 proposent quatre courbes de comparaison de deux objets A et B dont les « tailles » varient de 140 à 220. Dans ces figures, les zones sont d'autant plus noires que la propriété concernée est vérifiée. Cette solution ne nous semble cependant pas suffisante. Elle ne rend pas compte des

nuances linguistiques amenées par les opérateurs de comparaison. Nous allons donc proposer une nouvelle méthode.

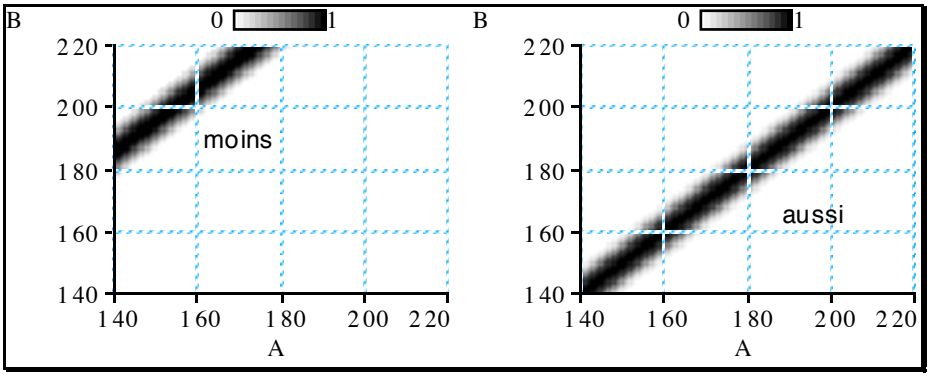

Figure 66. Comparaisons entre deux objets A et B (1)

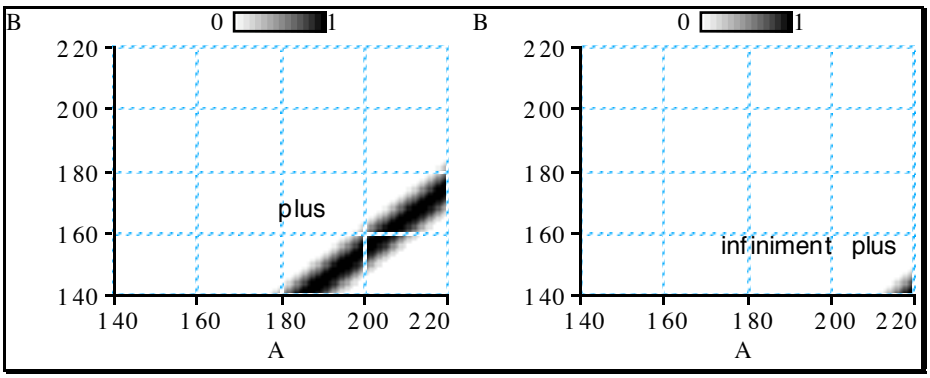

Figure 67. Comparaisons entre deux objets A et B (2)

2.4. Autre solution pour le concept de différence

Une autre méthode pour déterminer la propriété du concept de différence est de procéder comme pour les propriétés modificatrices, c'est-à-dire de chercher une propriété élémentaire paramétrée équivalente sur le domaine. Autrement dit, il s'agit de trouver une propriété élémentaire paramétrée de la forme « C [de X] est f_α m_β PP » sur $D_{Comp_{C_i,C_j}}$ telle qu'elle représente un intervalle flou des différences autorisées. Nous allons choisir un traitement identique à celui effectué pour les propriétés modificatrices (§2.4.1). La détermination de la propriété de base paramétrée est identique à celle effectuée pour les propriétés modificatrices.

Théorème I.5.2 : Le signe de la propriété de base paramétrée PP choisie pour former la propriété équivalente est le même que celui de la direction de comparaison.

Ici, la valeur de référence U est déterminée à priori. Les différences sont qualifiées par rapport à la différence minimale qui est 0. Nous aurons alors les situations suivantes :

- $Signe(s_\delta)=Signe(P_{ik})\neq 0 \Rightarrow Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik})>0 \Rightarrow signe(PP)>0 \Rightarrow PP=$« supérieur à 0 » ;
- $Signe(s_\delta)\neq Signe(P_{ik})\neq 0 \Rightarrow Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik})<0 \Rightarrow signe(PP)<0 \Rightarrow PP=$« inférieur à 0 » ;
- $Signe(s_\delta) = 0 \Rightarrow Direction(f_\alpha\ k_\beta\ s_\delta\ P_{ik})=0 \Rightarrow PP=$« de 0 ».

La Figure 68 montre l'utilisation de la propriété paramétrée « supérieur à 0 » pour déterminer la propriété de différence sur une propriété de comparaison de « tailles ».

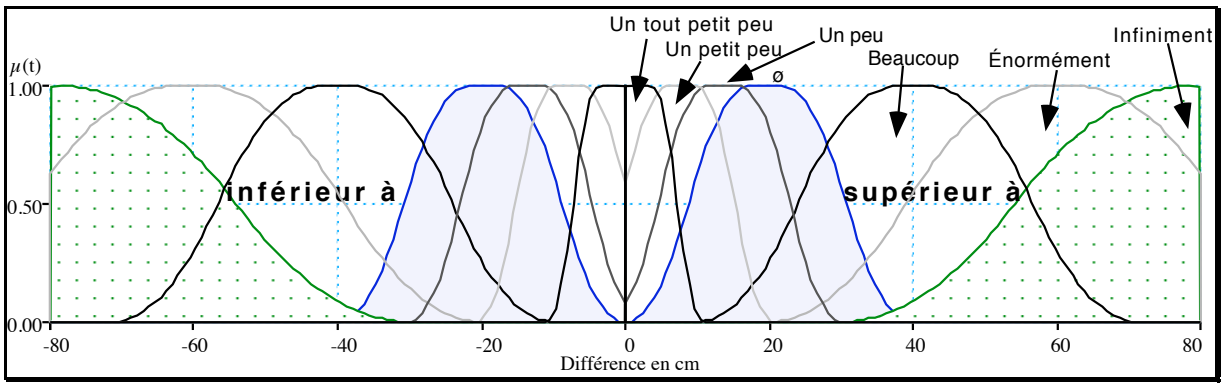

Figure 68. Concept de différence à l'aide de l'association entre « plus P_{ik} » et « Supérieur à 0 »

Le choix du m_β est exactement identique à celui exposé pour les propriétés modificatrices. Nous avons :

extrêmement peu	↔	un tout petit peu ;
très très peu	↔	un petit peu ;
très peu	↔	un peu ;
ø	↔	ø ;
très	↔	beaucoup ;
très très	↔	énormément ;
extrêmement	↔	infiniment.

Ainsi, si l'utilisateur énonce « *La taille de la statue A est plus importante que celle de la statue B* » la propriété de différente sera « *La différence de taille entre A et B est supérieure à 0* » et on obtiendra la courbe de la Figure 69A. L'autre courbe de cette figure et celles de la Figure 70 montrent d'autres propriétés de différence : « moins important » (Figure 70A), « aussi important » (Figure 70B) et « infiniment plus important » (Figure 69B).

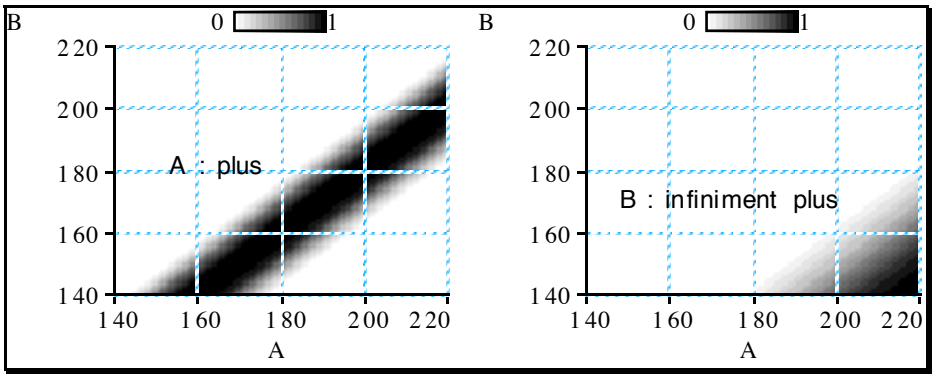

Figure 69. Comparaisons entre deux objets A et B selon la nouvelle méthode (1)

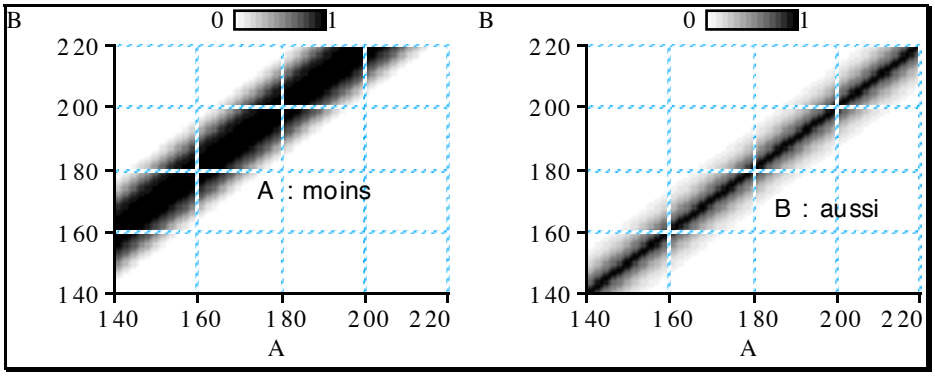

Figure 70. Comparaisons entre deux objets A et B selon la nouvelle méthode (2)

Cette solution nous semble plus intéressante que la précédente d'un point de vue sémantique. En effet, les courbes sont mieux réparties. La description sur des différences faibles est plus fine. De plus, ces courbes rendent compte d'une dilatation avec l'utilisation d'opérateurs de comparaison plus « puissants ».

Remarque : Cette méthode ne fait plus intervenir le coefficient de différence linguistique introduit pour la première méthode. Ce coefficient voulait rendre compte de la sémantique des concepts utilisés. Cependant, pour avoir des intervalles « bien répartis », sa valeur était directement dépendante des bornes des domaines des concepts paramètres. Il en est de même pour les propriétés calculées selon la seconde méthode. Elles dépendent indirectement des bornes des deux domaines.

2.5. Le concept de proportion

Lorsque l'utilisateur utilise une propriété basée sur le concept de proportion $Prop_{C_i,C_j}$, il entend indiquer un rapport entre les valeurs des mesures des deux concepts. Il doit pouvoir donner des énoncés comme : « *A est 4 fois plus grand que B* », « *A est une dizaine de fois moins grand que B* »… Plus généralement, nous visons ici le traitement de propriétés de comparaison où k_α = « p fois », c'est-à-dire des propriétés de la forme (avec p un réel quelconque) :

« C_i ([de X_p] *ou* [entre X_p et X_r]) est f_α p fois s_δ P_{ik} que C_j ([de X_q] *ou* [entre X_q et X_s]) ».

Remarques :

- La méthode présentée ici est celle présentée dans [Des95b].
- Ce ne sont donc que des propriétés paramétrées par un réel « p ».

Pour un tel concept (de niveau 2), la mesure est une fonction $m_{Prop_{C_i,C_j}}$ basée sur les mesures des concepts paramètres telle que :

$m_{Prop_{C_i,C_j}}$: $D_i \times D_j \rightarrow D_{Prop_{C_i,C_j}}$

$m_i, m_j \rightarrow d = \{m_i , m_j\}$ Avec $D_{Prop_{C_i,C_j}} = D_i \times D_j$.

Ce concept possède une propriété $P_{Prop_{C_i,C_j}}$ dont nous allons essayer de déterminer la fonction d'appartenance en fonction des opérateurs utilisés et des conditions d'applications (caractéristiques des concepts paramètres). Cette propriété est définie par :

$D_{Prop_{C_i,C_j}} \rightarrow [0,1]$

$d = \{a, b\} \rightarrow \mu_{Prop_{C_i,C_j}}(d)$

La propriété $P_{Prop_{C_i,C_j}}$ se comporte comme l'étude de l'égalité où les opérandes sont ajustées en fonction de l'opérateur de comparaison et de l'opérateur de direction. Cette étude dépend essentiellement de la direction de la comparaison. Pour une propriété « C_i est f_α "p fois" s_δ P_{ik} que C_j », si cette direction est positive alors nous étudierons $m_i \approx p * m_j$ ou, dans le cas contraire, $p * m_i \approx m_j$. Nous avons donc :

$$\mu_{Prop_{C_i,C_j}}(\{a, b\}) = \mu_{<\alpha, a, b, \beta>, L_{PropC_i,C_j}, R_{PropC_i,C_j}}(|a-p*b|) \text{ si Direction}(k_\beta\, s_\delta\, P_{ik}) > 0$$

$$\mu_{<\alpha, a, b, \beta>, L_{PropC_i,C_j}, R_{PropC_i,C_j}}(|p*a-b|) \text{ sinon}$$

Avec α et β deux réels

La Figure 71 présente les courbes des deux propriétés « *La taille de A est 3 fois plus importante que la taille de B* » (Figure 71a) et « *La taille de A est 3 fois plus faible que la taille de B* » (Figure 71b). Cette solution semble suffisante avec l'hypothèse que l'opérateur de comparaison est précis. Cependant, il reste à s'assurer que les opérateurs de comparaison ne peuvent pas être, dans ce cas, flous. Si c'était le cas, il faudrait chercher à déterminer un nouveau traitement.

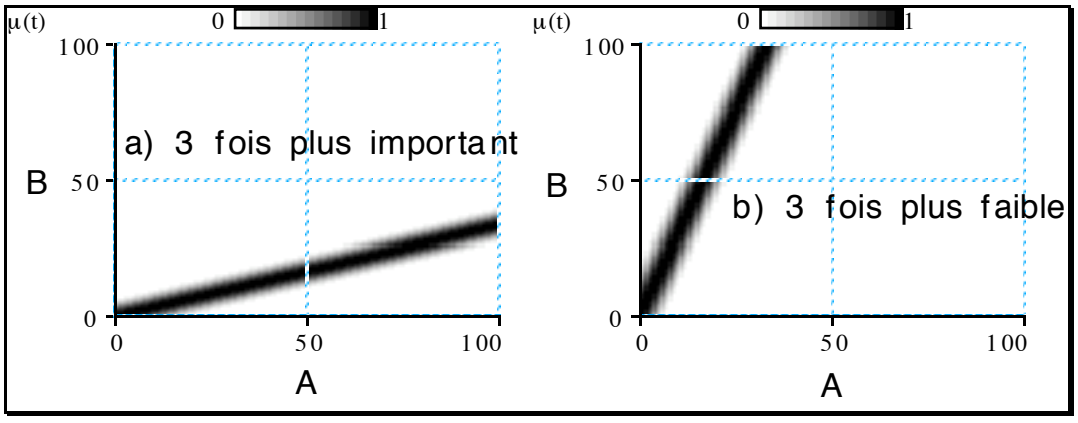

Figure 71. Proportion entre la taille de deux objets

2.6. Autre Solution pour les propriétés de comparaison

Cette solution est exposée dans [Bour97]. Seules les relations binaires sont prises en compte. Elles sont classées en trois catégories : les relations de construction (par exemple « *A est beaucoup plus grand que B* », « *A ressemble à B* »), les relations de placement (par exemple « *A est très à gauche de B* », « *A est inclus dans B* ») et les relations de modification (par

exemple « *A est beaucoup plus grand* »). Ce classement ne semble pas judicieux, car il mélange la forme (relation de construction), la sémantique (relation de placement) et le rôle (relation de modification). De toute manière, le traitement qu'il propose est identique quelle que soit la relation. Dans le cas des propriétés de modification, il considère que « *A est plus grand* » est équivalent à la relation de construction « *A est plus grand que B* » avec A=« A futur » et B=« A actuel ».

Remarque : Cette idée déjà présentée dans le chapitre I.2, [DeM97a] et [DeM97b] peut être appliquée avec la méthode que nous avons présentée dans les paragraphes précédents.

[Bour97] propose un autre classement : les relations qui se comportent selon un modèle de concept (en fait nos propriétés relatives) et celles qui se comportent selon un modèle de différence de concepts (nos propriétés de comparaison par différence). Le traitement des premières est identique à celui de nos propriétés relatives. L'idée de base est de définir un écart caractéristique ou *écart significatif* pour un concept donné. Il est fixe pour toute propriété du concept. Ainsi, l'utilisateur possède une idée de l'écart de base provoqué par une différence sur ce concept. [Bour97] a choisi une solution où la comparaison est indépendante des bornes et donc ne dépend que des valeurs relatives de paramètres. Cet écart est alors un ensemble flou défini à partir d'une propriété du concept et d'un *indice de différence*. La propriété est choisie parce qu'elle est « représentative du concept ». Ce choix est facile lorsqu'il existe un couple marqué (on prend le terme non marqué qui est, par définition, représentatif du concept). Par contre, lorsqu'il n'y a pas possibilité de différencier les propriétés, ce choix devient « aléatoire ». L'indice de différence est « très subjectif » selon [Bour97]. Aucune indication n'est donnée pour le choix de la valeur, car elle est, elle aussi, fortement dépendante du concept considérée. Cet écart E_{C_i} est défini par une fonction LR $<\alpha, a, b, \beta>$ telle que :

$$\alpha = \text{indice}*\delta/3 \quad \beta = \text{indice}*\delta/3$$
$$a = \text{indice}*\delta/3 \quad b = 2*\text{indice}*\delta/3$$

avec une propriété P_i définie à l'aide de $<\alpha_i, a_i, b_i, \beta_i>$, « indice » l'indice de différence et $[Bm_i, BM_i]$ l'intervalle support de P_i tel que $Bm_i = a_i-\alpha_i$ et $BM_i = b_i+\beta_i$ et $\delta = BM_i-Bm_i$.

Ensuite, [Bour97] définit l'orientation de la différence. Cette orientation est calculée de la même manière que ce que nous avons appelé la direction de comparaison, c'est-à-dire qu'elle est calculée à partir du signe de la propriété de référence et du signe de l'opérateur de comparaison de telle sorte que : Orientation = Signe(P)*Signe(OpeComp) avec OpeComp dans {« plus », « moins », « aussi »}.

[Bour97] oriente ses comparaisons. En effet, il détermine un « objet défini », noté Obj_{def}, et un « objet de référence », noté Obj_{ref}. Il met en évidence deux formes de relations qui corres-

pondent aux comparaisons par différence et aux comparaisons par proportion. Elles sont de la forme :

1. [C_x de] Obj_{def} est OpeComp P_{ik} [que/de/à] [C_y de] Obj_{ref} ;
2. [C_x de] Obj_{def} est n fois OpeComp P_{ik} [que/de/à] [C_y de] Obj_{ref}.

Pour chacune de ces formes, il détermine une formule permettant de « placer » l'écart significatif $E_{C_i} = <\alpha, a, b, \beta>$ dans le domaine du concept décrit pour l'objet défini. Cet intervalle devient :

- Pour la forme 1.

$\alpha_{def} = \alpha$	$\beta_{def} = \beta$
$a_{def} = a + V_{ref} +$ (Signe(P_{ik})*Signe(OpeComp)* ´δ')	$b_{def} = b + V_{ref} +$ (Signe(P_{ik})*Signe(OpeComp)* ´δ')

- Pour la forme 2.

$\alpha_{def} = p*\alpha$	$\beta_{def} = p*\beta$
$a_{def} = (1/n) * V_{ref}$ - (´(b-a)*p)	$b_{def} = (1/n) * V_{ref}$ + (´(b-a)*p)

avec V_{ref} la valeur supposée connue de Obj_{ref} dans C_y, $\delta' = b+\beta-(a-\alpha)$ et p = 0 si orientation=0, (1+n/100) si orientation>0 et (1-n/100) si orientation<0.

Ces deux formes peuvent avoir des variations en leur appliquant des opérateurs flous et des modificateurs (avec « beaucoup » remplaçant « très », « un peu » remplaçant « assez » et « un petit peu » remplaçant assez peu »). Ceux-ci portent alors sur l'intervalle flou obtenu de la même manière que sur une propriété de base pour les propriétés élémentaires.

Cette méthode présente l'avantage d'être moins dépendante de l'équivalence entre les deux concepts utilisés pour la comparaison. Elle ne demande pas la construction d'un nouveau domaine avec les problèmes de choix de bornes et d'unité qui sont liés. Elle se contente de prendre une valeur dans le concept de l'objet référence et travaille uniquement dans le concept de l'objet défini. Cette méthode est surtout intéressante pour la gestion des propriétés modificatrices. En effet, la valeur de référence est déjà connue et le concept défini est le même que le concept de référence. Elle a l'avantage de proposer une modification constante quelque soit les bornes. Le locuteur peut ainsi mieux maîtriser les modifications qu'il demande comme nous l'avons déjà montré en présentant les propriétés modificatrices. Pour les méthodes que nous avons présentées, leur adaptation aux propriétés modificatrices n'est pas facile à cause de la définition du nouveau concept de différence. Par contre, contrairement aux méthodes que nous avons présentées, cette méthode ne peut être appliquée que lorsqu'on connaît la mesure de l'objet de référence. Elle est donc dépendante de la génération et ne peut faire l'objet d'une éventuelle phase de raisonnement pour lever certaines incohérences et ambiguïtés. Enfin, elle pose un autre problème : les différents écarts obtenus après application des modificateurs ne sont pas tous normalisés. En effet, il peut arriver que, pour certains modificateurs (en particulier l'équivalent de « extrêmement peu »), le noyau ne soit pas sur la zone valide, c'est-

à-dire du « bon » côté de la valeur de référence. Ainsi, il sera impossible d'avoir la propriété totalement vérifiée (à un degré 1). Dans nos propositions, l'utilisation du concept de différence pour les relations ou l'utilisation des propriétés paramétrées pour les propriétés modificatrices permet d'éviter ce problème.

3. Combinaison de propriétés

Une description peut être considérée comme une disjonction de conjonctions de propriétés. Globalement, une description est de la forme :

« (P_{11} et P_{12} et P_{13}...) ou (P_{21} et P_{22} et P_{23}...) ou (P_{31} et P_{32} et P_{33}...) ... ».

Les P_i sont des propriétés quelconques telles que des propriétés élémentaires, des propriétés de comparaison… Une description est donc équivalente à une propriété de composition (définition I.2.26). Nous allons étudier comment interpréter simplement une conjonction, une disjonction et, plus généralement, une description. Ces interprétations sont celles utilisées en logique floue. Elles ne gèrent que les opérateurs "." (confondus avec "Et"). Elles ne prennent pas en compte les éventuelles relations entre les opérandes (voir §4.5, chapitre I.2).

3.1. Conjonction de propriétés

Une scène doit comporter plusieurs propriétés qui sont demandées simultanément. C'est une propriété de composition avec seulement des opérateurs conjonctifs "Et". Nous avons alors une conjonction de propriétés comme par exemple : « *La maison est grande, assez éloignée et plus allongée que la première* ». Ce sont des propriétés de la forme : « P_1 et P_2 et P_3... ». Pour interpréter de telles descriptions, nous utilisons l'extension du produit cartésien à *n* propriétés (intersection de *n* ensembles flous, voir Annexe 1). Nous avons alors :

$D_1 \times D_2 \times ... \times D_n \rightarrow [0,1]$

$t_1, t_2, ..., t_n \rightarrow \mu(t_1, t_2, ..., t_n) = T_{i \text{ dans } [1,n]}(\mu_{P_i}(t_i))$

avec T une t-norme adaptée (voir Annexe 1). Nous utiliserons, par exemple, l'opérateur classique "min(x)".

Remarque : Il serait intéressant d'étudier la possibilité d'utiliser une moyenne (pondérée ou non) à la place de la t-norme [DuP93].

3.2. Disjonction de propriétés

Dans certaines descriptions, l'utilisateur peut laisser le choix entre plusieurs descriptions. C'est une propriété de composition avec seulement des opérateurs disjonctifs "ou". Nous aurons alors une disjonction de propriétés comme par exemple : « *La maison est haute ou allongée* ». Ce sont des propriétés de la forme : « P_1 ou P_2 ou P_3... ». Pour interpréter de telles des-

criptions, nous utilisons l'extension du produit cartésien à n propriétés (union de n ensembles flous, voir Annexe 1). Nous avons alors :

$D_1 \times D_2 \times ... \times D_n \rightarrow [0,1]$

$t_1, t_2, ..., t_n \rightarrow \mu(t_1, t_2, ..., t_n) = \perp_{i \text{ dans } [1,n]}(\mu_{P_i}(t_i))$

avec $\perp$ une t-conorme adaptée (voir Annexe 1). Nous utiliserons, par exemple, l'opérateur classique "max(x)".

3.3. Description

L'interprétation d'une description comme disjonction de conjonctions consiste donc à évaluer :

$D_{11} \times D_{12} \times ... \times D_{1n} \times D_{21} \times D_{22} \times ... \times D_{2n} \times ... \rightarrow [0,1]$

$t_{11}, t_{12}, ..., t_{1n}, t_{21}, t_{22}, ..., t_{2n}, ... \rightarrow \mu(t,...) = \perp_{i \text{ dans } [1,m]}(T_{j \text{ dans } [1,n]}(\mu_{P_{ij}}(t_{ij})))$

4. Conclusion

Après avoir proposé un modèle cohérent de propriétés élémentaires, puis des propositions de traitement de propriétés qui sont liées à ces dernières, nous venons de présenter quelques idées de traitement pour d'autres catégories de propriétés qu'on peut trouver en modélisation déclarative : les relations. Ces traitements, nous l'avons vu, sont le plus souvent imparfaits. Il sera donc nécessaire de les reprendre afin de les améliorer. Dans le prochain chapitre, nous allons faire un bilan du modèle présenté.

CHAPITRE I.7 : CONCLUSION

1. Introduction

Après avoir exposé les différents types de propriétés et proposé des solutions de traitement, nous allons conclure en montrant d'abord comment sélectionner une forme en fonction de la validité de la description (section 2). Puis nous ferons un bilan de ce formalisme au niveau de la description (section 3), de la génération (section 4) et de la prise de connaissance (section 5). L'objectif de ce paragraphe est d'une part d'analyser les utilisations du formalisme présenté dans cette première partie et d'autre part de le situer par rapport à ce qui est réalisé en modélisation déclarative en mettant en évidence les avantages potentiels.

2. Sélection d'une scène solution

Définition I.7.1 : Une forme F de l'univers des formes U_f est dite *forme ou scène solution* si la description est considérée comme vérifiée, c'est-à-dire si toutes les propriétés d'une des conjonctions de la description (disjonction de conjonctions) sont vérifiées.

Pour simplifier, nous considérerons dans ce paragraphe, une description comme une conjonction de propriétés. Par ailleurs, lors d'une étude de forme, il est indispensable de déterminer si une propriété donnée est ou n'est pas vérifiée. Pour cela, il nous suffit de définir, comme dans [GAT96], le seuil d'acceptation.

Définition I.7.2 : Un *Seuil d'Acceptation* (S.d.A.) est une valeur de degré d'appartenance à partir de laquelle la propriété est considérée comme vérifiée. Il est défini par une fonction filtre ϕ et une valeur S tel que :

$$\phi_S : [0,1] \rightarrow [0,1]$$
$$t \rightarrow \begin{cases} t & \text{si } t \geq S \\ 0 & \text{sinon} \end{cases}$$

Le seuil d'acceptation peut être : calculé automatiquement, déterminé par le concepteur ou déterminé par l'utilisateur. Plusieurs politiques sont applicables quant à sa détermination suivant l'exigence portée sur la notion d'appartenance. En voici quelques unes :

- On considère qu'une propriété est présente dès que le degré d'appartenance n'est pas nul. Le seuil d'acceptation est alors le plus bas possible. Par exemple, en admettant que le degré d'appartenance est défini avec 1 chiffre significatif, nous aurons : $S = 0,1$.

- On considère que dès que le degré d'appartenance n'est pas négligeable, la propriété est vérifiée. Par exemple, si on considère que le degré d'appartenance est négligeable au-dessous de 0,15 nous aurons : $S = 0,15$.
- La propriété est vérifiée dès que le degré d'appartenance est supérieur à 50% (choix d'un seuil d'acceptation "raisonnable"). Nous aurons alors : $S = 0,5$.
- On considère qu'une propriété n'est vérifiée que lorsque le degré d'appartenance est proche de 1. Nous aurons par exemple : $S = 0,9$.
- Le seuil d'acceptation est déterminé en fonction des opérateurs flous de la propriété et de la description. On peut aussi prendre en compte l'ensemble des propriétés (en calculant un degré de flou moyen par exemple).

Remarques :

- Les spécialistes du flou utilisent généralement la valeur 0.1 ou 0.2 comme seuil.
- Le choix d'une politique peut être global (pour toutes les propriétés) ou spécifique à chaque propriété.
- Le seuil d'acceptation de la description ne semble pas nécessaire car il peut être déduit des seuils d'acceptation des propriétés. Il se détermine alors en fonction des conjonctions, des disjonctions, des t-normes et des t-conormes utilisés. Cependant, un seuil d'acceptation global permet de rendre compte d'une certaine incertitude quant à la description. Par exemple, nous pouvons avoir des débuts de description comme : *"La description est exactement la suivante : …"* ou *"La description est à peu près la suivante : …"*…
- Si S =0, la propriété devient inutile.
- Si S =1, la propriété est une propriété classique (précise ou imprécise et non floue).

Nous avons donc montré qu'il est possible de choisir entre plusieurs politiques pour déterminer un seuil d'acceptation. La détermination de la forme de la propriété est à la charge du concepteur : d'un bon choix dépendra une production raisonnable de solutions. A priori, dès que le degré d'appartenance de la scène à la propriété n'est plus nul, on peut considérer que cette scène appartient un peu à la propriété. Autrement dit, toute scène dont le degré d'appartenance à la propriété n'est pas nul vérifie (plus ou moins) cette propriété. En conséquence, nous pouvons choisir un seuil d'acceptation qui soit juste au-dessus de 0 (0.01 ou 0.1 suivant la précision des propriétés). La Figure 72 montre que, plus le seuil d'acceptation est haut, plus l'imprécision est affaiblie. Par conséquent, si on veut malgré tout « profiter » de l'imprécision des propriétés, il est préférable que ce seuil ne soit pas trop important.

On considère qu'avec un seuil d'acceptation S nous avons, pour une propriété $P=\{D, \mu,\tau\}$:

- si $\mu \geq S$ alors la propriété est vérifiée,
- si $\mu < S$ alors la propriété n'est pas vérifiée.

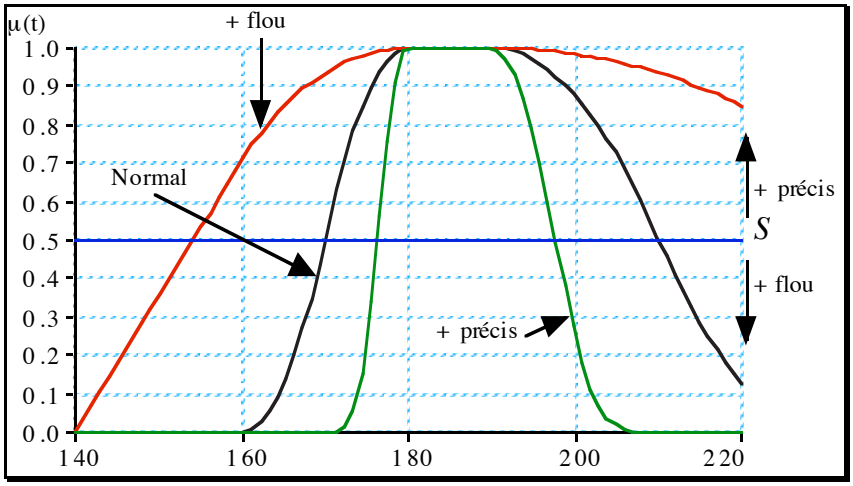

Figure 72. Étude de l'influence du seuil d'acceptation d'une propriété

Remarque : Nous pouvons facilement faire la correspondance avec la notion de α-coupe en logique floue (voir annexe 1). L'ensemble des mesures de D solutions par rapport au seuil d'acceptation est donc une α-coupe telle que $\alpha=S$. Donc, pour une propriété donnée P d'un domaine D, une forme est solution si la mesure t selon D est telle que : $\phi_S(t)>0$ ou $t \in P_S$ (P_S est la coupe de niveau *S* ou *S* -coupe de la propriété P) (voir Figure 72).

3. Description

3.1. La sémantique plus fine.

La "forme" de l'intervalle est fonction de la description. Le *modèle de description interne* [CDMM97b] utilisant les ensembles flous (manipulé par les outils de description et l'algorithme de génération) est alors beaucoup plus fidèle à la sémantique de la *description externe* (donnée par l'utilisateur). Il y a moins de pertes d'informations. Cette représentation rend compte du flou "naturel" des termes utilisés dans la description. Classiquement, une propriété est donnée par rapport à un "idéal" et plus on s'éloigne de cet idéal, moins la propriété est vérifiée. Tout cela se fait progressivement. Notons que les formes des énoncés proposés tout au long de cette partie sont généralement assez restrictives. Elles permettent de mettre en place un langage de description proche du langage naturel mais très restrictif. Pour espérer traiter un texte en langage naturel, il sera nécessaire d'étudier un système permettant de transformer le texte original selon les formes présentées et de proposer des variantes à ces énoncés. Ce système doit en outre être capable de lever un certain nombre d'ambiguïtés afin de bien déterminer quel concept est décrit, sur quel objet et avec quelle propriété.

3.2. La gestion de la cohérence de la description est plus efficace.

Les méthodes de *détection de l'incohérence* d'une description proposées par [Don93] (basée sur la logique de Allen [All83] appliquée aux intervalles), [Mac92] et [Chau94b] sont

valables. Les méthodes de *traitement de l'incohérence* sont valables de la même façon. En particulier, si on choisit la méthode de compromis ([Chau94b]…), ce modèle affine le traitement. Par exemple, au lieu de modifier directement les intervalles (perte partielle de sémantique), on va modifier les propriétés en cause dans la description en modifiant les opérateurs flous. Ainsi, la modification pourra être plus compréhensible par l'utilisateur et la sémantique de la description sera mieux conservée. Il existe aussi des méthodes plus complexes basées sur des systèmes à base de règles et raisonnant sur des réseaux sémantiques et des ensembles flous ([ADG84], [DAG86], [GFT92] et [GAT96]).

3.3. Intervalles classiques et intervalles flous

Un risque important pourrait être qu'un système utilisant les intervalles flous engendre beaucoup plus de solutions qu'un système habituel utilisant les intervalles classiques. En pratique, si on compare la sémantique d'une propriété donnée par les intervalles classiques et par les ensembles flous, on peut déduire cinq situations (Figure 73).

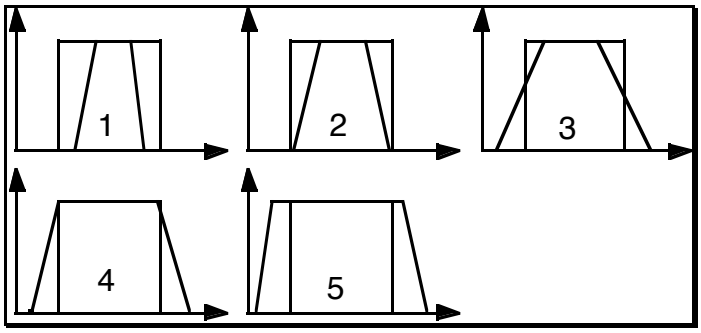

Figure 73. Comparaisons entre intervalles classiques et intervalles flous

Soit n le nombre de solutions avec les intervalles classiques et N avec les intervalles flous. Les situations 1 (le support de l'intervalle flou est inclus dans l'intervalle classique) et 2 (le support de l'intervalle flou est identique à l'intervalle classique) ne produiront certainement pas plus de solutions ($n \geq N$). Les situations 4 (le noyau de l'intervalle flou est identique à l'intervalle classique) et 5 (l'intervalle classique est inclus dans le noyau de l'intervalle flou), par contre, produisent effectivement plus de solutions ($n \leq N$). Le nombre de solutions engendrées par la solution 3 dépend du choix du seuil d'acceptation ($n \; ? \; N$). Par conséquent, le rapport n / N dépend de l'implémentation. Globalement, les choix les plus probables sont les solutions 2, 3 et 4.

C'est donc au programmeur de l'application de faire attention à bien choisir la fonction d'appartenance et les différents paramètres par rapport au domaine d'application et au nombre de solutions potentielles. La gestion du nombre de solutions potentielles par rapport à l'intervalle choisi est identique pour les deux méthodes.

4. Génération

4.1. Remarque sur le seuil d'acceptation

A première vue, le seuil d'acceptation provoque une équivalence entre ce modèle et la théorie classique en retournant a une situation binaire {Vrai, Faux}. Dans un certain sens, c'est effectivement le cas. Cela montre que les générateurs classiques sont utilisables avec ce modèle. Il n'est donc pas nécessaire de tout refaire. Nous ne prétendons pas fournir un modèle qui fasse travailler plus vite les générateurs. C'est même, si on n'y prend pas garde, parfois plus lent (plus de solutions…). Par contre, les informations sur la forme générée seront plus riches avec cette méthode. En effet, le seuil d'acceptation permet de déterminer les formes solutions mais ces solutions ne seront pas toutes équivalentes. Elles auront des degrés d'appartenance différents. Il y aura donc plus d'informations à fournir à l'utilisateur. Ce modèle permet en particulier de signaler que telle ou telle propriété est considérée comme suffisante mais qu'elle n'est pas idéale, etc. Il est ainsi possible de montrer les différences entre la scène proposée et la scène "idéale" par rapport à la description ou les autres scènes déjà proposées. Le seuil d'acceptation permet de faire une sélection mais laisse les différences entre les scènes choisies et donne la possibilité d'effectuer un premier classement.

Remarque : le modèle englobe les techniques classiques car on se retrouve dans une situation habituelle si on pose un seuil d'acceptation égal à 1. Ce qui montre qu'on n'a pas de coupure par rapport aux techniques habituelles mais que ce modèle est plus riche.

4.2. Les techniques de génération spécifiques

Les modèles flous possèdent des techniques de génération spécifiques pour la plupart regroupées sous le terme de « défuzzification » comme la technique dite de la moyenne pondérée, celle du maximum (choix de la plus grande valeur d'appartenance pour un concept) ou la méthode aléatoire (problème de transformation possibilité/probabilité où il s'agit de faire un tirage aléatoire d'une valeur prise dans une α-coupe elle-même déterminée aléatoirement [Yag82] et [DPS93]). Ces méthodes sont principalement axées sur la détermination d'une valeur d'un domaine en utilisant les fonctions d'appartenances des valeurs linguistiques la décrivant. Elles déterminent une seule valeur ou proposent un tirage aléatoire contrôlé uniquement par les degrés d'appartenance.

Le modèle flou permet de bien formaliser les propriétés et de mieux appréhender les formes générées. Des adaptations des techniques existantes sont aussi possibles dans un souci de bonne compréhension de la scène et, parfois, d'optimisation des performances de génération. En particulier, celles basées sur les systèmes à base de contraintes et d'arithmétique des intervalles semblent très intéressantes et peuvent exploiter de façon optimale le modèle flou.

4.3. Systèmes à base de tirage aléatoire sous contraintes

4.3.1 Méthodes de type CSP

Les techniques de résolution de contraintes de type CSP (Constraint Satisfaction Problem ; [Tsa93], [Fron94]) en modélisation déclarative sont celles proposées d'une part par [Don93] (utilisant la logique de Allen [All83]), et surtout, [ChM94a], [ChM94b], [Chau94b] et [MaM96] sur la réduction d'intervalles ([Sny92]) et d'autre part par [Lie96], [GAT96], [ZHH96], [PRJ97] et [Cham97a] avec des techniques CSP plus classiques sur le parcours et la réduction de domaines (souvent des intervalles). Ces méthodes de génération sont donc basées sur la manipulation d'intervalles. Or, la théorie des ensembles flous est construite comme un sur-ensemble des intervalles classiques. Elle indique que les opérations sur les intervalles classiques sont également valables sur les intervalles flous. Par conséquent, ces techniques peuvent être adaptées assez facilement à la manipulation des intervalles flous : par exemple en transformant les intervalles flous en intervalles classiques (en utilisant leur intervalle support ou toute autre α-coupe). Mais on peut aller plus loin en essayant d'adapter cette méthode de génération aux ensembles flous. Notons qu'il existe des travaux essayant d'associer d'un point de vue formel la théorie des ensembles flous avec les problèmes de contraintes ([Mat93]) et surtout avec les CSP qui sont alors appelés Fuzzy-CSP ou FCSP ([FDP95]).

4.3.2 Cas de la méthode de [Chau94b]

Cette méthode est une adaptation de ma méthode d'analyse des intervalles présentée dans [Sny92] et utilisée par [SnK92]. L'ensemble des paramètres et des contraintes constituent un pavé dans un espace IR^n. Avec notre modèle, nous n'avons plus un pavé mais une sorte de nuage. En effet, pour la méthode classique, si un point de l'espace appartient au pavé, il correspond à une solution. Ici, un point appartiendra à la solution à un degré près. Nous avons donc une sorte de nuage où les points appartiennent plus ou moins à la solution. La valeur en un point sera celle de l'évaluation de la description en ce point, c'est-à-dire le plus souvent le minimum des degrés d'appartenance aux différentes propriétés. Le calcul de la bande englobante se fait par rapport aux zones de densité non nulles ou supérieures à un seuil d'acceptation. Si besoin est, la technique de décomposition récursive reste valable.

Pour le tirage aléatoire d'un paramètre dans son intervalle, on peut utiliser la méthode classique. On peut aussi essayer de trouver un moyen de choisir une valeur pour laquelle le point de l'espace est à un extremum (local). Pour cela, soit une valeur est choisie selon une fonction spécifique, soit il faut mettre au point une méthode de tirage aléatoire « favorisant » les valeurs appartenant au noyau de la fonction d'appartenance. Pour cette dernière solution, la fonction d'appartenance est alors considérée comme une fonction de probabilité. Les techniques de défuzzification abordées au paragraphe 6.1 sont alors utilisables.

A ces dernières, on peut ajouter la méthode classique proche de celle de [Yag82] et utilisée par [Mou94] :

- on détermine la cardinalité C de la propriété (au sens de la théorie des ensembles flous) ;
- on fait un tirage aléatoire T (selon une loi normale) entre 0 et C ;
- on détermine le t en utilisant une intégration de la fonction suivant la méthode de Simpson et on intègre par parties jusqu'à atteindre T.

Remarque : cette intégration est facilitée par l'utilisation des fonctions LR en général assez facilement intégrables.

Cela permet de choisir une forme si possible la plus proche d'une solution "idéale" par rapport à la description. On essaye de choisir la meilleure possible a priori. On peut faire aussi un tirage aléatoire suivant la densité (plus probable dans les zones les plus denses). L'utilisation des ensembles flous permet une meilleure compréhension des zones connexes et surtout de mettre en évidence les différences par rapport à la description. L'idée principale est de rendre le tirage aléatoire le plus "intelligent" possible.

4.4. Systèmes à base d'arbres d'exploration

Il est clair que le modèle de représentation des propriétés fonctionne parfaitement pour tout ce qui est vérification a posteriori. La scène, une fois construite, est mesurée et la propriété est évaluée. L'utilisation du seuil d'acceptation est nécessaire ($S>0$ sinon toutes les formes seraient solutions et $S\leq 1$ avec $S=1$ pour la méthode classique). Les techniques d'élagage et de réduction des contrôles ([Des95a]) sont toujours applicables. Ces techniques ont aussi été affinées avec l'utilisation des ensembles flous dans [DeM97a].

4.5. Systèmes à base d'arbres de déduction

Là encore, le modèle fonctionne très bien pour les vérifications a posteriori. Il reste à étudier la possibilité d'utiliser la Logique Floue. Cette logique permettrait d'exploiter au mieux le modèle et de faire des déductions plus fines à l'aide de règles spécialisées. De plus, il semble que ces systèmes tendent vers la manipulation d'intervalles, en particulier à l'aide de fonctions d'intervalles ([MaM96]). Or, nous avons déjà vu au §6.2.2 que la manipulation d'intervalles est un domaine où notre modèle peut être particulièrement utile.

5. Prise de connaissance

Notre modèle présente un gros intérêt lors de la prise de connaissance. Il permet d'affiner la compréhension des solutions, d'améliorer les techniques existantes et de proposer de nouvelles techniques. nIl n'y a pas de changement par rapport aux techniques classiques. L'étude

du bon point de vue peut être affinée en permettant de choisir une vue en fonction de la propriété la mieux vérifiée ou en fonction d'autres critères de ce style.

Le système peut proposer une description de la scène par d'autres termes que ceux utilisés dans la description de l'utilisateur. Ces termes sont choisis pour être plus proches de la réalité. Autrement dit, le système décrit d'une autre manière la forme, il l'explique. Il propose une autre "opinion" qui permet une connaissance plus fine de la forme. Les termes sont choisis parmi les concepts les plus adaptés à la forme, parmi les propriétés ayant un degré d'appartenance le plus proche possible de 1.

Ainsi, on va pouvoir mettre en place des techniques d'apprentissage. Elles permettrons, par exemple, d'améliorer la description en fonction de l'avis de l'utilisateur pour augmenter les situations d'élagage. On imagine par exemple une méthode qui, à partir de l'avis de l'utilisateur sur les formes proposées, déduit les propriétés implicites et les ajoute à la description. Ainsi, une description plus riche permettra de mieux choisir les formes. Il est aussi tout à fait envisageable d'introduire une méthode d'apprentissage ajustant les paramètres des concepts et des différents opérateurs en fonction de ce que fait l'utilisateur et de ses avis sur les solutions proposées. Ainsi, le système adapte la sémantique de ses éléments à l'utilisateur.

6. Pour finir

Nous avons étudié un des aspects fondamentaux de la modélisation déclarative, à savoir la formalisation des propriétés, et ceci lorsqu'elles se réfèrent à des concepts imprécis ou vagues. Le cadre formel de la modélisation proposée a été construit à l'aide de modificateurs ou d'opérateurs flous définis sur des ensembles flous. Il apparaît clairement que le modèle proposé peut contribuer au développement formel de la modélisation déclarative en synthèse d'images dans un contexte d'informations imprécises ou vagues évaluées de façon quantitative ou qualitative. Nous avons aussi proposé une nouvelle méthode pour interpréter la négation dans une description. Cette interprétation linguistique basée sur la similarité propose un certain nombre de solutions plausibles à la négation d'une propriété élémentaire parmi lesquelles le locuteur choisit éventuellement la propriété qu'il sous-entend.

Cette étude, volontairement restreinte aux propriétés élémentaires dans un premier temps, montre bien que la théorie des ensembles flous est un outil tout à fait adapté, en modélisation déclarative, à la formalisation des propriétés comportant de l'imprécision et à l'interprétation linguistique de la négation.

Tous ces éléments ont été évalués à travers le projet LinéaFormes, reprise de l'application FiloFormes ([Paj94]), et font partie intégrante du noyau déclaratif du projet CordiFormes que nous présenterons dans la seconde partie.

Cependant, comme nous l'avons vu dans ces chapitres, il reste encore à préciser ou à améliorer le traitement de certaines propriétés. En effet, nous avons vu que le traitement des quantificateurs n'a pas été réellement effectué et que la solution présentée n'est pas suffisante. De plus, les solutions présentées correspondent à des cas particuliers qu'il faut s'attacher à étendre. Les propriétés élémentaires quantifiées présentées correspondent à un cas bien particulier. Comme pour les propriétés statistiques, elle ne prennent e compte que des groupes de concepts sélectionnés sur leur type. Elles ne permettent pas de traiter des énoncés de la forme « *La plupart des segments très longs ont une pente importante* » ou « *La longueur moyenne des segments de couleur claire est assez importante* ». De plus, ces quantifications et ces statistiques telles que nous les avons présentées, ne portent que sur des propriétés élémentaires. Il faux maintenant étudier leur traitement sur les autres types de propriétés.

De même, il serait intéressant de mettre en place un traitement des propriétés de composition plus complet permettant de prendre en compte les relations (parfois implicites) entre les opérandes. De plus, la formalisation des propriétés modificatrices proposée pose encore des problèmes en particulier au niveau de la cohérence. Il serait aussi intéressant d'affiner le traitement des propriétés de comparaison en essayant de prendre plus en compte des aspects linguistiques et psychologiques.

La négation n'a été présentée que pour les propriétés élémentaires. Il sera intéressant d'étudier leur traitement pour les autres types de propriétés comme les relations ou les compositions et leur comportement vis-à-vis des quantificateurs.

Actuellement, nous n'avons proposé aucune étude de la cohérence exploitant les ensembles flous. Cette étude n'est pas forcément facile, en particulier pour une description en langage naturel. Il faut donc s'intéresser à des systèmes permettant de gérer les problèmes de cohérence, d'ambiguïté et de connaissances par défaut comme celui proposer pour l'application NALIG ([ADG84], [DAG86], [GFT92] et [GAT96]).

Enfin, il serait très intéressant d'étudier plus précisément les CSP flous qui pourraient proposer des méthodes très intéressantes de génération utilisant efficacement notre modèle.

PARTIE II : LE PROJET CORDIFORMES

CHAPITRE II.1 : PRÉSENTATION DU PROJET

1. Introduction

La modélisation déclarative s'attache à construire un ensemble de solutions correspondant à une description effectuée à l'aide de divers médias. Depuis [LMM89], un certain nombre de projets ont vu le jour. Ils ont permis de mettre en évidence des connaissances et des techniques indispensables pour la construction d'un modeleur déclaratif. Citons par exemple les projets déjà présentés *PolyFormes*, *AutoFormes*, *VoluFormes*, *CurviFormes*, *MégaFormes*, *FiloFormes* mais aussi *SpatioFormes* (modélisation à l'aide de matrices d'énumération spatiales [Pou94a] et [Pou94b]), *MultiFormes* (génération de scènes par des descriptions complexes et structurées [Ple91]), *PastoFormes* (description d'objets obtenus par collages de polyèdres élémentaires [Col90] et [Col92]), *BatiMat* (description de maisons [Cham97a] et [Cham97b]), *FLATS* (description d'appartements [Koc94]), « *La Have House* » (description de maisons [RMS96] et [RMD96]), *GENWIN* (description de fenêtres [Khe95])… Ces projets ont tous en commun de traiter un sujet spécifique et ne sont pas directement adaptables à d'autres sujets. Cependant, la plupart d'entre eux mettent en œuvre des techniques similaires. Nous nous sommes donc intéressés à la mise en oeuvre d'un ensemble d'outils, une plate-forme, visant à faciliter la mise en oeuvre de futurs modeleurs déclaratifs : c'est *le projet CordiFormes*.

Afin d'éviter toute confusion, posons les deux définitions suivantes.

Définition III.1.1 : Nous appellerons *concepteur* une personne construisant un modeleur déclaratif.

Définition III.1.2 : Nous appellerons *utilisateur* une personne utilisant un modeleur déclaratif pour construire les formes qu'il désire.

Dans ce chapitre, nous présenterons les grands objectifs de notre projet CordiFormes (section 2). Puis, nous exposerons la méthode de construction d'un modeleur déclaratif à partir de notre plate-forme (section 3). Enfin, nous verrons les thèmes abordés dans les prochains chapitres de cette partie II.

2. Le projet CordiFormes

2.1. Contexte et objectifs

Un certain nombre de projets ont vu le jour, chacun visant à étudier des aspects particuliers de la modélisation déclarative. Bien que différents, ils mettent en œuvre des techniques similaires. Les études récentes ([Des95b], [LuD95], [CDMM97b] et les réflexions du groupe GEODE) s'appliquent à faire un bilan et une synthèse des travaux en modélisation déclarative. Les réflexions se portent essentiellement sur la définition des éléments fondamentaux d'un modeleur, leur utilisation ainsi que sur la mise en place d'outils permettant de les exploiter. L'attention se porte aussi sur la mise en place de modèles de génération permettant de rendre compte des modes de conception à l'aide de modeleurs déclaratifs ([CDMM97b] et [CDMM97c]). Il devient donc nécessaire de formaliser les connaissances manipulées et de proposer une plate-forme pour la construction de modeleurs déclaratifs exploitant ce formalisme et proposant des outils souples et génériques. Par conséquent, nous nous sommes intéressés à la mise en œuvre d'un ensemble d'outils, une plate-forme, visant à faciliter le développement de futurs modeleurs déclaratifs : le projet CordiFormes.

Pour mettre en place l'unification de ces outils, nous définissons d'abord un modèle de représentation des connaissances que nous avons présenté dans la partie I. Celui-ci est construit comme une synthèse, une généralisation, de modèles présentés dans les différentes études. Il trouve donc naturellement sa place dans une plate-forme comme support de l'ensemble des connaissances du modeleur et base essentielle pour des méthodes de génération performantes et générales. De ce point de vue, CordiFormes est actuellement plus un *noyau déclaratif* qu'une plate-forme (dans le sens « environnement de programmation de haut niveau »). Notre objectif est de proposer une base de connaissances et d'outils permettant de développer facilement un modeleur déclaratif indépendamment de la machine utilisée. Le concepteur peut exploiter des méthodes générales et des objets déjà implémentés ou proposer les siens. Les modeleurs produits à l'aide de CordiFormes sont des applications indépendantes mais aussi des modules que le concepteur peut greffer sur d'autres applications. Celui-ci peut créer facilement et rapidement un modeleur déclaratif. Il lui suffit de fournir les éléments spécifiques à son domaine d'application comme les objets manipulés, les algorithmes de construction et les caractéristiques (ou concepts) de la scène. Éventuellement, il les ajuste en fonction de ses propres besoins.

2.2. Caractéristiques attendues

Les objectifs que nous nous sommes fixés pour CordiFormes imposent un certain nombre de caractéristiques pour la plate-forme. Celles que nous attendons sont entre autre : *la simpli-*

cité, *la souplesse* de programmation, *l'efficacité*, *l'extensibilité*, la *réutilisabilité*, le *prototypage rapide* du modeleur.

2.2.1 La simplicité

Un des objectifs essentiels de CordiFormes est de rendre simple la conception d'un modeleur. L'idée est d'éviter au concepteur d'être noyé dans une multitude de paramètres à déterminer et de choix possibles. Dans le cas le plus simple, il se contente d'indiquer les objets de son domaine d'application. Il peut ensuite immédiatement décrire une scène, générer les solutions et en prendre connaissance. Il n'est plus obligé de s'occuper de la génération ou de la définition des propriétés de base. CordiFormes utilise pour cela, nous le verrons plus en détails au chapitre II.2, un mode de génération dit récursif. Pour la description, le concepteur bénéficie du calcul automatique des propriétés de base dans le cas des propriétés de niveau 1 (comme nous l'avons présenté dans la partie I). Il n'a pas besoin de définir les propriétés de niveau supérieur, construites automatiquement à partir de concepts ou de propriétés de niveau 1. Bien évidemment, s'il le désire, le concepteur peut ajuster, modifier ou refaire certains éléments qui ne lui plaisent pas.

2.2.2 La souplesse

Notre objectif essentiel est de laisser le concepteur libre pour la création de son application. Il doit notamment pouvoir construire un modeleur déclaratif contenant ses propres méthodes de génération même si c'est aux dépens de la performance du modeleur (contrôle du parcours des solutions, étude de toutes les solutions…). La plate-forme lui fournit des outils spécifiques adaptés aux problèmes de la modélisation déclarative, à lui d'en faire ce qu'il désire. Hormis des raisons de performance que nous aborderons plus loin, cette souplesse est importante pour pouvoir ajuster le modeleur à ses propres besoins.

2.2.3 L'extensibilité

Les structures et les algorithmes proposés sont suffisamment généraux pour pouvoir s'appliquer à la plupart des problèmes posés. Par conséquent, ces outils ne sont pas toujours parfaitement adaptés aux problèmes du concepteur. Celui-ci doit donc être en mesure redéfinir totalement ou partiellement les éléments de CordiFormes. Néanmoins, il faut aussi garantir une utilisation cohérente de ces outils afin de s'assurer du bon fonctionnement des éléments clés. De plus, il est difficile de prévoir convenablement tous les cas spécifiques à chaque domaine d'application et à chaque objectif du concepteur. Il y aura donc nécessairement des éléments qu'il faudra adapter au problème.

2.2.4 L'efficacité

L'utilisation d'une plate-forme a un inconvénient : les algorithmes utilisés, en particulier les algorithmes de génération, sont généraux (nous les appellerons systématiques) et ne sont pas forcément optimisés pour les problèmes traités. Le concepteur doit donc pouvoir proposer ses propres outils et notamment des techniques de génération spécifiques aux concepts utilisés. Toutefois, bien que plus performantes, elles risquent de limiter certaines fonctionnalités spécifiques à la génération systématique.

2.2.5 La réutilisabilité

Les concepts manipulés par les modeleurs déclaratifs sont généralement spécifiques a un domaine d'application. Cependant, un certain nombre d'entre-eux sont fréquemment utilisés par la plupart des modeleurs. CordiFormes propose donc une base de connaissance de concepts courants et d'outils utilisés qui constituent l'ontologie ([Gru93a], [Gru93b], [GuG95], [CBB96]) de la modélisation déclarative et de la modélisation géométrique [DeM97b]. Cette base permet au concepteur de porter son attention uniquement sur les objets spécifiques du domaine d'application. Il peut éventuellement l'enrichir au fur et à mesure de ses développements.

Remarques :

- « *Le terme "Ontologie" vient de la philosophie où il désigne "la partie de la métaphysique qui s'intéresse à l'Etre en tant qu'Etre" (Petit Robert). Mais l'Ontologie est habituellement davantage comprise comme étant une science des étants que comme une science de l'Etre en tant qu'Etre, c'est-à-dire qu'elle s'intéresse davantage à ce qui existe (les étants ou existants) qu'aux principes de ce qui existe (l'Etre). Cette science, l'Ontologie, produit des ensembles, les ontologies.* » [CBB96] ;
- « *Une ontologie est l'ensemble des objets reconnus comme existant dans le domaine. Construire une ontologie d'un domaine, c'est donc décider quels sont les objets que l'on retient comme existant, c'est-à-dire décider quels objets possèdent une consistance ontologique, et lesquels n'en ont aucune. Construire une ontologie, c'est aussi décider de la manière d'exister des objets retenus comme possédant une consistance ontologique…* » [CBB96].

2.2.6 Outils d'interface et de prototypage

La majeure partie des publications en modélisation déclarative concerne les outils de génération. Toutefois, ceux-ci restent difficilement utilisables sans de bonnes techniques de description et de prise de connaissance. Il est donc important d'inclure dans CordiFormes des outils d'interface utilisant ces techniques. Ceux-ci ne sont pas indispensables au modeleur déclaratif mais permettent de l'exploiter au mieux.

Lorsque le concepteur a mis en place tous ces éléments, il doit pouvoir évaluer, tester le modeleur qu'il a construit. La plate-forme doit donc donner la possibilité de développer rapidement une application prototype. Celle-ci pourra utiliser tous les outils disponibles dans la plate-forme (saisie de description et de prise de connaissance, gestion de la génération...).

2.2.7 Choix de Java comme langage de programmation

Lors de la mise en place de cette plate-forme s'est posé le problème du choix du langage de programmation à utiliser. La plate-forme doit pouvoir engendrer des applications qui tourneront sur plusieurs types de système (stations Unix, PC ou Macintosh). Les caractéristiques d'efficacité, d'extensibilité et de réutilisabilité sous-entendent naturellement l'utilisation d'un langage basé sur la programmation orientée objet. Ces remarques nous ont conduit au choix du langage de programmation multi-systèmes Java™ ([ArG96], [Fla96], [GoY97a] et [GoY97b]) pour développer CordiFormes. Outre les caractéristiques citées ci-dessus, Java, avec ses outils d'encapsulation, nous permet d'obliger la définition de méthodes (grâce aux méthodes abstraites), d'autoriser ou d'interdire la surcharge de certaines autres (méthodes classiques ou méthodes dites finales). Ce langage possède toutefois un inconvénient : son manque d'efficacité (temps d'exécution). Java, plutôt de la classe des langages interprétés mais utilisant une technique de compilation « Just In Time » (à l'exécution) a une efficacité moindre qu'un langage compilé. Cependant, il est actuellement en pleine évolution, ce qui laisse envisager un accroissement des performances. Déjà plus rapide que les langages interprétés « classiques », il bénéficiera bientôt de processeurs spécifiques.

2.3. Les trois couches de CordiFormes

Les caractéristiques que nous avons définies dans les paragraphes précédents nous ont conduit à structurer CordiFormes en trois couches (Figure 74) :

1. *la couche noyau*. Cette couche est composée d'algorithmes et de structures classiques que le concepteur peut éventuellement redéfinir et paramétrer pour créer son modeleur déclaratif. Elle contient également la bibliothèque d'objets et de méthodes réutilisables formant l'ontologie de la modélisation déclarative et de la modélisation géométrique. Cette bibliothèque sera remise à jour par le concepteur qui pourra y introduire ses propres concepts et méthodes de génération qu'il voudra pouvoir réutiliser pour d'autres applications.
2. *la couche interface*. Cette couche regroupe un ensemble de dialogues standard permettant la description (sélection des objets, construction de la description à l'aide de propriétés...), la génération et la prise de connaissance (affichage des solutions, mise en évidence des propriétés, sélection de bons points de vue...) et d'autres fenêtres ou diverses parties de dialogues.

3. *la couche prototype*. Cette couche permet de produire un premier prototype du modeleur escompté. Elle utilise les outils des couches interface et noyau. Elle correspond à un modeleur déclaratif minimal permettant de faire une saisie de description, de générer les scènes correspondantes et d'en prendre connaissance.

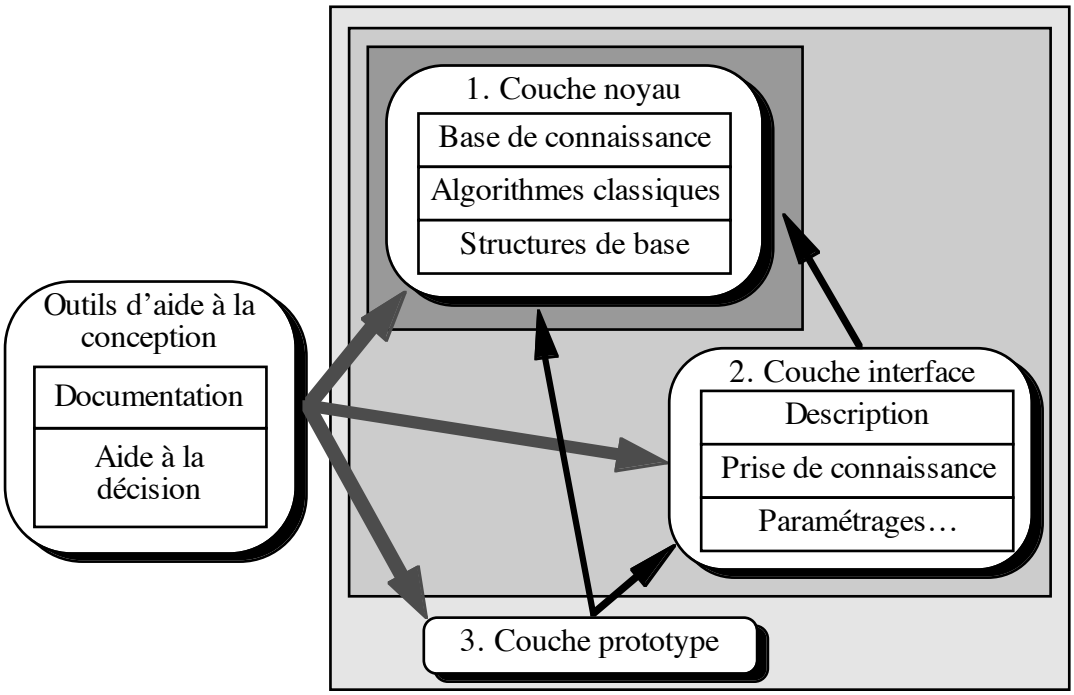

Figure 74. Niveaux d'utilisation et éléments constitutifs de CordiFormes

Le concepteur peut utiliser l'intégralité ou seulement une partie de cette plate-forme. Il utilise les éléments de la couche noyau pour construire le modeleur déclaratif (en particulier tout ce qui concerne la phase de calculs). S'il en ressent le besoin, il peut se servir des outils de la couche interface pour saisir des éléments de description, prendre connaissance des solutions… Pour évaluer son modeleur, il lui suffit d'utiliser l'application type de la couche prototype.

En plus des outils directement liés à la construction du futur modeleur, la plate-forme doit proposer un ensemble d'outils d'aide à la conception du modeleur ([DeT97]). L'objectif est de proposer des outils de haut niveau permettant d'aider le concepteur à comprendre et à préciser son problème, à concevoir son application et à valider ses choix. Ces outils (systèmes experts, systèmes d'aide à la décision, systèmes de validation…) permettent de guider le concepteur dans les choix des techniques à utiliser et les éléments essentiels à programmer. Le but est d'obtenir un système minimal et de produire rapidement une maquette de son application. Nous verrons dans le chapitre de conclusion de cette partie quelques idées pour de tels systèmes.

3. Construire une application déclarative

Détaillons maintenant le processus de conception d'un modeleur déclaratif. Au départ, le concepteur arrive avec son problème : concevoir un modeleur déclaratif pour construire des objets dans un domaine d'application spécifique. Plutôt que de refaire tous les outils, la plate-forme CordiFormes lui en propose un certain nombre. Nous supposerons ici que le concepteur désire construire une application complète, c'est-à-dire qu'il utilise au maximum les outils mis à sa disposition. La conception se déroule en huit phases (selon l'application, certaines ne sont pas indispensables) :

1. Construction des objets de la scène :
   - création des concepts ou reprise (ou modification) de concepts de la base de connaissance,
   - détermination de la sémantique et des caractéristiques des domaines ainsi que des propriétés de base utilisables ;
2. Construction du concept « scène » : sémantique et paramètres globaux ;
3. Mise en place des relations entre les objets de la scène (contraintes sémantiques,…) ;
4. Construction des concepts liés à des caractéristiques globales ;
5. Détermination du mode de génération désiré ;
6. Implémentation des méthodes spécifiques de génération si les méthodes standard ne sont pas suffisantes ;
7. Mise en place des dialogues utiles ;
8. Utilisation de l'application prototype.

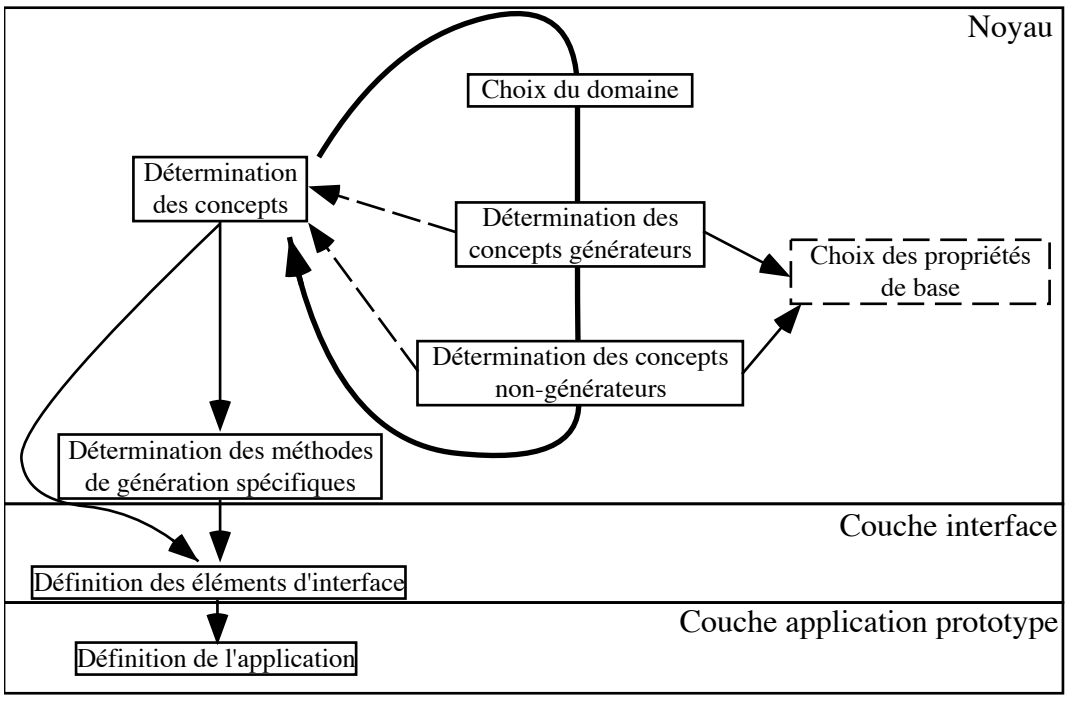

Figure 75. Conception d'un modeleur déclaratif

Les phases essentielles sont les quatre premières. Elles définissent la sémantique liée au domaine d'application. Elles constituent la base de connaissance du modeleur déclaratif. Tous les éléments que nous avons rencontrés dans ces phases seront détaillés dans les chapitres qui suivent.

4. Conclusion

Les caractéristiques essentielles attendues pour notre projet CordiFormes sont : la simplicité, la souplesse, l'efficacité, l'extensibilité, la réutilisabilité et le prototypage aisé. Cette plate-forme est constituée de trois couches : le noyau, les éléments d'interface et l'application prototype. A l'aide de ces outils ainsi que ceux d'aide à la conception, un concepteur doit pouvoir construire assez facilement un modeleur déclaratif de son choix.

Dans la suite, nous étudierons d'abord les éléments principaux du noyau, en particulier, les concepts, les tâches de génération qui leurs sont associées et les contraintes (chapitre II.2). Ces éléments sont issus de la notion de concept de la partie précédente. Nous verrons comment est organisée la structure de la scène et comment, à partir de celle-ci, mettre en place une méthode de génération générale.

Puis, nous aborderons tous les types de propriétés, éléments de la description, leur structure et les optimisations que l'on peut en tirer (chapitre II.3). Nous retrouverons toutes les propriétés présentées dans la première partie. Nous exploiterons leur formalisation pour les implémenter et proposer des optimisations possibles de la génération.

Le chapitre II.4 sera consacré à la présentation d'autres éléments importants du noyau : les modules de description, de génération et de prise de connaissance. Ces éléments sont chargés d'exploiter les différentes connaissances et les différents éléments de la plate-forme par rapport aux grandes phases d'un modeleur déclaratif.

Ensuite, nous compléterons la présentation du projet par l'étude des deux autres couches : la couche interface et la couche application prototype (chapitre II.5).

Le chapitre II.6 sera consacrée à la présentation de trois applications développées à partir de CordiFormes. Elles sont simples mais permettent de mettre en avant les différentes caractéristiques de notre plate-forme. Elles soulignent toutes la simplicité de développement où le concepteur n'a pas à s'occuper à mettre en place toutes les techniques de génération nécessaires.

Enfin, nous conclurons cette partie par l'introduction aux outils d'aide à la conception et d'apprentissage qui devront être développés afin de faciliter le travail du concepteur (chapitre II.7).

CHAPITRE II.2 : NOYAU ET GÉNÉRATION DE LA SCÈNE

1. Introduction

Après avoir présenté les grandes lignes du projet CordiFormes, nous allons nous intéresser aux éléments constitutifs du noyau. Nous étudierons d'abord les éléments les plus importants de ce noyau : *les concepts*. Ce sont eux qui représentent essentiellement le coeur de la plateforme, car ils dépendent fortement du domaine d'application. Ce sont ces éléments que le concepteur doit « travailler » en priorité. Pour simplifier la manipulation des différents éléments de ce noyau, nous avons choisi d'utiliser une représentation orientée objet. Le concepteur se contente alors de surcharger certaines méthodes importantes, de donner une valeur à certaines variables ou de les modifier. Il peut aussi ajouter des variables qui lui semblent nécessaires.

Les éléments constitutifs du noyau sont directement issus du formalise exposé dans la partie I de ce document. Tout le noyau est construit autour de l'implémentation des notions de concept, domaine et propriété.

Dans un premier temps, nous allons élargir et détailler la notion de concept vue dans la première partie en considérant que tous les objets ou parties d'objets manipulés sont des concepts (section 2), c'est-à-dire aussi bien les caractéristiques « descriptibles » que les objets intermédiaires (ébauches) ou finaux. Dans la section 3, nous regarderons comment ces concepts sont structurés par rapport à l'univers du domaine d'application. Nous étudierons ensuite la génération de ces objets à l'aide de tâches de génération. Après avoir présenté des méthodes plus classiques (section 4), nous étudierons la génération récursive. C'est une méthode de génération systématique indépendante du domaine d'application. Cependant, une méthode automatique n'est pas toujours adaptée du point de vue de la performance (nombre de solutions explorées, temps pour construire une solution…). Nous verrons donc dans la section 6 le moyen de personnaliser la génération en introduisant localement des tâches de génération plus spécifiques. Puis, après nous être intéressés à la génération d'un objet, nous étudierons l'organisation de la génération des objets afin de produire une scène (section 7). Toutes ces méthodes supposent que les concepts sont indépendants entre eux. Or, cette hypothèse n'est pratiquement jamais vérifiée. Il est indispensable d'introduire des contraintes de construction permettant de limiter le nombre d'objets invalides dès la construction. L'introduction et la gestion de ces contraintes seront présentées à la section 8. Nous finirons ce chapitre par une étude des conséquences de l'utilisation de ces contraintes sur la génération.

2. Les concepts de la scène

Un concept est un objet ou une caractéristique de la scène. Une maison, la densité d'habitations, la couleur d'un vélo, la puissance d'une voiture, le nombre d'intersections entre objets, la hauteur d'un menhir… sont des concepts.

Définition II.2.1 : Nous appellerons *concept* une caractéristique ou un objet de la scène.

Parmi tous ces concepts, nous distinguons les concepts terminaux, les concepts simples et les concepts complexes.

2.1. Le domaine

Un domaine est un intervalle classique auquel on ajoute une unité (Définition I.1.2) qui peut être nulle (cas du domaine continu). Le domaine comprend des méthodes permettant d'ajuster une valeur en fonction de son unité, de donner la valeur suivante ou précédente d'une valeur donnée… La Figure 76 illustre la définition d'un domaine du point de vue programmation orientée objet.

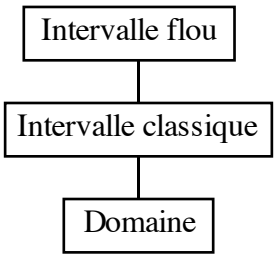

Figure 76. Hiérarchie d'un domaine

2.2. Concept terminal

Définition II.2.2 : un *concept terminal* est un concept ne faisant référence qu'à un domaine (liste de valeurs possibles, unité…).

Les concepts terminaux représentent les caractéristiques de la scène ou des composants d'objets de la scène. Ce sont eux qui sont généralement utilisés pour décrire une scène. Ils correspondent strictement à la mise en œuvre des concepts présentés dans la partie I. Nous leur associons donc un domaine $[Bm, BM]_u$ et un ensemble de propriétés de base (comme par exemple : {}, {vérifié}, {faible, moyen, important} ou {petit, moyen, grand}).

Par exemple, à la Figure 77, « Rouge », « Vert » et « Bleu » sont des concepts terminaux dont les domaines sont $[0,255]_1$. De même, à la Figure 78, « Texture » est un concept terminal dont le domaine est {Pierre, Bois, Brique}.

2.3. Concept non-terminal

Les concepts terminaux ne sont pas suffisants pour représenter des notions plus complexes comme une couleur, un segment, un cube ou un navire. Nous proposons donc d'étendre la notion de concept à toutes les caractéristiques, et surtout, tous les objets de la scène. Ces concepts sont, la plupart du temps, construits à partir d'autres concepts dont les concepts terminaux.

Définition II.2.3 : un *concept non-terminal* est un concept défini par un ensemble de concepts appelés *concepts composants*.

Parmi ces concepts, on distingue les concepts simples et les concepts complexes.

2.3.1 Concept simple

Définition II.2.4 : un *concept simple* est un concept défini uniquement par un ensemble de concepts terminaux.

Les objets élémentaires (les plus simples) de la scène sont généralement des concepts simples. Ils seront appelés *objets simples*. Par exemple, la couleur présentée à la Figure 77, est un concept simple.

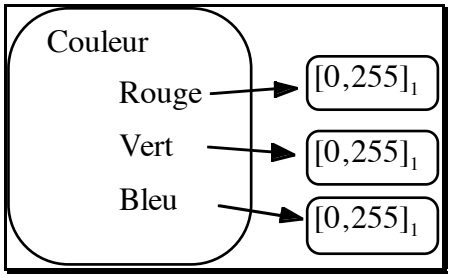

Figure 77. Exemple de concept simple

Remarque : Les concepts terminaux servent uniquement à construire les concepts simples. Cependant, il n'est pas possible de construire des éléments plus élaborés seulement avec des concepts simples. Nous introduisons donc les concepts complexes.

2.3.2 Concept complexe

Définition II.2.5 : un *concept complexe* (ou *objet complexe*) est défini par un ensemble de concepts terminaux, simples et complexes.

Par exemple, une maison est composée d'un toit et de murs. Le toit possède une forme, une hauteur, une couleur… Les murs sont définis par une hauteur, une texture… (Figure 78). Cependant, un concept complexe n'est pas forcément un objet de la scène. Par contre, les objets

d'une scène sont des concepts complexes. Ils sont composés de concepts simples ou de concepts complexes.

Définition II.2.6 : Construire un objet complexe à partir de concepts complexes ou simples s'appelle une *composition par définition*.

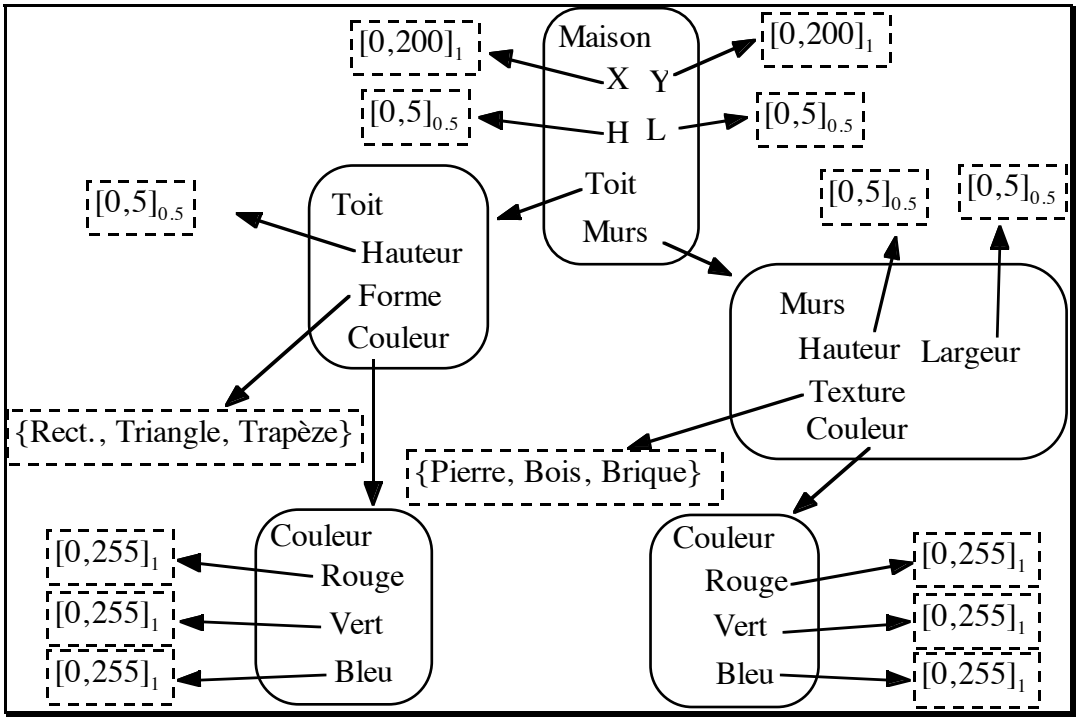

Figure 78. Exemple de concept complexe

2.3.3 Liste de concepts

A ces concepts nous pouvons en ajouter un autre dont la gestion est un peu particulière : la liste de concepts.

Définition II.2.7 : Une *liste de concepts* est un concept non-terminal dont le nombre de concepts composants n'est pas fixé par le concepteur mais par la description de l'utilisateur.

La liste est un concept particulier dont l'étude sera précisée dans l'avenir.

2.4. Structure d'un concept

Généralement, conformément au paragraphe précédent et à ce qui a été présenté dans la partie I, un concept est composé de (Tableau 8) :

- un *nom*, utile en particulier pour pouvoir le décrire ;
- un *domaine* contenant, comme nous l'avons déjà vu, l'ensemble des valeurs possibles du concept associé ;
- une *méthode de vérification de validité* ;

- une *fonction de mesure* permettant de « construire » le concept, c'est-à-dire de déterminer la valeur courante du concept appelée *mesure* (cette mesure peut être vue comme *une instance* de la classe d'objets ou de valeurs représentée par le concept).

Tableau 8. Définition d'un concept

Nom : Concept **Mère** :	
Principaux champs	**Principales méthodes**
Nom	Vérifier validité
Domaine	
Mesure	
Fonction de mesure	

Remarques :

- En plus de ces éléments, un concept doit être capable de produire sa représentation dans un modèle géométrique donné.
- Pour les concepts simples ou complexes, le domaine peut être indéterminé, car les valeurs du concept ne sont pas énumérables ou sont trop nombreuses. La valeur de cet objet est déterminée par construction ou par mesure. Il convient donc que de tels concepts comportent une méthode de vérification de validité. Ainsi, le domaine est défini en intention (par cette méthode) et éventuellement en extension (définition du domaine). Par défaut, cette méthode peut, par exemple, se contenter de vérifier que la valeur courante est bien dans le domaine.

A cette structure de base, les concepts non-terminaux ajoutent une *liste de concepts composants* et les concepts terminaux une *liste de propriétés de base*.

Tous les concepts que nous venons de présenter peuvent être hiérarchisés du point de vue de la conception orientée objet (Figure 79).

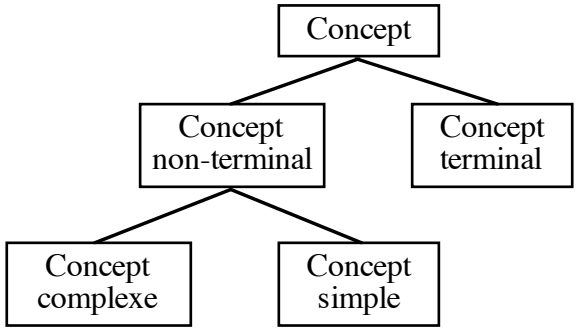

Figure 79. Hiérarchie des concepts

Remarques :

- Les concepts non-terminaux ne peuvent avoir de propriétés de base car celles-ci sous-entendent que le domaine est ordonné. Or, par définition, les concepts non-terminaux

sont multidimensionnels et donc ne peuvent pas être ordonnés globalement. Par contre, les concepts terminaux qui les composent comportent des propriétés de base.

- Les concepts ne sont pas toujours des objets ou des constituants d'objets. Ils peuvent servir à décrire la construction d'objets ou correspondent à des relations entre au moins deux concepts (par exemple, les concepts supports aux propriétés relatives et plus généralement aux relations).

2.5. Construction d'un nouvel objet de la scène

Lorsque le concepteur désire créer un nouvel objet ou une nouvelle caractéristique pour son modeleur, il lui suffit de surcharger (d'étendre) un concept défini dans le noyau. Ainsi, il donne la sémantique de son objet. Par exemple, s'il désire construire un concept « couleur », il créera une sous-classe d'un concept simple. Pour cela, il ajoutera dans la liste des concepts composants, les concepts terminaux « Rouge », « Vert » et « Bleu » définis sur des domaines de la forme $[0, 255]_1$ (Figure 80). Une fois définie la sémantique de l'objet, il doit aussi déterminer la fonction permettant d'obtenir l'objet dans un modèle géométrique adapté.

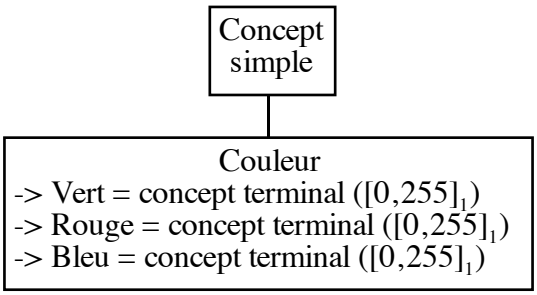

Figure 80. Construction d'un concept "couleur"

3. Organiser la description

A partir des concepts définis par le concepteur, il est possible de construire les propriétés permettant de décrire une scène. Dans cette section, nous étudierons les éléments décrits. Les propriétés composant la description seront abordées dans le chapitre II.3.

3.1. Deux descriptions

Lorsque l'utilisateur décrit une scène, les propriétés portent évidemment sur les objets qu'il a choisis ainsi que sur leurs relations. En fait, la description se déroule en deux phases : la sélection d'un ou plusieurs objets puis les descriptions de leurs propriétés. Par conséquent, à un instant donné, les propriétés de la description ne peuvent concerner, directement ou indirectement, que des objets déjà décrits par l'utilisateur.

3.2. Objets de la description

La description est composée d'un ensemble de propriétés. Celles-ci font référence à des concepts présents ou qui seront présents dans la scène. Le plus souvent, il s'agit de concepts terminaux d'objets décrits ou de concepts terminaux indiquant des relations entre ces objets. Ils sont organisés de manière hiérarchique.

Pour conserver les objets sélectionnés par l'utilisateur, il nous faut une *table des objets*. Celle-ci doit contenir l'ensemble des objets de la scène ainsi que certaines informations utiles (Tableau 9). Notamment, elle garde une trace de la hiérarchie entre les objets décrits (champs « père », « fils » et « frère »). Cette hiérarchie est d'ailleurs contrôlée par les graphes sémantiques que nous verrons dans la section 9. « Niveau » indique le niveau de l'objet dans la hiérarchie. Les objets décrits sont des concepts simples ou complexes, c'est-à-dire des concepts non-terminaux.

Tableau 9. Constitution de la table des objets

Champ	Numéro	Nom	Type	Père	Fils	Frère	Niveau
Type	Entier	Chaîne	Concept	Entier	Entier	Entier	Entier

Il existe un objet particulier : la « scène ». Elle constitue la racine de la hiérarchie des objets. Son niveau est 0. Elle n'a ni père ni frère. C'est un concept simple ou complexe déterminé par le concepteur. Elle est toujours présente dans la table avant même le début de la description. Les concepts terminaux qui la composent donnent en général les paramètres globaux pour les objets de la scène comme, par exemple, la forme de l'espace…

Associées à cette table, nous avons deux listes : la description de l'utilisateur et les concepts « indépendants ». Ces derniers décrivent les relations entre des objets. Ils sont à la base des propriétés que nous appellerons propriétés relatives. Comme ils ne sont pas liés à un objet donné, ils sont regroupés dans une liste spécifique. De même, la description totale de la scène est conservée pour être utilisée par la suite au niveau de la génération et de la prise de connaissance. Ainsi, elle permet non seulement de rappeler la description mais aussi de calculer le degré d'appartenance globale de la scène à la description.

3.3. Graphes sémantiques

Il est indispensable de poser des règles permettant de contrôler ce qu'un utilisateur, construisant sa scène, a le droit de faire ou de ne pas faire. Pour cela, le concepteur doit mettre en place un ensemble de graphes sémantiques.

Définition II.2.8 : Nous appellerons *graphe sémantique* un graphe permettant de représenter des règles de constructions autorisées dans une scène. « Est_un » et « Est_Composé_de » sont les uniques opérateurs disponibles.

Définition II.2.9 : Cette composition, par opposition à la composition par définition, sera appelée *composition par construction*.

Par exemple (Figure 81a) :

- Est_Composé_de(Scène, Forme Géométrique) ;
- Est_un(Zone, Forme Géométrique) ;
- Est_un(Rectangle, Forme Géométrique) ;
- Est_un(Cercle, Forme Géométrique) ;
- Est_un(Segment, Forme Géométrique) ;
- Est_Composé_de(Zone, Forme Géométrique).

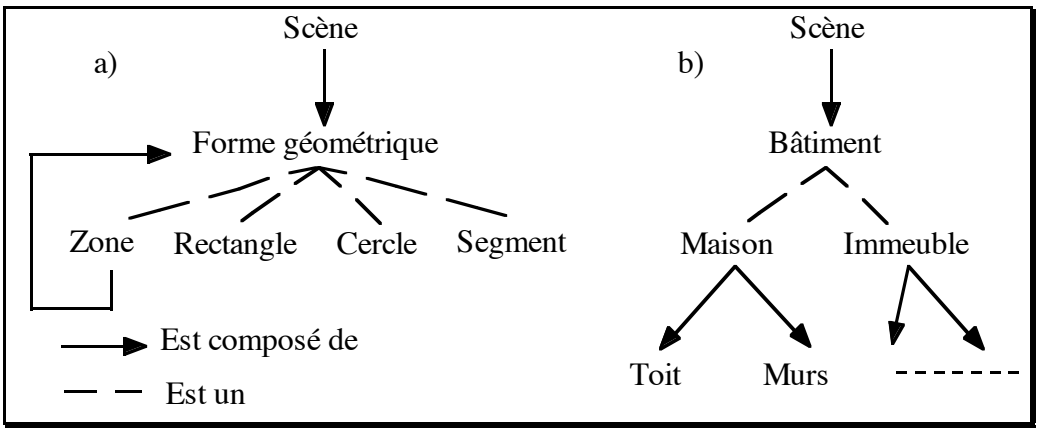

Figure 81. Exemples de graphes sémantiques

Selon le choix du concepteur, un même objet peut être construit par définition (Figure 78) ou par construction (Figure 81b).

4. Générer les objets

4.1. Classes de génération

Les différents travaux réalisés en modélisation déclarative sont généralement classés suivant *les moyens utilisés*. Ce classement est globalement le suivant :

- Arbres d'exploration explicites tels que les arbres d'énumération et autres algorithmes de parcours d'arbres ([Col90], [Mar90], [Paj94], [Khe95], [PoL96]…) ;
- Moteurs d'inférences ([Ple91] et [MaM93]) ;
- Grammaires génératives ([Woo91], [Koc94] et [RMS96]) ;
- Systèmes de résolution de contraintes ([Lie96], [GAT96], [ZHH96], [PRJ97], [Cham97a]…) ;
- Systèmes d'arithmétique des intervalles ([SnK92], [Chau94] et [MaM96]) ;
- Simulation inverse ([KaB92] et [SDS93]) ;
- Autres méthodes telles que les algorithmes génétiques ([Mou96])...

Remarque : Ce classement et les méthodes de génération qui le composent sont présentés plus en détail dans [Des95a].

Dans le cadre du projet CordiFormes, la classification adoptée repose non pas sur les techniques de génération utilisées mais sur les *objectifs de la génération et des effets pour l'utilisateur*. Dans ce contexte, seules deux classes de génération de scènes sont retenues à savoir *l'énumération* et le *tirage aléatoire sous contraintes*. Ainsi, toutes les méthodes développées en modélisation déclarative peuvent être réparties selon ces deux grandes classes.

Définition II.2.10 : Une génération est dite *par énumération* si, étant donné un objet k et l'état de la base de connaissances courante, il est possible de prédire avec exactitude ce que sera l'objet k+1. Le passage d'un objet (ou d'une valeur) à un autre est totalement contrôlé.

Définition II.2.11 : Une génération est dite *aléatoire* si elle n'est pas par énumération, c'est-à-dire s'il n'est pas possible de prédire ce que sera l'objet k+1 connaissant l'objet k.

Remarques :

- Ces dernières définitions peuvent être assouplies en ne considérant que certains concepts jugés comme caractéristiques.
- Éventuellement, le domaine, s'il est continu, doit être préparé pour une énumération. Il s'agit alors de le discrétiser afin de le rendre fini. Cette opération s'appelle *l'échantillonnage du domaine*.
- Dans le cas de la génération aléatoire, pour proposer toutes les solutions, la méthode de génération doit interdir les situations déjà produites.

Par ailleurs, nous avons montré que certains concepts sont composés d'autres concepts. Selon l'utilisation ou non de cette hiérarchie par l'algorithme de génération, nous pouvons mettre en évidence deux classes d'algorithmes :

- les algorithmes directement basés sur l'objet en tant que tel (génération globale, algorithmes généraux ou spécifiques à l'objet) ;
- les algorithmes exploitant la hiérarchie en faisant appel à une *génération récursive* (algorithmes que nous appellerons aussi *algorithmes systématiques*).

Les algorithmes de la première classe, par nature, sont dépendants du domaine d'application considéré. Par contre, les algorithmes systématiques sont indépendants. Notre attention se portera plus particulièrement sur ces derniers.

4.2. Méthodes sur les objets

Actuellement les plus répandues sont les méthodes de génération globales à l'objet. Elles sont basées sur des algorithmes considérant l'objet dans son ensemble (principalement en

fonction du modèle géométrique de représentation) et non selon sa structure logique. L'algorithme permet de produire directement les objets. Selon ce principe, il existe des méthodes spécifiques à un type d'objet et des méthodes plus générales.

4.2.1 Les méthodes spécifiques

Dans certains cas, il existe un algorithme spécifique et performant permettant de produire, aléatoirement ou par énumération, des objets d'un type donné. Ils peuvent produire uniquement des objets répondant totalement ou partiellement aux descriptions qui les concernent. Le plus souvent, ces méthodes ne sont pas exploitables par les modeleurs déclaratifs, car elles sont difficilement contrôlables et, très souvent, ne génèrent qu'une seule solution.

4.2.2 Les méthodes générales

Il existe aussi des techniques de génération d'objets simples basées sur des algorithmes généraux (partiellement indépendants de l'objet effectivement construit). La classe d'algorithmes de ce type la plus évidente concerne les algorithmes génétiques. Nous pouvons aussi imaginer l'utilisation d'autres techniques comme les réseaux de neurones, les techniques d'optimisation telles que le recuit simulé… Notons que ces méthodes, lorsqu'elles fournissent plusieurs solutions, se trouvent très souvent dans la classe des algorithmes de génération aléatoire. En fait, elles ne sont pas réellement exploitables dans l'optique de la modélisation déclarative. En particulier, il n'est souvent pas possible de contrôler totalement la génération pour garantir l'exploration de toutes les solutions. Elles nécessitent donc des adaptations spécifiques (voir [Mou96] pour l'utilisation d'algorithmes génétiques).

Les techniques présentées en modélisation déclarative (arbres de construction…) sont, la plupart du temps, assez générales. Cependant, leur mise en place et les méthodes de construction qu'elles utilisent, sont spécifiques au domaine d'application considéré. Prenons l'exemple des algorithmes à base d'arbres d'énumération.

4.2.3 Les arbres d'énumération

Les arbres d'énumération sont une classe d'algorithmes assez courante en modélisation déclarative. Elle est présentée par [LMM89], [Elk89], [Pou94b], [Paj94], [Des95a], [Khe95] et [LuD95]. Elle est utilisée par [Mar90] et, plus particulièrement, par [Paj94] et [Pou94b]. L'objectif principal est de générer des arbres minimaux, c'est-à-dire des arbres où toutes les formes sont différentes. Cette technique se base sur un arbre d'énumération de numéros. L'arbre générera toutes les combinaisons de "*chiffres*" possibles selon une stratégie donnée (avec ou sans répétitions ; avec ou sans ordre des chiffres…). Une combinaison forme ce qu'on appelle un *numéro*. Ensuite, on fait une association entre un numéro et une forme par l'intermédiaire de tables ou d'une fonction de conversion (un décodeur) permettant de construire la scène. Tout ceci constitue un codage de la construction d'une forme par un numéro à

base n (n correspondant au nombre de couples possibles). Autrement dit, un générateur de numéros produit toutes les combinaisons de chiffres possibles (dans une base quelconque). Ces numéros sont fournis à un algorithme spécifique de décodage qui produit la scène ou l'objet (Figure 82).

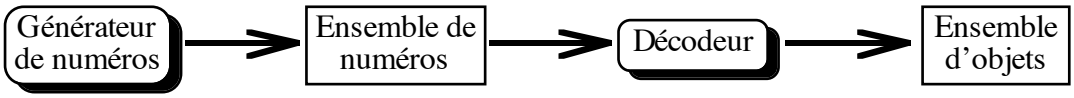

Figure 82. Principe de fonctionnement de la génération à base d'arbre d'énumération

Supposons que l'on possède un ensemble de *n* chiffres. Dans certains cas, la profondeur sera limitée artificiellement à P. Au nœud p, le numéro s'écrit $\{C_1, ..., C_p\}$. Nous avons alors les résultats présentés dans le Tableau 10.

Tableau 10 : Caractéristiques des différents arbres d'énumération

Caractéristique	CSR	CAR	ASR	AAR
Ordre	NON	NON	OUI	OUI
Répétitions	NON	OUI	NON	OUI
Équilibré	NON	NON	OUI	OUI
Fini nat.	OUI	NON	OUI	NON
Prof. Max.	n	P	n	P
Prof. Min.	1	P	n	P
Nb de noeuds à p chiffres	$C_n^p = \frac{n!}{p!(n-p)!}$	$\Gamma_n^p = C_{n+p-1}^p$	$A_n^p = \frac{n!}{(n-p)!}$	n^p
Prof. restant au niv. p	n - Cp	P - p	n - p	P - p
Nb. de fils au niv. p	n - Cp	n - Cp + 1	n - p	n
Nb. de frères au niv. p	n - Cp	n - Cp	?	n - Cp
Nb de nœuds au total	$\sum_{i=0}^{n} C_n^i = 2^n$	$\sum_{i=0}^{P} \Gamma_n^i$	$\sum_{i=0}^{n} A_n^i$	$\sum_{i=0}^{P} n^i = \frac{n^{P+1}-1}{n-1}$
Nb total de feuilles	2^{n-1}	Γ_n^P	n!	n^P

Tous ces arbres ont aussi certaines particularités communes :

- Lorsqu'on descend d'un niveau, un seul chiffre est ajouté. Ceux déjà placés sont invariants dans tout le sous-arbre.
- Le niveau d'un nœud dans l'arbre correspond au nombre de chiffres que contient ce nœud.
- A chaque nœud peut correspondre une forme (complète ou partielle).
- Etant donné un nœud, il est souvent facile de déterminer la profondeur maximale restant à atteindre et le nombre de frères restants (Tableau 10).
- Le parcours en préordre fournit les combinaisons dans l'ordre lexicographique (à condition d'ordonner les chiffres et de compléter les combinaisons par des zéros à droite).
- Etant donné un numéro, il est souvent facile de déterminer le numéro suivant.

Ces arbres sont intéressants dans un univers discret et fini (parfois artificiellement en limitant la profondeur de l'arbre), où l'on peut trouver un codage performant en fonction des objets de base, de leurs caractéristiques, des opérations élémentaires et aussi des propriétés demandées. Il est possible alors d'effectuer une *énumération totale* de l'univers des formes solutions. Par contre, pour certains codages mal choisis, deux problèmes peuvent arriver :

- des combinaisons de chiffres peuvent être interdites (le nombre ne permet pas de construire une forme) ;
- des nombres différents peuvent générer une même forme.

Il est donc important de bien étudier le codage afin d'éviter qu'il y ait des combinaisons interdites et de ne générer que des formes différentes. Il permet de garder une trace de la conception de la forme. Ceci est utile pour la visualisation d'une part et pour la génération du mode opératoire pour construire réellement la forme.

Cette méthode de génération est *semi-systématique* puisque seul le décodage est spécifique à l'objet. Par contre, il faut trouver d'une part un mode intéressant de génération des numéros (avec ou sans répétitions, ordonné ou non…) et mettre au point d'autre part un codage performant garantissant si possible l'unicité des solutions. Néanmoins, l'intérêt essentiel réside dans la possibilité de moduler ce décodage en fonction des propriétés demandées pour ne générer que des solutions ou, si ce n'est pas possible, ordonner l'exploration pour proposer les solutions au début de l'exploration. Notons que les techniques pour ordonner les solutions semblent difficiles à mettre en place dans le cas général. Cependant, nous le verrons dans la suite, cette méthode n'est pas exploitable d'un point de vue général pour une plate-forme. Nous préférerons un algorithme adapté aux structures de données que nous avons choisi.

5. Méthode de génération systématique

5.1. Introduction

Compte-tenu des caractéristiques des objets, outre les méthodes de génération spécifiques, il est possible de mettre en place une génération standard, systématique en mode aléatoire ou par énumération en fonction des besoins. Pour cela, nous proposons de nous baser sur la structure des objets. Tout est alors fonction de la méthode de parcours des domaines des concepts terminaux.

5.2. Génération et concepts composants

Les objets étant des concepts, nous nous intéresserons d'une manière générale à la génération des concepts terminaux, simples ou complexes. Notons que tous les concepts composants d'un concept simple ou complexe ne sont pas indispensables pour le construire.

Définition II.2.12 : Un concept composant participant à la construction d'un concept est appelé *concept générateur* ou *générateur*. Les autres sont appelés *concepts non-générateurs*.

Pour un concept complexe, l'ensemble des concepts générateurs n'est pas unique. C'est donc au concepteur de choisir l'ensemble le plus puissant (en rapidité mais surtout en contrôle des solutions). Nous pouvons assimiler les différents ensembles de concepts générateurs à différents modèles de représentation de l'objet. Les concepts non-générateurs sont déterminés à partir de l'ensemble de concepts générateurs. Ils sont redondants du point de vue de la représentation mais représentent une caractéristique pouvant être décrite par l'utilisateur. Dans la suite, pour simplifier, nous considérerons que l'objet ne comporte qu'un seul groupe de concepts générateurs sans concepts non-générateurs (ces derniers n'intervenant pas dans la génération à ce niveau de notre présentation).

Définition II.2.13 : Les procédures chargées de la construction d'un concept (de la détermination de sa mesure) sont appelées *tâches de génération* (ou plus simplement *tâches*).

5.3. Principe

La *génération récursive* repose sur la représentation des objets de la scène sous forme de hiérarchies de concepts. Cette technique de génération consiste à générer un concept non-terminal C_i (déterminer sa mesure) à partir de ses concepts composants générateurs : dans un premier temps, chaque concept composant générateur du concept C_i est généré, puis le concept C_i est construit en fonction des mesures des concepts composants (Algorithme 7). La condition d'arrêt est que le concept à générer est un concept terminal (pas de concept composant) ou pseudo-terminal (possède sa propre méthode de génération).

Définition II.2.14: Un *concept pseudo-terminal* est un concept non-terminal possèdant une méthode de génération spécifique. Il sera par conséquent considéré comme terminal par la génération.

Algorithme 7. Algorithme général de la génération récursive

```
Algorithme Générer(Concept c)
 Pour tout concept ci de c Faire
   Générer(ci)
 Fin pour
 Construire(c)
Fin algorithme Générer
```

Remarque : Dans ce cadre, la classe de génération d'un concept non-terminal C_i dépend des classes de génération de ses concepts composants. Intuitivement, un concept C_i est généré de façon aléatoire si au moins un de ses concepts composants est généré de façon aléatoire. Dans le cas contraire, la méthode de génération de C_i est de type énumération.

5.4. Génération des concepts terminaux

La génération d'un concept terminal ou pseudo-terminal peut être de la classe énumération ou aléatoire. Différentes méthodes de parcours sont possibles selon l'ordre d'énumération pour la première (croissant, décroissant, etc.) ou la probabilité de tirer telle ou telle valeur du domaine pour la seconde (lois de probabilité utilisées). La classe de génération appropriée dépend des caractéristiques du domaine. En effet, aussi bien pour énumérer que pour le tirage aléatoire, il est nécessaire d'avoir un domaine ayant des bornes connues (finies). Pour pouvoir énumérer, il est indispensable de savoir d'où on part et où on va. De même, le tirage aléatoire ne peut se faire que sur un intervalle donné.

Pour générer un objet par énumération, il faut que le domaine soit « énumérable » et fini. Cette seconde contrainte est assurée par les hypothèses que nous venons de prendre sur les bornes. La première contrainte n'est possible que si ce domaine est discret. Dans ce cas, il convient donc de vérifier que l'unité du domaine n'est pas nulle ou de discrétiser (échantillonner) ce domaine par exemple en fixant une certaine unité. Cette discrétisation est arbitraire ou calculée en fonction d'éléments divers comme un paramètre de l'application, un avis du concepteur ou de l'utilisateur… Cet échantillonnage n'est pas forcément régulier. Les modes d'énumération dépendent du concept ou des propriétés demandées. Nous pouvons imaginer autant de parcours possibles que nous voulons. Classiquement, les parcours sont : croissants, décroissants, aléatoires… Certains sont des parcours optimisés en fonction des propriétés demandées.

Pour générer un objet par tirage aléatoire, il faut déterminer la loi de probabilité du générateur de valeurs. Parmi ces lois, on trouve principalement la loi équiprobable où toute valeur du domaine à la même probabilité d'être sélectionnée ou la loi normale où les probabilités sont réparties selon une courbe de Gauss autour d'une valeur donnée. Cette dernière loi permet un tirage de valeurs « autour » d'une valeur du domaine considérée comme standard.

Définition II.2.15 : La tâche de génération associée à un concept terminal ou pseudo-terminal et chargée de déterminer sa mesure est appelée *tâche terminale*.

5.5. Génération des concepts simples

La génération d'un concept simple est fonction de la génération de ses concepts terminaux. La tâche de génération chargée d'un objet (concept simple) fait appel aux tâches de génération des différents champs (concepts terminaux) de l'objet (Figure 83). Cette tâche s'occupe de produire une mesure pour le concept dont elle a la charge à partir des mesures qu'elle obtient des sous-tâches des concepts terminaux. La génération d'un concept simple dépend étroitement de celles de ses champs.

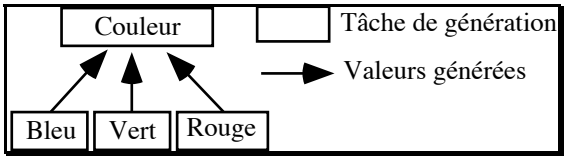

Figure 83. Organisation des tâches de génération d'un concept simple

Définition II.2.16 : La tâche de génération associée à un concept simple est appelée *tâche simple*.

5.5.1 Tirage aléatoire

La méthode pour produire de manière aléatoire un concept simple est triviale. Chacun de ses concepts générateurs produit une mesure de façon aléatoire. La mesure du concept simple est ensuite construite à partir de ces différentes mesures. On obtient alors l'Algorithme 8.

Algorithme 8. Génération aléatoire d'un concept simple

```
Algorithme Tâche_Simple.Suivant_Aleatoire(concept_simple i↓↑)
 Pour tout concept_générateur ci faire
   Tâche_Terminale.Suivant_Aleatoire(ci)
 Fin pour
 Construire i avec les mesures des ci
Fin algorithme Tâche_Simple.Suivant_Aleatoire
```

5.5.2 Enumération

Nous nous intéresserons ici à l'énumération des valeurs possibles d'un concept simple. La solution est basée sur un arbre d'énumération. Globalement, un algorithme général de construction peut être mis au point en numérotant les instances de cet objet ainsi que les valeurs des champs.

Soit $O=\{C_1, \ldots, C_n\}$ un objet comportant n concepts terminaux ordonnés. Si on veut l'objet de numéro i, alors chacun des champs C_j génère sa valeur numéro k = i[j] (i[j] est le chiffre de rang j du numéro i). Pour le champ C_j, le passage de la valeur k à la valeur k+1 lui est propre (croissant, décroissant, quelconque…). Une fois calculées toutes les valeurs de ses champs, l'objet peut être construit à son tour (Algorithme 9).

Algorithme 9. Génération par énumération d'un concept simple

```
Algorithme Tâche_Simple.Suivant_Par_Enumeration(concept_simple i↓↑)
 Si Numéro_Suivant(n) Alors
   Pour tout concept_générateur cj faire
     Tâche_Terminale.Suivant_Par_Enumeration(cj, n[j])
   Fin pour
   Construire i avec les ci
 Sinon Afficher(« Il n'y a plus d'autres objets »)
 Fin Si
Fin algorithme Tâche_Simple.Par_Enumeration
```

Remarques :

- L'ordre des champs peut être fonction de l'objet, des propriétés demandées…
- Cette méthode de génération s'appelle aussi une génération par *arbre de paramètres* ([Fau95]).
- Cette méthode demande aux concepts terminaux le moyen d'atteindre une valeur quelconque rapidement.

5.6. Génération des concepts complexes

Nous nous intéressons ici aux tâches permettant de générer des objets complexes, c'est-à-dire des concepts composés de concepts terminaux, simples ou complexes (Figure 84).

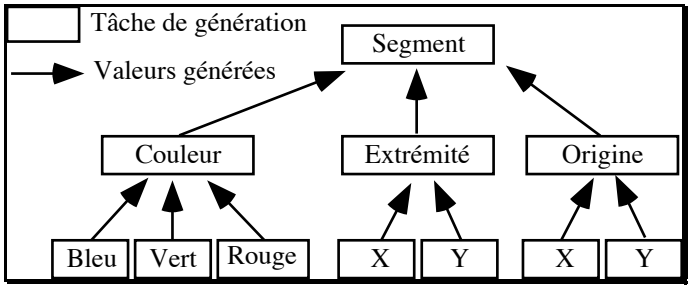

Figure 84. Organisation des tâches de génération d'un concept complexe

Définition II.2.17 : La tâche de génération associée à un concept complexe est appelée *tâche complexe*.

5.6.1 Enumération

Une première idée consiste à essayer d'utiliser (et éventuellement de généraliser) la méthode proposée pour les concepts simples. Pour chaque concept générateur, on calcule le numéro de la solution que l'on attend. Cependant, la technique par arbre d'énumération, si elle est facilement applicable pour le parcours de concepts simples, est plus compliquée à mettre en place pour des objets comme les listes d'objets simples (concepts simples) et surtout pour les concepts complexes. En effet, pour chaque chiffre du numéro, il faut déterminer le nombre de valeurs possibles (base). Ceci se fait assez bien pour des concepts terminaux dont les domaines sont facilement énumérables et dont le nombre d'éléments se calcule aisément. Par contre, pour un concept non-terminal, il n'est pas toujours facile de déterminer le nombre d'objets correctement construits et valides possibles. Nous allons donc mettre en place une autre méthode basée sur l'énumération récursive des concepts. Pour chacun des concepts de la hiérarchie, les seules fonctions connues sont le premier élément et le passage à l'objet suivant.

Les concepts générateurs sont organisés en « pile ». L'objet suivant correspond au suivant de l'objet situé au dessus de la pile. Si celui-ci n'a pas de suivant, on dépile et on recommence. Une fois que l'on a trouvé un suivant, on empile à nouveau les concepts en prenant le

premier (première valeur). Si la pile est vidée, la génération est terminée. Nous avons donc l'Algorithme 10.

Algorithme 10. Génération par énumération d'un concept complexe

```
Algorithme Tâche_Complexe.Suivant_Par_Enumeration(concept_simple i↓↑) :
booleen
 Pile Pile_Temp
 Pile Pile_Concepts_Générateurs(∀c_j)
 booleen Existe_Suivant = faux
 Tant que ¬Existe_Suivant ET Pile_Concepts_Générateurs non vide faire
   Existe_Suivant =
                Suivant_Par_Enumeration(Tete(Pile_Concepts_Générateurs))
   Si ¬Existe_Suivant alors
     Empiler(Pile_Temp, Depiler(Pile_Concepts_Générateurs))
   Fin Si
 Fin Tant que
 Si Existe_Suivant alors
   Tant que Pile_Temp non vide faire
     Concept c = Depiler(Pile_Temp)
     Premier_Par_Enumeration(c)
     Empiler(Pile_Concepts_Générateurs, c)
   Fin Tant que
   Construire i avec les c_i
 Sinon
   Afficher(« Il n'y a plus d'autres objets »)
 Fin Si
 Retourner Existe_Suivant
Fin algorithme Tâche_Complexe.Par_Enumeration
```

Ainsi, il n'est pas nécessaire de calculer a priori le nombre de solutions pour un concept. Il suffit de connaître le premier élément et une fonction permettant de passer au suivant. Notons que cette fonction peut être complexe et dépendre de critères spécifiques selon le domaine d'application, la description fournie par l'utilisateur et l'état de la base de connaissances.

Cette méthode est aussi applicable pour les concepts simples. La mise en place des méthodes donnant le premier et le suivant pour les concepts terminaux est simple. Nous allons donc adopter cet algorithme pour tout concept non-terminal.

5.6.2 Tirage aléatoire

En ce qui concerne la méthode de génération par tirage aléatoire, la génération de concepts complexes est identique à celle des concepts simples.

5.7. Remarque

Les concepts simples et complexes étant traités de la même façon (les seconds étant une généralisation des premiers), nous ne considérerons dans la suite qu'un seul concept : le *concept non-terminal*.

5.8. Personnaliser la génération récursive

La méthode de génération systématique est très utile lorsque le concepteur ne veut pas s'occuper de la génération ou ne connaît pas de méthode spécifique particulière. Néanmoins, il arrive que, pour un concept non-terminal donné, une méthode globale soit plus efficace que la génération récursive. De plus, le concepteur peut vouloir panacher la génération d'un objet en produisant certains concepts générateurs par tirage aléatoire et d'autres par énumération. Nous essayerons donc d'assouplir la génération systématique en fonction des besoins par une personnalisation des techniques et un mixage des modes de génération.

5.8.1 Création de concepts pseudo-terminaux

La méthode récursive n'est ni la plus efficace ni la plus naturelle. Le concepteur peut disposer d'un algorithme spécifique plus intéressant, en particulier du point de vue performance. Pour cela, il lui suffit de créer une nouvelle tâche de génération spécialisée pour le concept non-terminal qui devient alors pseudo-terminal. Cette tâche est alors une tâche terminale par rapport à l'algorithme systématique (à partir de ce niveau, la génération n'est plus du ressort de la méthode systématique). Introduire une méthode de génération spécifique consiste donc à créer une nouvelle tâche terminale. Comme nous l'avons vu dans les paragraphes précédents, une tâche s'occupe de la génération d'un concept. Elle doit donc évidemment posséder des méthodes de génération : « init » et « suivant » (Tableau 11).

Tableau 11. Définition d'une tâche

Nom : Tâche **Mère** :	
Principaux champs	**Principales méthodes**
Concept associé	Init()
Mode (Aléatoire ou Énumération)	Suivant()

Les tâches associées aux différents types de concept héritent de cette tâche « générale ». Une *tâche non-terminale* est une tâche associée à un concept non-terminal dans l'optique de la génération systématique. Il en existe deux types suivant que la génération se fait par tirage aléatoire ou par énumération. Elles ont une méthode « suivant » spécifique comme nous l'avons vu au paragraphe 5. La Figure 85 propose toute la hiérarchie standard des tâches.

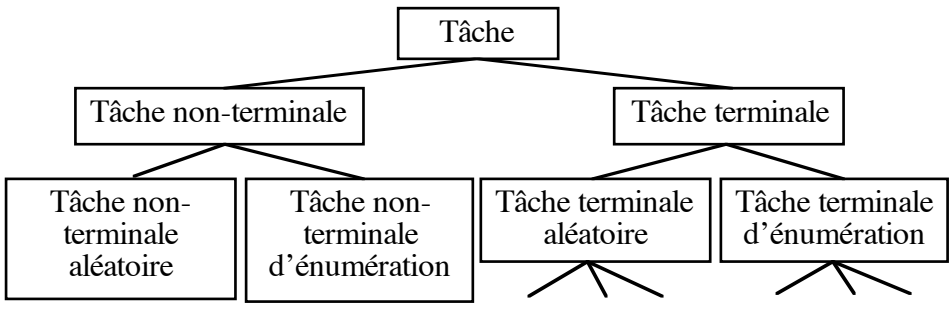

Figure 85. Hiérarchie des tâches

Il existe plusieurs types de tâches terminales. Ils proposent chacun une méthode de parcours du domaine spécifique. Par exemple, on peut trouver :

- la tâche aléatoire (tirage selon un générateur pseudo-aléatoire) uniformément distribuée ou selon une distribution de Gauss ;
- la tâche par énumération croissante ou décroissante ;
- la tâche par énumération aléatoire (on énumère toutes les valeurs sans répétitions mais dans un ordre « quelconque »)…

Pour mettre en place une méthode de génération spécifique, il suffit de créer une nouvelle tâche terminale en surchargeant les tâches « Init » et « Suivant » (Figure 86). Il est aussi important de spécifier si cette nouvelle tâche est effectuée par tirage aléatoire ou par énumération. En conséquence, pour chaque concept (objet de la scène) défini par le concepteur, il faut associer un champ indiquant la tâche de génération voulue. Les concepts générateurs des concepts-objets ont, par défaut, le même type de génération.

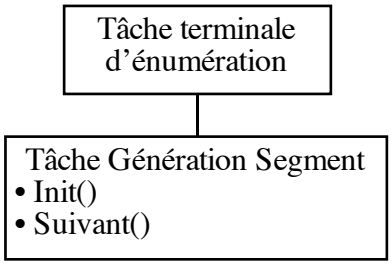

Figure 86. Construction d'une tâche de génération spécifique

5.8.2 Combiner énumération et tirage aléatoire

Dans une application, le concepteur doit pouvoir « mélanger » les deux modes de génération pour un même objet. Ce concept comporte alors certains concepts générateurs produits par énumération et certains autres par tirage aléatoire. Par exemple, imaginons une application permettant de construire un ensemble de bâtiments. L'important est la position et les dimensions des bâtiments. Par contre leur couleur importe peu. Ainsi, l'application énumère les bâtiments selon leurs caractéristiques géométriques alors que leur couleur est générée aléatoirement. Les concepts de ce type sont considérés comme produits à l'aide d'une méthode par énumération. En effet, une telle méthode est plus coûteuse et plus complexe à gérer qu'une méthode par tirage aléatoire. C'est pourquoi elle est prépondérante. Par conséquent, les concepts, dont la méthode n'est pas explicitement déterminée par le concepteur, sont générés par tirage aléatoire (génération par défaut).

Dans le cas que nous venons de décrire, il n'est plus possible de séparer une tâche non-terminale aléatoire d'une tâche non-terminale d'énumération. Nous n'avons plus qu'une tâche non-terminale pouvant générer aussi bien selon l'une ou l'autre des méthodes (Figure 87).

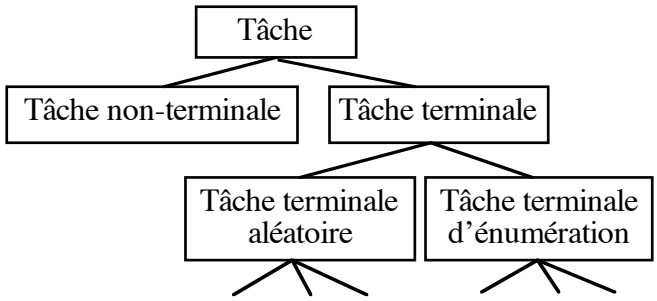

Figure 87. Hiérarchie des tâches

6. Générer une scène

Habituellement, une scène est composée de plusieurs objets. Nous nous intéresserons ici à la génération d'une scène. Dans une scène à générer, on trouve deux sortes d'objets (ou concepts) :

- les objets « virtuels », que l'on appelle aussi *ébauches* d'autres objets ;
- les objets « réels » que l'on retrouvera dans la scène finale.

Tous les objets de la scène ne peuvent pas être générés en même temps. Il existe certaines dépendances entre eux, en particulier par rapport à la méthode de conception choisie par le concepteur ([CDMM97b] et [CDMM97c]). Ils sont donc organisés de manière hiérarchique en fonction de cette méthode. Cette organisation sera prise en compte par la génération. La hiérarchie de la scène est à différencier de celle d'un concept complexe. Dans la première, chaque niveau de l'arbre peut être un objet très différent de l'objet de niveau inférieur et qui ne sera pas forcément présent dans la scène finale alors que dans la seconde les objets de niveau inférieur sont des composants du niveau courant.

Par exemple, s'il désire une conception par ébauche successives (Figure 88), il peut convenir que chaque niveau de la hiérarchie de la scène est un niveau de détail, une ébauche. La génération doit alors s'effectuer par étapes selon un ordre donné. Il est donc indispensable d'avoir un système, que nous appellerons chef d'orchestre, permettant de gérer la génération de la scène en fonction des contraintes de l'application.

Définition II.2.18 : Le *chef d'orchestre* est chargé d'organiser la génération de la scène vis-à-vis de la méthode de conception choisie et des actions de l'utilisateur. Il ordonne la construction des objets et gère les phases de conception.

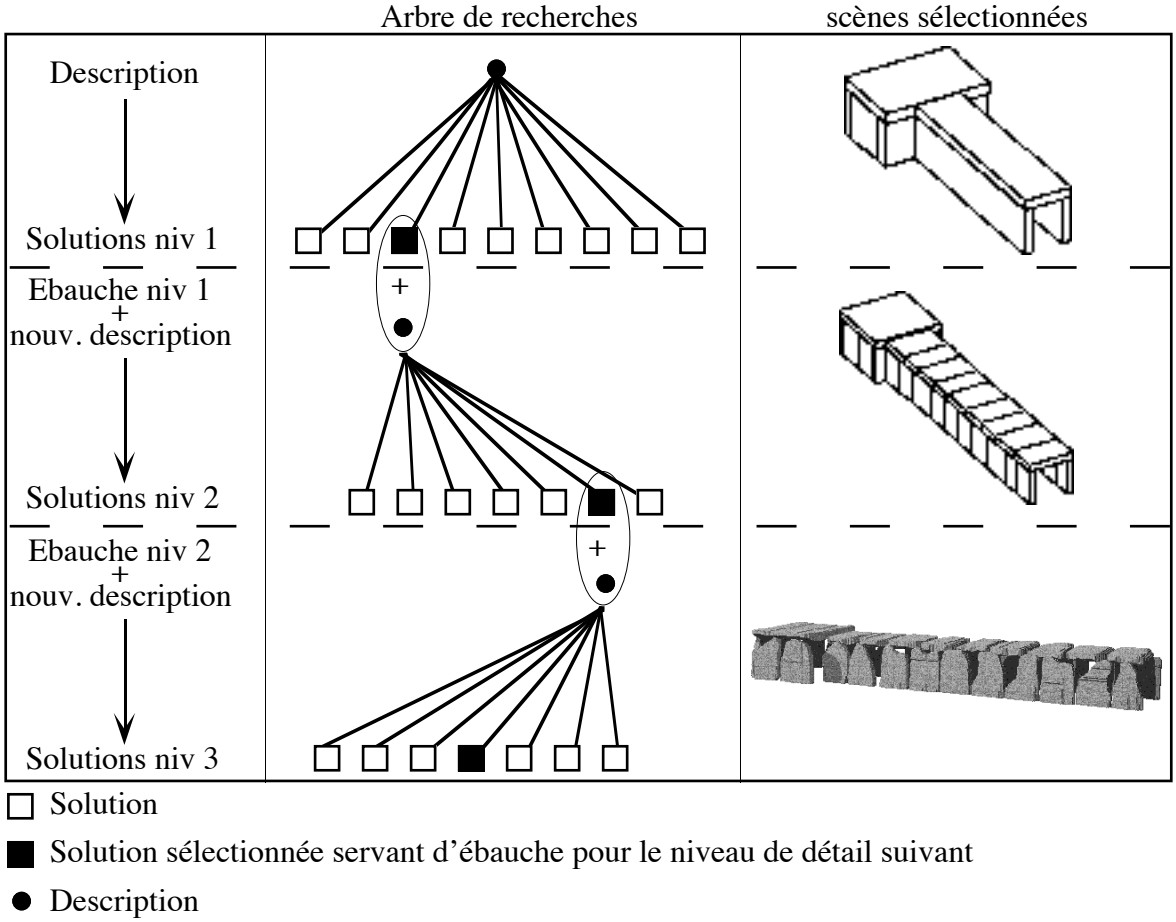

Figure 88. Mode de travail "conception par ébauches successives" [CDMM97c]

6.1. Classification en groupes

Le chef d'orchestre se charge d'abord de classer, de grouper les objets de la scène pour toute la hiérarchie. Chaque groupe possède un numéro d'ordre. Le groupe 0 ne comporte que la racine, c'est-à-dire l'objet « scène ». La classification effectuée dépend du mode de génération désiré. Il existe quelques méthodes de classification générales mais le concepteur peut proposer sa propre méthode (Figure 89).

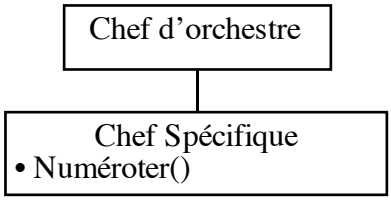

Figure 89. Redéfinition de la méthode de classification et numérotation

Parmi les classifications « standard », nous proposons les suivantes :

- un seul niveau (mode automatique), tout est généré en même temps (Figure 90) ;
- par niveaux de l'arbre (génération par niveau de détail), où tous les objets de même niveau appartiennent au même groupe dont le numéro est égal à ce niveau (Figure 91) ;

- par frères, où tous les fils d'un objet sont dans le même groupe (Figure 92) ;
- en profondeur, où chaque objet est dans un groupe différent, numéroté par parcours de la hiérarchie en profondeur d'abord (bien évidemment préfixé puisque tout objet de niveau i est ébauche pour ses fils de niveau i+1)…

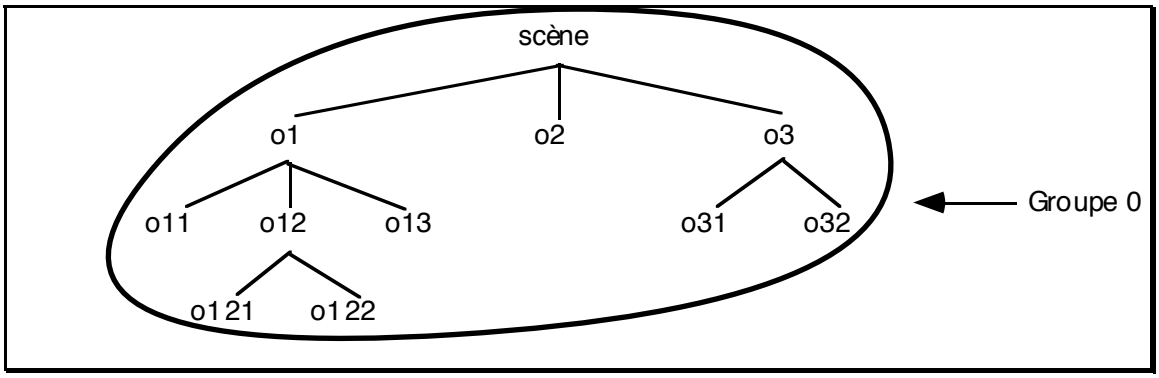

Figure 90. Classification pour une génération en mode automatique

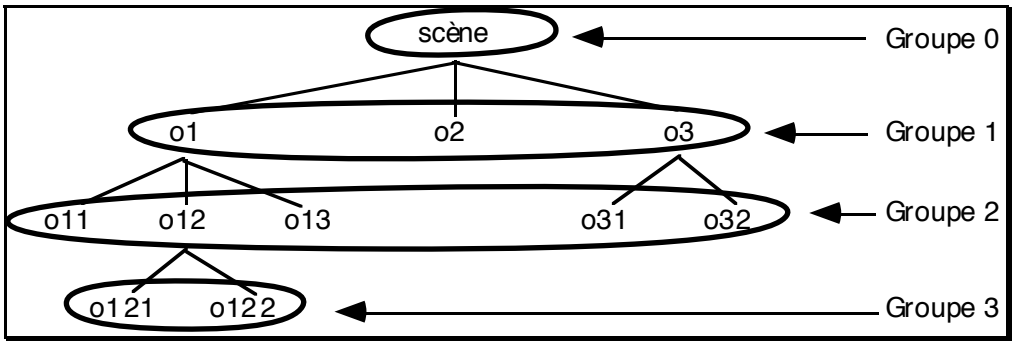

Figure 91. Classification par niveaux de la hiérarchie

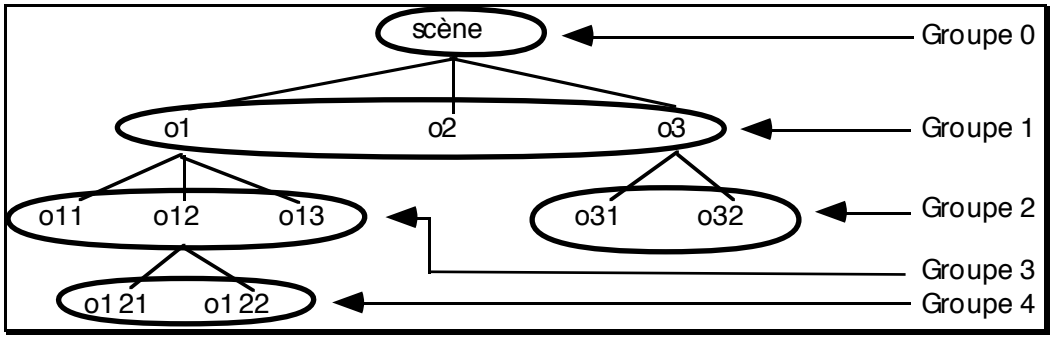

Figure 92. Classification par frères

6.2. Génération d'un groupe

Le chef d'orchestre s'occupe aussi de la génération des groupes. Pour un groupe donné, cette génération est en mode « automatique », c'est-à-dire l'utilisateur n'a pas à intervenir dans la génération (voir [CDMM97c]) sauf, nous le verrons, pour résoudre des situations de blocage. Le chef d'orchestre doit générer une solution pour le groupe afin qu'elle soit présentée à l'utilisateur. Il doit donc gérer des situations où certains objets sont produits de manière

aléatoire alors que d'autres le sont par énumération. Il donnera la priorité aux objets énumérés afin d'obtenir le plus de solutions possibles. Cependant, le concepteur peut faire en sorte que l'utilisateur dirige la génération en demandant de s'occuper en priorité de tel ou tel objet. Ainsi, nous avons la possibilité d'avoir un mode de conception guidée (voir [CDMM97c] ; Figure 93). Pour bien gérer la valeur d'un concept pendant la génération et au cours des phases de conception, nous allons introduire la notion de *validité*. Un objet de la scène peut être dans trois états : « à générer », « à valider » ou « validé ».

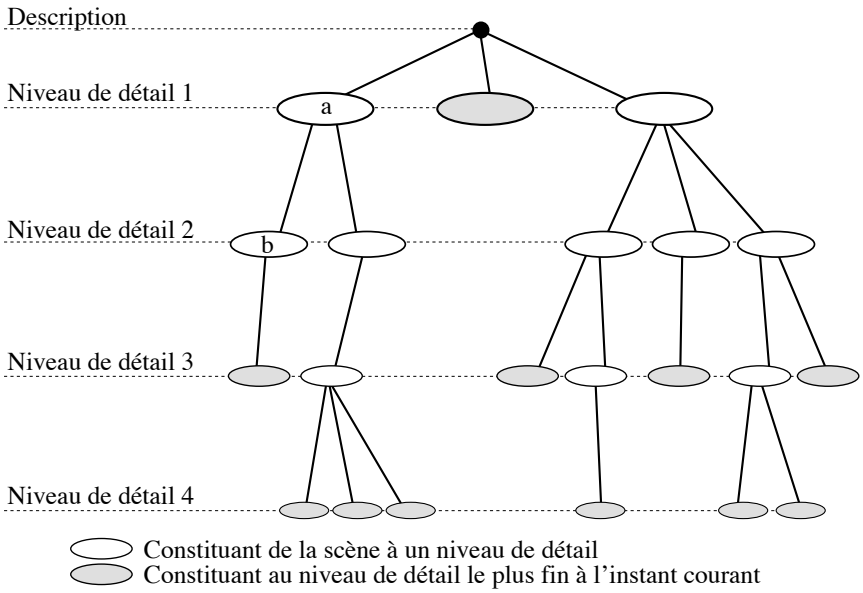

Figure 93. Scène à un instant t [CDMM97c]

L'objet est « à générer » en début de génération ou lorsque l'utilisateur le refuse. Il est « à valider » lorsqu'il vient d'être produit et qu'il est pret à être proposé. L'objet est « validé » quand l'utilisateur le considère comme satisfaisant. Il ne doit donc plus être généré. C'est alors un objet final ou une ébauche pour un prochain groupe. La Figure 94 montre l'évolution de l'état de validité d'un objet en fonction des événements lors de la génération d'un groupe.

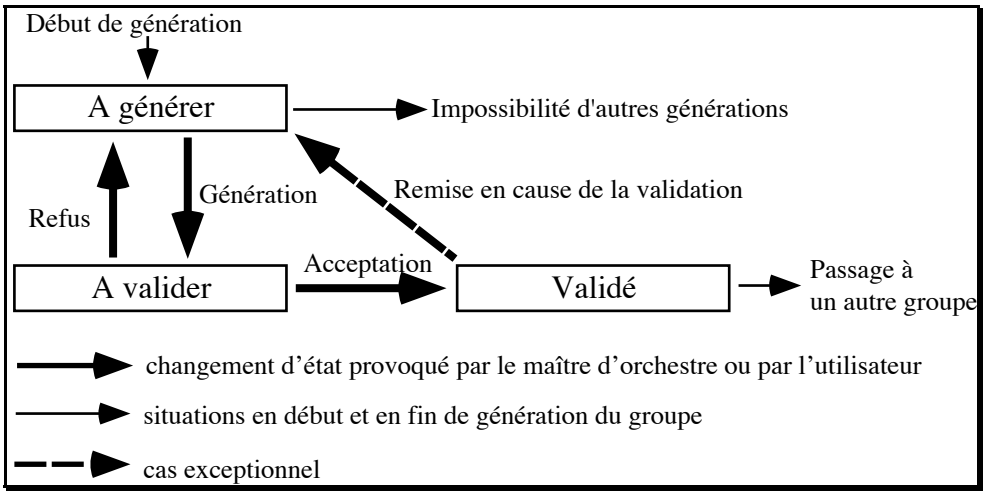

Figure 94. Changement d'état d'un objet au cours des générations de son groupe

Ainsi, aux informations déjà présentes dans la table des objets, il faut ajouter la validité courante de l'objet ainsi que le numéro de groupe auquel il est associé. De plus, lors de la génération, il est nécessaire de stocker les objets construits (selon leur modèle géométrique). Nous aurons donc une table spécifique à la génération que nous appellerons *table de génération*. Pour chaque objet de la table des objets, cette table indique la validité de l'objet et contient l'instance (le modèle géométrique des objets déjà conçus) le numéro du groupe auquel il appartient et la tâche de génération qui le concerne (Tableau 12).

Tableau 12. Constitution de la table de génération

Champ	Validité	Numéro	Instance	Groupe	Tâche
Type	Entier	Entier	Ptr	Entier	Ptr

Remarque : Avec ce que nous avons désormais présenté, il est possible de construire un modeleur déclaratif dont la génération se fait par ébauche (Figure 91 et Figure 88) et pour lequel, le passage d'une ébauche à la suivante se fait par conception guidée. Nous avons ainsi un mixage de deux modes de conception présentés dans [CDMM97b] et [CDMM97c].

7. Contraintes de construction des objets

7.1. Introduction

Les méthodes de génération proposées dans les paragraphes 5 et 6 supposent une indépendance totale entre les différents concepts générateurs d'un concept non-terminal. Cependant, prenons simplement la génération d'un cube ou d'un segment, nous constatons alors que cette indépendance ne peut pas être raisonnablement maintenue. Dans le second cas, nous allons générer plusieurs fois le même segment et dans le premier, nous risquons de produire des cubes sortant de l'univers. Il apparaît donc indispensable d'ajouter d'une part des contraintes de construction permettant de limiter la génération d'objets incorrects et d'autre part une méthode de vérification de l'objet après la construction. L'utilisation d'une telle méthode de vérification complique un peu la construction d'un concept. Quelle que soit la méthode choisie, il faut recommencer autant de fois que nécessaire la génération de l'objet jusqu'à la réussite de la vérification ou un constat d'échec définitif. Nous allons nous intéresser à l'introduction des contraintes de construction entre les concepts pour éviter au maximum l'échec de cette vérification. Ces contraintes mettent en relation :

- des concepts composants d'un concept (Figure 95, les contraintes 1, 4 et 7) ;
- des concepts de différents objets de la scène, que ce soit des objets parents (Figure 95, les contraintes 3 et 5) ou des objets sans lien de parenté (Figure 95, la contrainte 2 et 6).

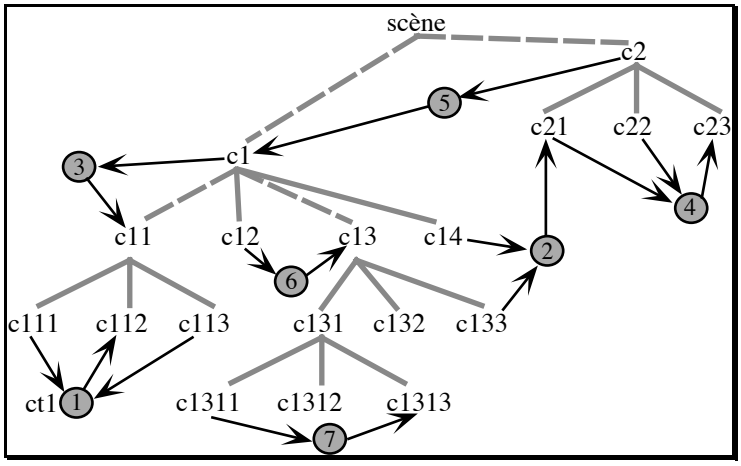

Figure 95. Différents cas de contraintes de construction entre les concepts de la scène

7.2. Une contrainte

7.2.1 Définition

Définition II.2.19 : une *contrainte* est une fonction portant sur un *concept cible* et dont les paramètres sont des concepts appelés *concepts de référence* (Figure 96).

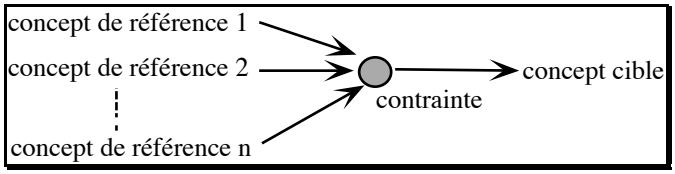

Figure 96. Structure d'une contrainte

Toute contrainte possède une *fonction d'évaluation* prenant en paramètres les mesures des concepts de référence et agissant sur la mesure ou le domaine du concept cible.

7.2.2 Types de contrainte

En fonction du moment où la contrainte est appliquée et de l'élément sur lequel elle porte, nous distinguons trois types de contrainte : initialisation, optimisation et mesure.

Définition II.2.20 : Les *contraintes d'initialisation* permettent de déterminer le domaine du concept cible avant le début de sa génération.

En effet, ce domaine n'est pas forcément connu par le concepteur. Ces contraintes le déterminent alors dynamiquement en fonction des conditions fournies par l'utilisateur. Prenons par exemple un ensemble de segments du plan. Supposons que ces segments soient définis dans des zones rectangulaires. Le domaine associé à chacun des segments n'est connu que lorsque la zone est fixée, c'est-à-dire ses dimensions et sa position validées par l'utilisateur.

Lorsque l'utilisateur ajoute des objets, le système consulte les graphes sémantiques afin d'autoriser cette opération. Ces graphes permettent aussi de décrire les contraintes d'initialisation entre des objets. Ainsi, une liste de ces contraintes est associée à chacun d'eux. Par exemple, le graphe de la Figure 81a possédera les contraintes d'initialisation de la Figure 98 entre « Largeur » de scène et « x », entre « Hauteur » de scène et « y »…

Au cours de la construction d'un concept non-terminal, certaines combinaisons de concepts composants ne sont pas autorisées. De plus, connaissant les valeurs de certains concepts, il est possible de réduire le domaine d'un autre concept afin de limiter le nombre de valeurs à explorer. Pour tout cela, il est intéressant de mettre en place des contraintes d'optimisation.

Définition II.2.21 : Les *contraintes d'optimisation* ne modifient pas l'univers des solutions à explorer mais le réduisent provisoirement afin d'éviter de construire des scènes qui seront certainement rejetées.

Prenons comme exemple la création de rectangles (ou zones) dans un univers à deux dimensions. Ces rectangles sont placés dans un univers de dimensions finies. S'ils sont placés selon la position de leur centre et leurs dimensions en largeur et en hauteur, toutes les combinaisons ne sont pas valides. Par exemple, dans la Figure 97, les rectangles A et B sont bien construits alors que les rectangles C et D ne le sont pas. Pour C, la hauteur n'a pas une valeur correcte. Il aurait fallu une valeur inférieure ou égale à celle représentée par les tirets. Il en est de même pour la largeur D. Dans cet exemple, il sera donc intéressant d'ajouter deux contraintes d'optimisation réduisant l'une le domaine des hauteurs et l'autre le domaine des largeurs en fonction de la position du centre.

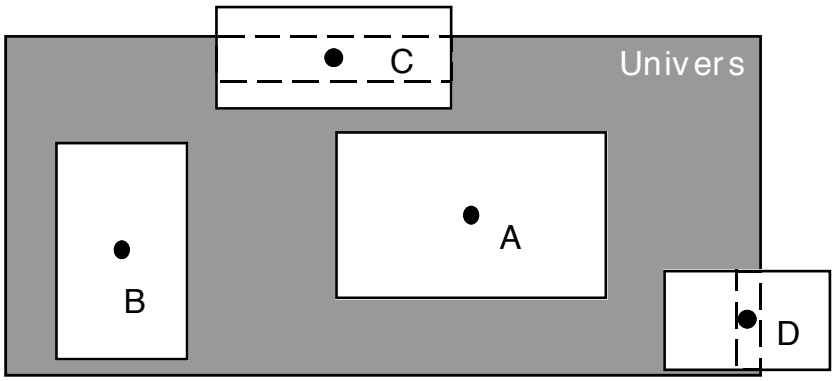

Figure 97. Placement de rectangles et contraintes d'optimisation

Nous avons vu dans la seconde partie que, pour tout concept, on associe une *fonction de mesure* permettant de déterminer, pour une scène donnée, la valeur du domaine qui lui correspond appelée *mesure*. Jusqu'ici, nous avons considéré les concepts composants uniquement comme des concepts générateurs. Ceux-ci n'ont pas besoin de fonction de mesure, car c'est à partir de leur valeur que la scène est construite. Or, ce n'est pas toujours le cas. Il existe cer-

tains concepts appelés concepts non-générateurs qui, bien que ne participant pas à la génération, peuvent faire l'objet d'une description. Il est alors nécessaire de déterminer une fonction de mesure. Par définition, c'est une fonction de l'univers des formes possibles dans le domaine considéré (définition II.1.4). Nous la considérerons comme une contrainte prenant en référence un ou plusieurs concepts de la scène et portant sur la mesure du concept cible.

Définition II.2.22 : La *contrainte de mesure* est une contrainte jouant le rôle de fonction de mesure pour un concept donné.

La Figure 98 propose la définition d'un rectangle. Nous y retrouvons les trois types de contraintes que nous venons de présenter. Les contraintes 1, 2, 3 et 4 sont des contraintes d'initialisation. Les contraintes 5 et 6 sont des contraintes d'optimisation. Enfin, la contrainte 7 est une contrainte de mesure pour le concept non-générateur « surface ».

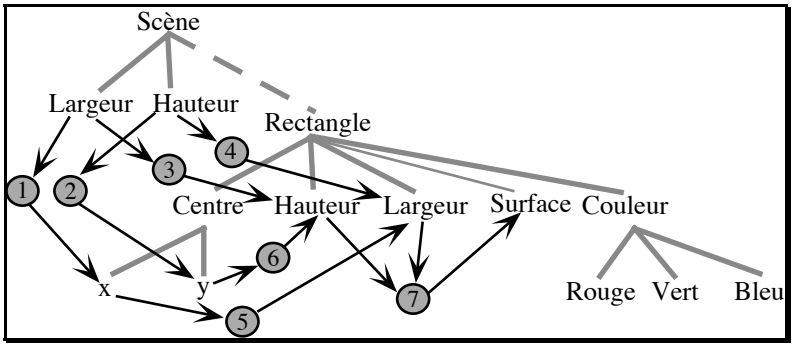

Figure 98. Types de contrainte dans la définition d'un rectangle

7.2.3 Optimisation par le calcul des intervalles

Parfois, avant même de connaître les mesures des concepts de référence, il est possible de réduire le domaine du concept cible à l'aide des domaines de ces concepts. Ce calcul s'effectue à l'aide de l'arithmétique des intervalles ([Sny92], [SnK92], [Chau94b] et [MaM96]). En effet, les domaines sont des intervalles possédant une unité. Par conséquent, il est possible d'appliquer cette arithmétique. Cela permet de réduire l'intervalle (le domaine) du concept cible et d'éliminer rapidement certaines combinaisons des concepts de référence (avant même de connaître toutes les mesures).

7.3. Structure d'une contrainte

Finalement, une contrainte est composée (Tableau 13) :

- d'une liste des concepts de référence ;
- du concept cible ;
- de la liste des « valeurs » courantes des concepts de référence (mesure ou domaine).

Tableau 13. Définition d'une contrainte

Nom : Contrainte **Mère** :	
Principaux champs	**Principales méthodes**
booléen Initialisante?	Init()
Liste concepts de référence	Activer()
concept cible	Désactiver()
	Calculer()
	CalculerDomaine()

De plus, une contrainte peut être activée ou désactivée en fonction des besoins. Par exemple, une contrainte de mesure n'est activée que lorsque le concept considéré fait l'objet d'une description par l'utilisateur. Si ce n'est pas le cas, il n'est pas nécessaire d'effectuer ces calculs alors inutiles. Une contrainte possède aussi deux méthodes de calculs : l'une concernant les optimisations selon l'arithmétique des intervalles (« CalculerDomaine() ») et l'autre l'évaluation de la contrainte (« Calculer() »).

Une contrainte permet de réduire le domaine du concept cible. Par défaut, une contrainte propose un intervalle de validité. L'application de la contrainte sur le concept cible consiste en une intersection entre cet intervalle et le domaine. Pour faciliter la programmation du modeleur, le noyau propose des cas particuliers de contraintes où seule une borne est à calculer. Nous obtenons alors les contraintes de supériorité ou d'infériorité (Figure 99).

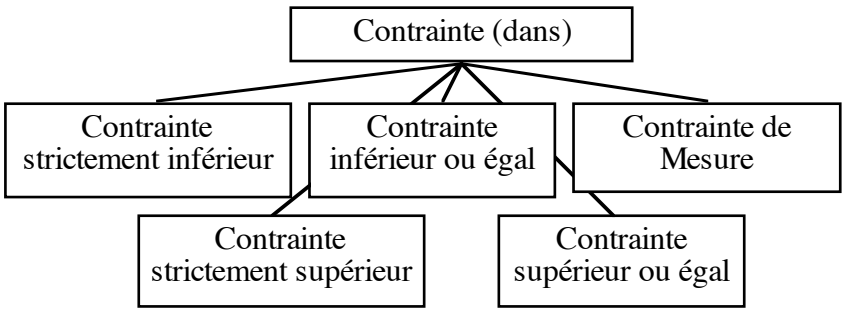

Figure 99. Hiérarchie des contraintes

7.4. Créer une contrainte

Ajouter une contrainte consiste simplement à déterminer les concepts de référence, le concept cible et à spécifier la fonction d'évaluation (Figure 100). De plus, dans le cas où cette contrainte est une fonction de mesure, il faut indiquer au concept concerné son existence.

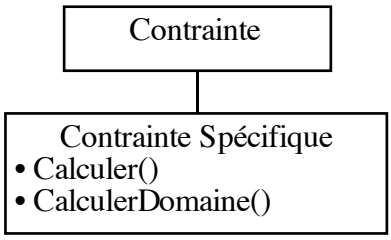

Figure 100. Redéfinition de la méthode de calcul d'une contrainte

7.5. Gestion des contraintes

Nous allons maintenant nous intéresser à l'utilisation de ces contraintes lors de la génération d'instances des concepts.

7.5.1 Les domaines d'un concept

Tout au long de la génération d'une scène, le domaine d'un concept passe par certaines phases :

1. Intervalle donné par le concepteur ;
2. Intervalle calculé à partir des contraintes d'initialisation ;
3. Intervalle « réduit » en fonction des contraintes d'optimisation.

Définition II.2.23 : Afin de pouvoir gérer convenablement les retours en arrière et améliorer les performances, au lieu d'un seul domaine, nous considérerons pour un concept trois domaines :

1. le *domaine d'origine* donné par le concepteur ;
2. le *domaine de travail* calculé à partir des contraintes d'initialisation et du domaine d'origine ;
3. le *domaine courant* issu du domaine de travail et sur lequel on applique les contraintes d'optimisation (Figure 101).

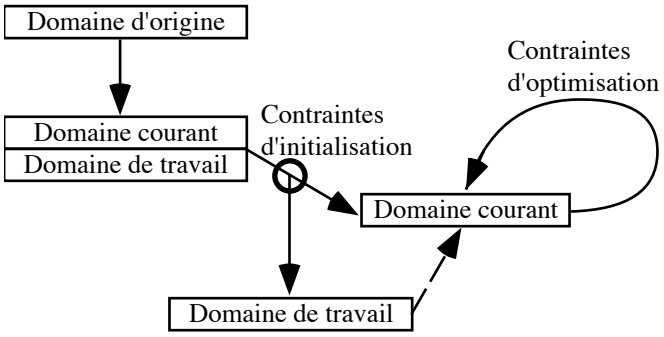

Figure 101. Contraintes et domaines d'un concept

8. Concepts, contraintes et génération

8.1. Méthode de génération et CSP

Les contraintes d'initialisation sont appliquées à l'initialisation du concept, c'est-à-dire au moment où l'on commence sa prise en compte dans la génération. Cela permet de déterminer le domaine de travail. Le comportement d'un concept vis-à-vis des contraintes est le même qu'il soit ou non concept générateur.

En cours de génération, à chaque fois qu'une mesure est calculée (par la tâche de génération ou par une contrainte de mesure), la modification est répercutée sur les concepts liés par des contraintes avec le concept modifié. Ainsi, il y a une propagation des modifications. A tout moment, les domaines des différents concepts sont réduits au mieux. Cette méthode permet de réduire l'espace de recherche des tâches de génération et de détecter rapidement les constructions ne donnant pas de résultat. Ainsi, il y a un nouveau cas d'échec pour le choix d'une valeur d'un domaine par une tâche de génération. En effet, chaque valeur doit être dans le domaine courant et vérifier les propriétés qui portent sur le concept mais aussi ne pas provoquer des domaines sans valeur pour d'autres concepts en relation par des contraintes, c'est-à-dire ne pas provoquer d'incohérence. Si c'est le cas, il faut alors passer à une autre valeur pour ce concept.

Nous pouvons approcher cette méthode de la méthode CSP nommée « lookahead - forward checking » ([Tsa93]). Ainsi, nous pouvons aussi appliquer des algorithmes d'optimisation issus de ces techniques. En particulier, il est possible d'établir un ordre de génération des concepts générateurs en fonction de leurs liens par les contraintes. Nous pouvons notamment utiliser la méthode de largeur minimale (« Minimal Width Ordering Heuristic » ou MWO) qui semble particulièrement intéressante pour nos problèmes ([Lie96]). Cependant, ces techniques demandent quelques adaptations, car d'une part nous ne voulons pas une mais plusieurs voire toutes les solutions et d'autre part les contraintes ne sont pas forcément binaires et symétriques ou antisymétriques (par exemple la contrainte qui mesure la surface d'un rectangle dans l'exemple de la Figure 98). Les méthodes CSP ne sont pas toujours adaptées à ces conditions ([Tsa93]).

8.2. Cycle de génération

D'un point de vue global, la méthode de génération suit un cycle en trois phases :

1. la réduction au mieux des domaines (application des contraintes d'initialisation et d'optimisation) ;
2. la recherche d'une valeur pour un concept donné ;
3. la création de la mesure de ce concept et l'application des contraintes de mesure.

Autrement dit, notre génération suit le cycle : réduction de l'univers, exploration et création de connaissances (Figure 102). Le cycle est terminé lorsque tous les concepts ont une valeur ou lorsqu'il n'est plus possible de construire une forme.

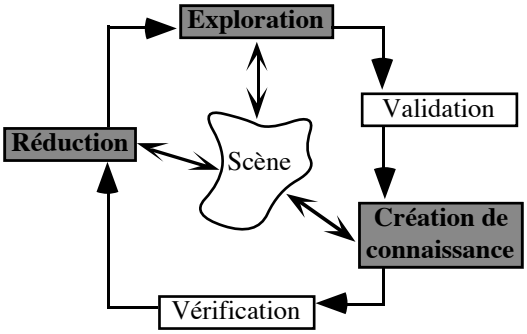

Figure 102. Cycle de fonctionnement

8.3. Suspension de la génération

Les objets de la scène sont générés par groupes. La constitution de ces derniers dépend de la méthode de conception choisie. Dans un groupe, les objets sont générés en mode automatique, c'est-à-dire sans intervention de l'utilisateur dans la génération. Cependant, l'application d'une contrainte d'initialisation sur un objet n'est pas toujours possible. En effet, elle peut dépendre d'objets de groupes différents qui ne sont pas encore générés ou même d'objets de ce groupe dont l'initialisation n'est pas encore effectuée. Sa génération est alors reportée jusqu'au moment où les contraintes d'initialisation seront applicables. Par conséquent, il peut arriver qu'un groupe ne puisse pas être produit. Sa construction est alors suspendue et la génération passe à un autre groupe. La génération du groupe suspendu sera reprise plus tard, lorsque les contraintes seront applicables.

La suspension de la génération a aussi lieu lorsque le concept et la tâche de génération qui lui est associée ne sont pas compatibles. Par exemple, pour énumérer les valeurs d'un domaine, il faut que celui-ci soit borné et possède une unité non nulle. Dans le cas contraire, la génération de l'objet est suspendue dans l'attente de l'application d'une contrainte d'optimisation qui pourrait résoudre le problème.

8.4. Intervention de l'utilisateur

Il peut arriver que, par manque d'informations ou par défaut de construction, un ou plusieurs groupes soient suspendus sans pouvoir générer un seul objet supplémentaire. A ce moment, l'utilisateur doit intervenir pour débloquer la situation. Si le mode de génération autorise l'utilisation d'interfaces, le problème peut être résolu (nous verrons cela dans le chapitre II.4) sinon c'est un défaut de construction que doit résoudre le concepteur. Éventuellement, ce dernier peut prévoir une stratégie par défaut en fonction de la sémantique du concept. Par

contre, il semble très difficile de proposer une stratégie par défaut au niveau général, car le problème est étroitement lié à la sémantique.

9. Conclusion

Dans ce chapitre, nous venons de présenter un certain nombre d'éléments importants du noyau de CordiFormes issus du formalisme de la première partie et formant les structures de représentation des connaissances essentielles. Nous avons présenté la notion de concept et de tâche de génération de ces concepts. La génération est automatique (génération récursive) ou spécifique au concept considéré quand le concepteur la trouve plus performante. Ensuite, nous nous sommes intéressés à la génération de l'ensemble des concepts d'une scène. Nous avons vu aussi que ces notions ne sont pas suffisantes, même pour générer des objets très simples (rectangle, segment…). Nous avons donc eu besoin d'introduire la notion de contrainte. Nous avons mis en évidence trois principaux types de contraintes : les contraintes d'initialisation, d'optimisation et de mesure. Finalement, l'ajout des contraintes provoque des modifications au niveau de l'organisation générale de la génération. Nous avons montré que notre méthode consistant à propager les modifications des concepts aux autres concepts grâce aux contraintes s'approche des méthodes utilisées en CSP. Il sera donc intéressant de voir de plus près ces méthodes, en particulier les études permettant d'optimiser la génération en limitant encore plus le nombre de combinaisons explorées telles que l'organisation des concepts composants en fonction de leurs contraintes. De plus, nous avons vu dans la seconde partie que ces méthodes sont adaptées au formalisme flou (CSP flous).

Dans ce que nous venons de présenter, il manque un concept important : le concept de liste. En effet, pour l'instant, les objets de la scène sont connus en type et en quantité au moment de la description. Le nombre d'objets de la scène ne peut pas varier dynamiquement lors d'une génération. Il n'est pas possible de demander un ensemble d'objets d'un ou de plusieurs types donnés sans en connaître exactement le nombre. En conséquence, il n'est actuellement pas possible de donner une description de la forme « *Le nombre de segments est très important* », « *Le nombre de segments est plus important que celui de rectangles* » , « *La scène comporte une dizaine de boîtes* »… Dans cette dernière description, la scène comporte alors un nombre de boîtes compris approximativement entre 9 et 11. Les solutions comportent alors un nombre de boîtes qui peut être différent. Il sera donc très important d'étudier les éléments à introduire ou à modifier dans le noyau pour ajouter la notion de liste d'éléments en nombre inconnu.

Dans le prochain chapitre, nous allons étudier un autre élément très important du noyau de CordiFormes : la description. Nous allons regarder de plus près l'implémentation des propriétés et leur utilisation pour améliorer la génération.

CHAPITRE II.3 : NOYAU, DESCRIPTION ET PROPRIÉTÉS

1. Introduction

Après avoir étudié les concepts, les tâches de génération et les contraintes, intéressons-nous maintenant à la représentation et à la gestion d'une description considérée comme une conjonction de propriétés. Les propriétés que nous présenterons dans ce chapitre correspondent à la mise en œuvre des propriétés définies dans la partie I. Nous étudierons :

- les propriétés élémentaires (section 2) ;
- les relations (section 3) ;
- les propriétés élémentaires quantifiées (section 4) ;
- les propriétés statistiques (section 5) ;
- les propriétés de composition (section 6) ;
- les propriétés modificatrices (section 7).

Nous verrons dans la section 8 que toutes ces propriétés constituent une hiérarchie et ont certains points en commun. Ces études nous montreront que les propriétés exploitent parfaitement les contraintes exposées au chapitre précédent. Cette combinaison entre propriété et contrainte permet de mettre en place un certain nombre d'optimisations (section 9).

2. Les propriétés élémentaires

2.1. Rappels sur la constitution des propriétés élémentaires

Une propriété élémentaire est un énoncé de la forme « C_i de X est f_α m_β P_{ik} » où C_i est un concept, X un objet ou une partie d'objet de la scène, f_α un opérateur flou, m_β un modificateur et P_{ik} une propriété de base qui est une fonction LR définie sur le domaine associé à C_i. L'application du modificateur et de l'opérateur flou sur cet intervalle flou donne un intervalle flou transformé, fonction d'appartenance de la propriété élémentaire. Dans cette section, nous nous intéresserons à la représentation et à la gestion de tous ces éléments.

2.2. Les fonctions d'appartenance

A priori, la fonction d'appartenance est quelconque. Le concepteur peut, s'il le désire, proposer son propre type. Les fonctions d'appartenance doivent seulement posséder un certain nombre de méthodes telles que :

- le calcul du degré d'appartenance d'une valeur à la propriété (μ = mu(t)) ;
- la possibilité de transformer cette fonction en son complémentaire…

Nous avons choisi, dans notre formalisme, d'utiliser les fonctions d'appartenance de type fonction LR pour représenter les ensembles flous. Une telle fonction est un intervalle flou défini par quatre paramètres <α, a, b, β> et deux fonctions L et R. Elle doit aussi respecter les caractéristiques d'une fonction d'appartenance (Figure 103). De plus, il existe un cas particulier de fonction LR : les fonctions LR « filtrées ». Celles-ci ont la particularité de n'être définies que sur une partie du domaine. En dehors de ces limites, le degré d'appartenance est nul. Elles sont utiles en particulier pour les propriétés de comparaison élémentaire (« supérieur à U » et « inférieure à U ») afin que ces dernières restent « du bon côté » de la valeur de référence. En dehors du fait qu'il peut vouloir proposer son propre type de fonction d'appartenance, le concepteur n'a pas, en règle générale, à redéfinir les fonctions LR.

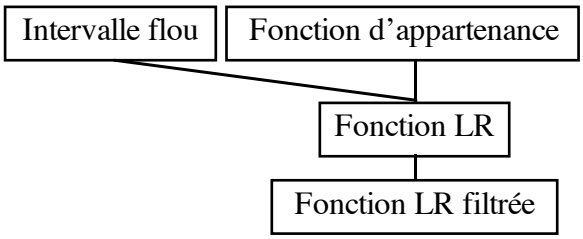

Figure 103. Hiérarchie des fonctions d'appartenance

2.3. Les propriétés de base paramétrées ou non

Dans le chapitre I.3, plusieurs catégories de propriétés de base ont été proposées. Plusieurs méthodes de construction des propriétés paramétrées et des propriétés de base (« important », « faible » et « valide ») ont aussi été données. Ainsi, le noyau CordiFormes propose toutes ces propriétés selon la hiérarchie présentée à la Figure 104. Notons un cas « étrange » : les propriétés « supérieur à » et « inférieur à » sont considérées comme des propriétés non-paramétrées. De par leur mode de construction, elles sont plus proches des propriétés de base que des propriétés paramétrées, car la fonction d'appartenance qui leur est associée est calculée sur la portion de domaine de la même façon que les propriétés de base sur tout le domaine.

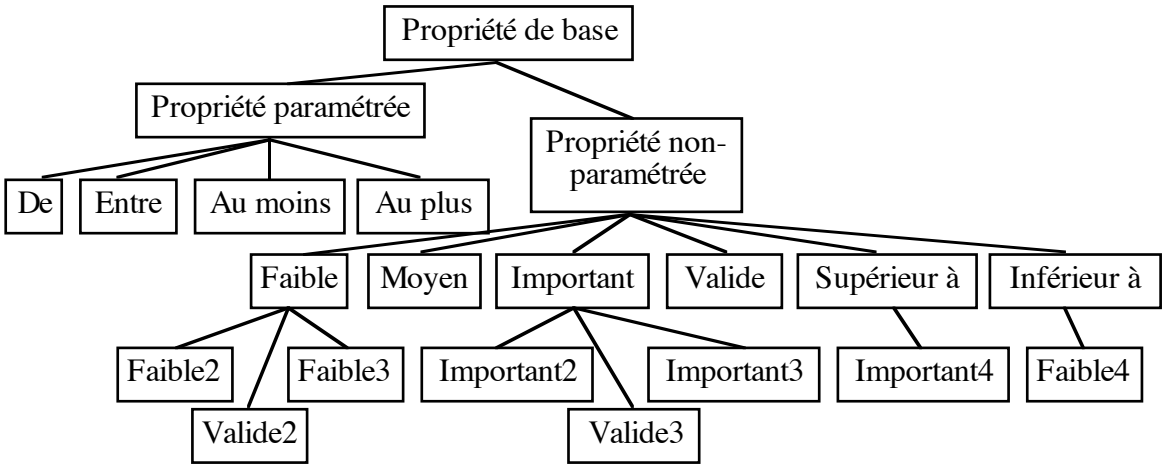

Figure 104. Hiérarchie des propriétés de base paramétrées et non-paramétrées

Une propriété de base est constituée de tous les paramètres introduits dans le chapitre I.3. Nous trouvons en particulier :

- le nom standard (généralement dans « important », « faible », « moyen », « valide »…) ;
- le nom spécifique, en particulier pour les propriétés de base non marquées (par exemple « large » avec « important », « étroit » avec « faible » pour le concept de « largeur ») ;
- la fonction LR associée ;
- les coefficients τ_{ik} et γ_{ik} spécifiques à la propriété de base ;
- un indicateur spécifiant si elle est ou non modifiable ;
- un indicateur spécifiant si elle est non-marquée, marquée ou ne faisant pas partie d'un couple (marquée ; non-marquée)
- le seuil d'acceptation pour la propriété…

Comme pour les fonctions LR, le concepteur n'a généralement pas besoin de redéfinir les propriétés de base. Celles qui sont proposées sont habituellement suffisantes. Elles calculent toutes les coefficients en fonction du domaine. Les propriétés paramétrées demandent, en plus des paramètres classiques, des valeurs de référence. Bien entendu, le concepteur peut modifier certaines valeurs afin de les adapter au domaine d'application.

Remarques :

- Les propriétés de base (non-paramétrées autres que les propriétés de comparaison élémentaires) sont définies pour chacun des concepts. Ceux-ci possèdent une liste de ces propriétés de base. Compte-tenu de la définition d'une propriété de base, seuls les concepts terminaux peuvent en posséder, car elle demande un domaine ordonné.
- Les propriétés paramétrées sont, quant à elles, construites au moment où l'utilisateur les énonce, car elles dépendent des valeurs de référence. Il est possible d'utiliser une propriété paramétrée avec des concepts non-terminaux.

2.4. Les opérateurs

Les opérateurs possibles sont de deux types : les modificateurs et les opérateurs flous (Figure 105). Un opérateur est composé d'un nom et d'un coefficient réel (appelé au chapitre I.3 coefficient de flou j_α pour l'opérateur flou ou coefficient de modification k_β pour le modificateur). Les opérateurs flous et les modificateurs se différencient par la méthode de modification de la propriété de base. Ces méthodes sont décrites dans le chapitre I.3.

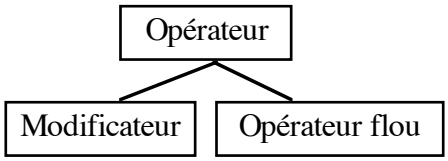

Figure 105. Hiérarchie des opérateurs

Les différents opérateurs (modificateurs et opérateurs flous) existent par défaut. Le noyau contient une liste de modificateurs et une liste d'opérateurs flous. Les opérateurs choisis sont ceux de M_9 et F_6 (Chapitre I.3). Le concepteur a cependant la possibilité de modifier les opérateurs existants (nom ou coefficient), d'en supprimer ou d'en ajouter de nouveaux.

2.5. Construction d'une propriété élémentaire

Il nous reste maintenant à décrire la propriété élémentaire elle-même. Cette propriété est simplement composée d'une propriété de base, d'une référence au concept qu'elle décrit, d'un modificateur et d'un opérateur flou. Pour garder intacte la fonction initiale, la propriété possède une seconde propriété de base correspondant à la propriété de base modifiée par les opérateurs. Elle est appelée propriété de référence. Cette propriété élémentaire possède une méthode permettant d'évaluer le degré d'appartenance de la mesure du concept sur lequel elle porte à sa fonction de référence. Là encore, le concepteur n'a généralement rien à redéfinir. En effet, la sémantique de l'application est particulièrement centrée sur la génération et, surtout, sur les concepts utilisés.

Remarque : L'utilisation d'un autre type de fonction d'appartenance pour une propriété de base (autre que LR) n'est possible que si la propriété n'est pas modifiable. En effet, l'application des opérateurs n'est prévue que pour les fonctions LR. Les opérateurs ne feront aucune modification si la fonction qui leur est proposée n'est pas issue d'une fonction LR.

2.6. Remarque

Habituellement, lorsque l'utilisateur fournit une description comme « *Le segment S1 est long* », le domaine du concept « longueur » est initialisé en fonction d'informations provenant du concept père dans la scène. Cependant, il peut arriver qu'il veuille comparer deux segments qui ne sont pas, par exemple, dans la même zone. Leurs domaines respectifs ne sont donc pas forcément identiques. Il est alors intéressant d'ajouter, ou plutôt de remplacer, les contraintes initialisantes par d'autres contraintes faisant référence non plus au père de chacun mais à leur ancêtre commun le plus proche. Cependant, cette solution n'est pas satisfaisante, car elle n'est plus correcte si les deux types de propriétés (élémentaire et comparaison) sont demandés. Une meilleure solution est de construire un nouveau concept connu uniquement de la propriété. Ce concept n'est pas générateur. La contrainte de mesure se contente alors de reprendre la mesure locale du concept. L'initialisation du domaine de ce nouveau concept est fonction de l'ancêtre commun. Ainsi, dans notre exemple, nous avons les longueurs de segments définies normalement en fonction du père de l'objet. Lorsque l'utilisateur se réfère à une longueur par rapport à un autre ancêtre, le système crée un nouveau concept non-générateur identique au concept « local » avec des contraintes d'initialisation différentes, fonctions de l'ancêtre considéré, et la même valeur de mesure.

3. Les relations

Comme nous l'avons vu dans le chapitre I.6, il existe deux types de relations : les propriétés relatives et les comparaisons. Nous ne considérerons ici que les relations binaires.

3.1. Les relatives

Les propriétés relatives sont, comme nous l'avons vu, très proches des propriétés élémentaires. En fait, elles n'en diffèrent que parce qu'elles sont associées à un concept se rapportant à deux objets de la scène. Il n'y a pas de différence du point de vue de leur structure. Seulement, le concept associé n'est pas générateur et la contrainte de mesure a comme concept de référence les deux objets (ou plutôt un ou plusieurs concepts composants des deux objets).

3.2. Les comparaisons

Nous avons divisé les propriétés de comparaison en deux classes : les comparaisons par différence et les comparaisons par proportion. Ces deux classes diffèrent par la méthode de comparaison et le concept utilisé. Il existe en effet deux concepts, spécifiques à ces propriétés, indépendants du domaine d'application : le concept de différence et le concept de proportion. Leurs domaines sont calculés à partir des domaines des concepts référencés par la comparaison. Les propriétés de comparaison comportent une référence à la propriété de référence, deux références aux concepts pris en compte, l'opérateur de direction et, éventuellement, l'opérateur flou utilisé.

3.2.1 Les comparaisons par différence

La propriété par différence comporte en plus l'opérateur de comparaison. La structure de cette propriété dépend de la méthode utilisée. Nous choisirons ici de traiter les comparaisons par différence à l'aide des comparaisons élémentaires « supérieur à » et « inférieur à » (chapitre I.6, §2.4). La propriété possède alors une table d'équivalence entre les opérateurs de comparaison et les modificateurs. Cette table peut être modifiée, si besoin est, par le concepteur. La propriété paramétrée (en particulier, sa fonction d'appartenance), définie sur le domaine du concept de différence, est donc calculée par la propriété de comparaison. Cette propriété paramétrée est éventuellement modifiée par l'opérateur flou.

3.2.2 Les comparaisons par proportion

La propriété par proportion comporte, quant à elle, le coefficient de comparaison utilisée. Pour cette propriété, la fonction d'appartenance est définitivement calculée en fonction de la direction de comparaison et du coefficient (chapitre I.6, §2.5) sur le domaine du concept de proportion. Nous obtenons alors une fonction d'appartenance modifiée par l'opérateur flou.

3.2.3 Évaluation des propriétés de comparaison

Comme pour les propriétés relatives, les concepts sur lesquels se basent les propriétés de comparaison ne sont pas des concepts générateurs. Il existe donc pour chacun une contrainte de mesure spécifique. Celle-ci a comme concepts de référence les concepts utilisés dans la comparaison. La fonction de calcul de cette contrainte dépend de la comparaison utilisée (la différence des mesures pour la différence et le couple des mesures pour la proportion).

4. Les propriétés quantifiées

4.1. Rappels sur la constitution des propriétés quantifiées

Une propriété élémentaire quantifiée est un énoncé de la forme « C de Q_t X est A » où Q_t est un quantificateur et les autres éléments sont ceux définis pour les propriétés élémentaires.

4.2. Les quantificateurs

Les quantificateurs sont ici composés d'un nom et d'une fonction d'appartenance dépendant du nombre de concepts pris en compte (§2 du chapitre I.5). La forme de la fonction dépend de la méthode utilisée.

Par exemple, pour la méthode proposée dans le chapitre I.5, nous avons choisi de représenter les quantificateurs par (m étant le nombre de concepts étudiés) :

- La_Plupart(m) = <m/4,3*m/4,m,0> LR ;
- Tous(m) = <0,m,m,0> LR ;
- Au_Moins(k,m) = <0,k,m,0> LR…

Remarque : Dans le chapitre I.5, les figures illustrant la méthode d'interprétation des propriétés quantifiées ont été obtenues à l'aide de ces fonctions.

Un quantificateur est donc un opérateur possédant, en plus, une fonction d'appartenance. Selon la méthode d'application qui lui est associée, le coefficient n'est pas toujours utilisé. Nous obtenons alors la hiérarchie présentée dans la Figure 106.

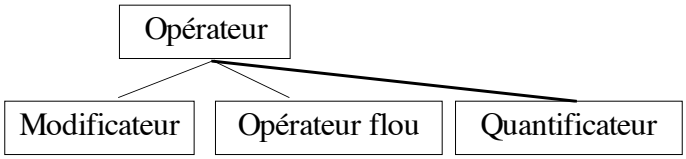

Figure 106. Hiérarchie des opérateurs avec quantificateurs

4.3. Construction d'une propriété quantifiée

Une propriété quantifiée est constituée du quantificateur et d'une liste de propriétés sur lesquelles porte la quantification. Nous pouvons aussi bien avoir des propriétés élémentaires que des propriétés relatives ou encore de comparaison. L'évaluation d'une telle propriété est un cas particulier. Pour toutes les propriétés étudiées jusqu'ici, elle consistait à déterminer le degré d'appartenance d'une mesure à une fonction d'appartenance donnée. Nous avons vu dans le paragraphe 2 du chapitre I.5 la difficulté de cette évaluation. Elle reste en effet un problème. Le traitement, légèrement différent de celui des propriétés élémentaires, consiste à déterminer un degré d'appartenance à partir de ceux des propriétés de la liste. L'évaluation de celles-ci est effectuée, non pas par la tâche de génération des concepts, mais par la propriété quantifiée. En effet, comme nous autorisons ici qu'une propriété ne soit pas satisfaite (sauf avec « pour tout »), l'évaluation ne doit alors pas être gérée par la tâche de génération. Sinon, cette tâche prendrait cela pour un échec et essayerait une autre mesure. Une fois les propriétés évaluées, il suffit d'appliquer l'algorithme adapté.

5. Les propriétés statistiques

Une propriété statistique est associée à un concept spécifique. Le domaine de ce dernier dépend de la fonction statistique FS utilisée. Ce concept ne peut évidemment pas être générateur. La contrainte de mesure associée à ce concept a comme concepts de référence les concepts étudiés. La fonction de calcul dépend quant à elle de FS.

6. Composition de propriétés

Les propriétés de composition sont des propriétés binaires. Elles comprennent deux références à des propriétés quelconques. Il existe principalement quatre cas de propriétés de comparaison qui diffèrent simplement par leur méthode d'évaluation du degré d'appartenance (C*f*. Chapitre I.2 et I.6) : le "Et", le ".", le "Ou" et le "Ou bien". Les propriétés de composition par alternative ("Ou" et "Ou bien") posent des problèmes au niveau de la génération. Il n'est pas toujours évident de bien gérer ces propriétés. Elles empêchent beaucoup d'optimisations. Souvent, lorsqu'un concept ne vérifie pas une propriété, cela ne veut pas dire que l'on doit le rejeter, car, s'il fait partie d'une alternative, il peut encore être conservé. Nous risquons alors de produire jusqu'au bout des solutions fausses. Nous avons vu aussi au chapitre I.6 que la gestion des propriétés de composition "Et" n'est pas résolue. Nous allons donc poser que seul l'opérateur de composition "." est géré. Les propriétés sont alors dans une liste. La description n'est qu'une conjonction de propriétés.

7. Modification d'une description

Une propriété de modification est une propriété portant sur un concept. Le traitement que nous avons choisi (C*f*. Chapitre I.5, §4) s'attache à trouver une propriété de comparaison élémentaire équivalente. Une propriété modificatrice possède donc une référence au concept modifié, l'opérateur de comparaison, l'opérateur de direction et la propriété de comparaison élémentaire calculée. L'évaluation de cette propriété correspond simplement à l'évaluation de la propriété de comparaison élémentaire qui lui est équivalente.

8. Les propriétés

Nous venons de recenser la plupart des propriétés d'une description. Compte-tenu de nos remarques sur les relatives, nous considérerons ces propriétés plutôt comme des propriétés élémentaires. Les propriétés s'organisent donc selon la hiérarchie de la Figure 107. Chaque propriété possède les informations suivantes :

- un concept de référence ;
- le nom correspondant à la chaîne de caractère de la forme spécifique de l'énoncé de la propriété. Cette chaîne est « construite » à partir des concepts et des opérateurs utilisés ;
- une fonction permettant de calculer le degré d'appartenance de la scène (ou d'un concept particulier)

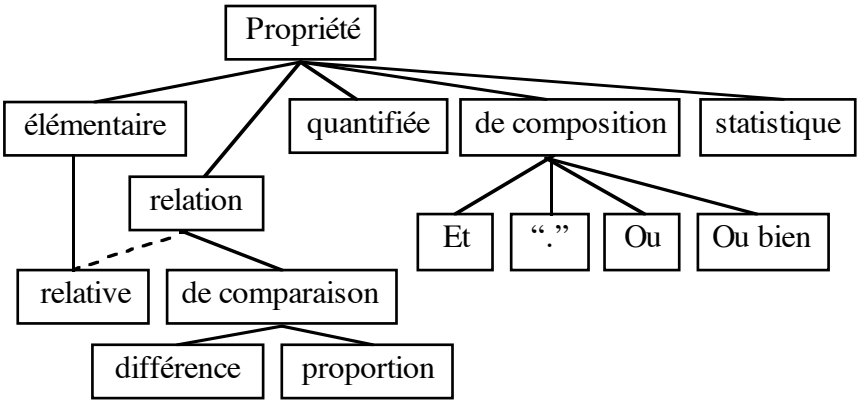

Figure 107. Hiérarchie des propriétés

9. Optimisations liées aux propriétés

Actuellement, les propriétés sont utilisées après la génération de la scène, c'est-à-dire qu'elles sont utilisées en vérification a posteriori. Cependant, elles sont utilisables pour optimiser la génération des solutions.

9.1. Propriétés élémentaires

Les propriétés élémentaires sont particulièrement utiles. Le principe d'optimisation à partir de celles-ci est simple. Il consiste à éviter de construire un objet sachant qu'il sera forcément rejeté. Une propriété élémentaire est représentée par un intervalle flou défini par une fonction LR. Toute valeur située à l'intérieur du support, et plus précisément à l'intérieur de l'intervalle défini par la *S*-coupe (S étant le seuil d'acceptation), est solution vis-à-vis de la propriété. Par conséquent, lorsque l'utilisateur propose une propriété élémentaire sur un concept générateur, il ne faut proposer que les valeurs appartenant à cet intervalle ([GAT96]). La Figure 108 illustre le cas où l'utilisateur demande une propriété P (par exemple « *La hauteur est faible* »). La zone non-hachurée comprend toutes les valeurs vérifiant "au moins un peu" la propriété, c'est-à-dire toutes les valeurs dont le degré d'appartenance est supérieur au seuil d'acceptation de la propriété (dans la figure le seuil est à 0,1).

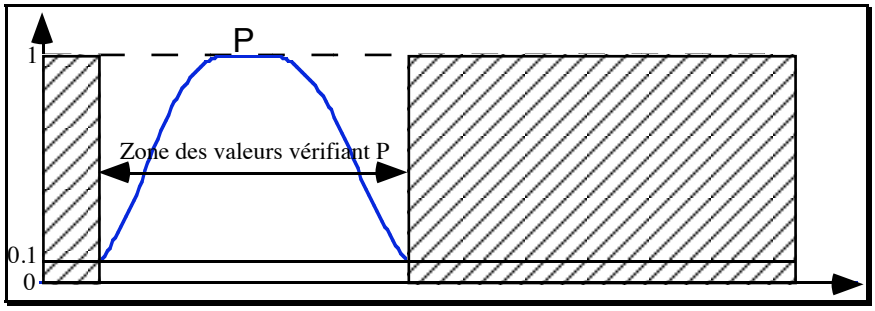

Figure 108. Zone du domaine où toutes les valeurs vérifient au moins un peu une propriété P

Les propriétés élémentaires sur un concept permettent donc de réduire l'ensemble de ses valeurs possibles. Par ailleurs, nous avons déjà défini dans le chapitre précédent trois domaines pour un concept : le domaine d'origine, le domaine de travail et le domaine courant. Les propriétés sont nécessairement définies sur le domaine de travail, c'est-à-dire sur le domaine fixé après l'application des contraintes d'initialisation. Nous introduisons donc à ce niveau un nouveau domaine : le domaine décrit.

Définition II.3.1 : Le *domaine décrit* est obtenu après réduction du domaine de travail par les propriétés.

Remarques :

- Cette optimisation a lieu aussi pour les concepts non-générateurs. En effet, nous supposons que la validation d'une valeur est effective seulement lorsque les contraintes de mesure sont valides. Or, l'application de telles contraintes n'est valide que si la valeur calculée appartient au domaine courant du concept cible. Cela nous permet ainsi de rejeter une valeur plus tôt.
- Nous pouvons imaginer que d'autres types de propriétés peuvent permettre de faire une telle optimisation.

Finalement, nous avons quatre domaines dont la gestion est illustrée par la Figure 109.

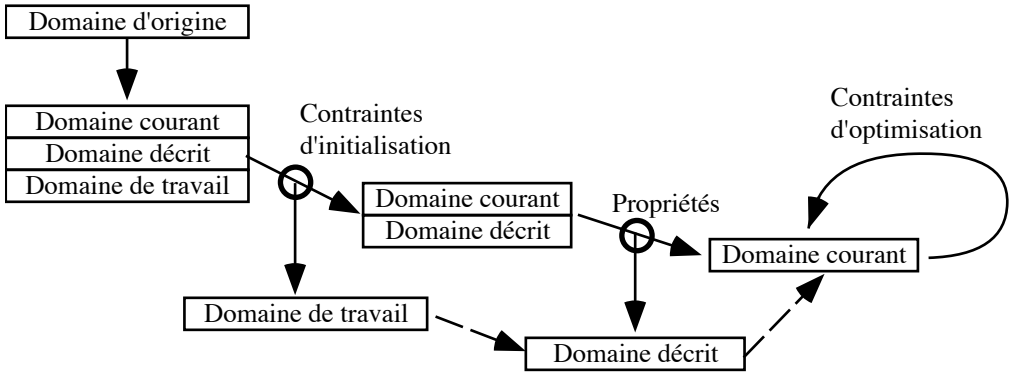

Figure 109. Contraintes, propriétés et domaines d'un concept

9.2. Génération de concepts terminaux

Les propriétés élémentaires, et plus généralement les ensembles flous associés à une propriété, permettent aussi d'optimiser les algorithmes de génération sur des concepts terminaux. Toutes les méthodes dites de « défuzzification » dans les modèles flous (présentées en conclusion de la première partie) sont intéressantes à ce niveau. Adaptées aux caractéristiques de la génération, elles permettent de faire une équivalence possibilité/probabilité ([Yag82] et [DPS93]) en classe aléatoire. En classe énumération, elles permettent de proposer d'abord les valeurs qui ont un degré d'appartenance maximal par rapport à l'ensemble des propriétés puis de continuer par degré décroissant. Comme la description est une conjonction de propriétés, celles portant sur un concept donné constituent une description qu'on peut appelée « locale ». Cette méthode permet d'essayer d'abord (ou le plus souvent) les meilleures valeurs du point de vue de la description locale mais il n'est pas garanti qu'elles sont les meilleures pour la solution générale. Enfin, notons que ces méthodes permettent difficilement un contrôle de la génération basé sur d'autres objectifs que le degré d'appartenance (ordre des mesures selon leur valeur…).

9.3. Concepts générateurs

Lorsqu'il y a plusieurs ensembles possibles de concepts générateurs, les propriétés de la description peuvent permettre d'en sélectionner un plus particulièrement parce qu'apparemment mieux adapté. En effet, il est préférable d'avoir des concepts générateurs faisant l'objet d'une description, notamment à cause de l'optimisation proposée ci-dessus. Leurs domaines sont ainsi réduits et, vis-à-vis de leurs propriétés respectives, ils ne produiront que des valeurs correctes. Prenons l'exemple de la génération de segments dans un plan. Nous pouvons définir au moins deux ensembles de concepts générateurs : {l'origine, l'extrémité} (1) ou {l'origine, la pente, la longueur} (2). Si l'utilisateur demande « *Un segment long* », il est préférable de prendre le second ensemble. Par contre, s'il demande « *Un segment dont les*

extrémités sont placées sur un cercle », le premier ensemble apparaît alors plus adapté. Dans le cas où le choix n'est pas aussi trivial (plusieurs ensembles envisageables), il est possible d'utiliser quelques heuristiques. Une solution consiste alors à évaluer le nombre de combinaisons possibles avec chacun des ensembles candidats et de prendre celui qui en a le moins (donc moins d'objets à construire). Mais d'autres solutions sont envisageables !

Remarque : Pour chaque concept, l'utilisateur peut décrire une tâche de génération privilégiée. Une idée serait plutôt de proposer les tâches les plus adaptées en fonction des ensembles de concepts générateurs choisis. Ainsi, non seulement l'ensemble choisi sera celui générant le moins de solutions mais, en plus, la tâche de génération sera optimisée. Inversement, le type de tâche de génération pour un ensemble peut être un critère de sélection.

9.4. Relations

Les relations permettent aussi de mettre en place des optimisations. Il est notamment souvent possible de leur associer des contraintes spécifiques. Par exemple, avec les propriétés de comparaison, nous pouvons associer deux contraintes pour réduire le domaine courant d'un concept connaissant l'autre. Plus précisément, elles permettent, connaissant la mesure d'un des deux concepts et l'intervalle de différence (ou de proportion) utilisé, de réduire le domaine de l'autre concept. Par exemple, si deux concepts A dans $[Bm_A, BM_A]$ et B dans $[Bm_B,BM_B]$ sont tels que leur différence doit être dans [i,j] (support de l'intervalle flou) alors, connaissant m_B, nous avons « nécessairement » m_A dans $[m_B+i, m_B+j]\cup[m_B-j, m_B-i]$ (et inversement, connaissant m_A, on obtient m_B dans $[m_A+i, m_A+j]\cup[m_A-j, m_A-i]$).

9.5. Propriétés élémentaires quantifiées

Les propriétés élémentaires quantifiées permettent de mettre en place des optimisations. Par exemple, lorsque le quantificateur utilisé est « Pour tout » ($\forall$), par définition, tous les concepts doivent vérifier la propriété sur laquelle il porte. Dans ce cas, il est intéressant d'ajouter réellement cette propriété dans la description pour chacun des concepts. Ainsi, les optimisations liées à cette propriété peuvent être mises en place. Par exemple, si la description est « *La longueur de tous les segments est importante* », l'optimisation liée à la propriété élémentaire « *La longueur du segment est importante* » peut être utilisée afin de réduire tous les domaines liés à la longueur. D'autres méthodes d'optimisation existent au cours de la génération. En particulier, lorsqu'une propriété ne pourra plus être vérifiée compte-tenu des actions restant à effectuer, la génération devra explorer une nouvelle configuration.

9.6. Propriétés modificatrices

Les propriétés modificatrices permettent, aussi, de mettre en place des optimisations. En effet, l'intervalle flou calculé à partir de ce type de propriété contient des valeurs que

l'utilisateur préfère (pour le moment !). L'idée est donc de réduire l'intervalle du domaine associé à ce concept. Lorsque l'utilisateur demande une modification, le système ajoute une contrainte d'optimisation permettant de réduire le domaine de la même façon qu'avec les propriétés élémentaires.

10. Conclusion

Nous venons d'exposer l'implémentation et l'utilisation des propriétés que le noyau de CordiFormes doit proposer. Nous avons montré aussi que la structure de ce noyau accepte tous les types de propriétés. De plus, celles-ci permettent d'effectuer un certain nombre d'optimisations pour la génération.

Cependant, nous avons vu au chapitre I.5 que la formalisation des propriétés autres qu'élémentaires n'est pas toujours satisfaisante. Il faudra donc tenir compte de l'évolution de ce formalisme au niveau du noyau (en particulier au niveau des propriétés quantifiées et des relations binaires mais aussi n-aires). Nous avons cependant montré que le formalisme actuel permet de gérer un bon nombre de situations.

Au cours de ce travail, nous avons fait l'hypothèse que la description est composée d'une conjonction élémentaire (opérateur ".") de propriétés. Il sera important d'étudier l'utilisation des autres opérateurs et, en particulier, de la disjonction (aussi bien au niveau des propriétés que des objets).

Dans tout ce que nous avons exposé ici, nous n'avons pas abordé les optimisations liées au comportement des propriétés lors de la génération ([Col90], [Des95a] et surtout [DeM97a] qui propose une extension de ces études aux propriétés de notre formalisme, c'est-à-dire aux propriétés définies à l'aide des ensembles flous comme les propriétés élémentaires). Il serait intéressant d'appliquer toutes les optimisations ayant rapport à la stabilité, la périodicité et autres comportements que peut avoir une propriété et la valeur d'un domaine pendant la construction d'un concept non-terminal ou de la scène. Il faudrait aussi étudier l'influence des contraintes dans ces caractéristiques.

Nous allons maintenant étudier d'autres éléments du noyau permettant de gérer tout ce que nous venons de traiter dans ces deux chapitres.

CHAPITRE II.4 : NOYAUX ET MODULES DÉCLARATIFS

1. Introduction

Dans les chapitres précédents, nous avons décrit les principaux éléments du noyau, c'est-à-dire les fonctionnements et la structure des concepts, des tâches de génération et des propriétés de la description. Un modeleur déclaratif (et donc le noyau) est constitué de trois modules principaux :

- le module de gestion de la description (section 2) ;
- le module de génération, appelé aussi module de calcul(section 3) ;
- le module de prise de connaissance (section 5).

Leur rôle est d'aider la manipulation des structures du noyau et de proposer des opérations de haut niveau en effectuant un certain nombre des opérations de bas niveau. Nous reviendrons aussi sur le problème de la négation linguistique et nous verrons à quel moment et dans quel module, elle est gérée (section 4). Nous terminerons notre description des éléments du noyau de CordiFormes en détaillant la base de connaissances (section 5).

2. Module de description

La modélisation déclarative «*repose sur l'idée que nous pouvons appréhender le "monde" autrement que par sa description géométrique : nous pouvons le percevoir par ses propriétés, par ses caractéristiques, c'est-à-dire non pas seulement par l'apparence qu'il nous présente mais par les mécanismes et les contraintes qui font qu'il nous apparaît sous cette forme. Ainsi, nous nous plaçons à un plus haut niveau d'abstraction*» [LeG90].

La phase de description permet la gestion de la description de l'utilisateur. A l'aide des outils les plus adaptés au domaine d'application, il entre, d'une façon assez naturelle, les diverses propriétés qui constituerons la description de la scène qu'il désire obtenir. Les outils de description permettent aussi de modifier ou, éventuellement, de supprimer des propriétés. A ce niveau, le système effectue une vérification de la description afin de détecter contradictions, incohérence ou ambiguïté. Parfois, lorsque c'est nécessaire, le modeleur demande à l'utilisateur des précisions afin de tenter de résoudre ces problèmes. La description est finalement envoyée au module d'exploration (de calcul) afin qu'il génère les solutions (Figure 110).

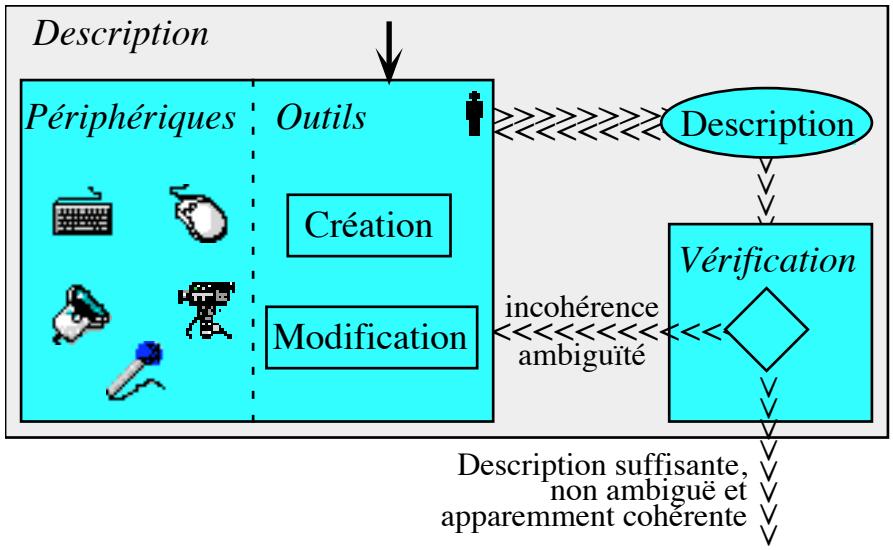

Figure 110. La phase de description [CDMM97b]

Remarque : La description qu'il fournit n'est pas forcément totalement cohérente, car certains cas ne peuvent être détecter que lors de la génération.

Le module de description gère la description de la scène et toutes les interventions sur la table des objets. C'est un élément essentiel dans un modeleur déclaratif. Ses principales fonctions concernent de la gestion des objets de la scène ainsi que celle de la description et de ses propriétés.

2.1. Gestion des objets de la scène

Toutes les opérations effectuées sur la table des objets passent par le module de description. Ce dernier possède une liste des objets disponibles, c'est-à-dire tous les types d'objets que l'utilisateur pourra utiliser pour construire sa scène. Il possède aussi l'ensemble des graphes sémantiques existants. Toutes ces informations sont fournies par le concepteur lors de la création de ce module.

L'ajout d'un nouvel objet dans la table s'effectue en indiquant le nom de son concept et son père dans la hiérarchie de la scène. Avant que l'ajout soit effectif, le module vérifie dans la table des graphes sémantiques la validité de cette opération. Si elle est valide, l'objet est créé et ajouté dans la table. De plus, à partir des informations fournies par ces graphes, le module se charge de créer et de mettre en place les contraintes d'initialisation nécessaires.

La modification d'un objet ne consiste pas uniquement à le changer de place dans la hiérarchie. Le module se charge alors de valider cette opération, de supprimer les contraintes liées à cet objet et de créer les nouvelles.

2.2. Gestion des propriétés et de la description

Le module de description gère aussi la description de l'utilisateur. Il propose les outils nécessaires pour ajouter, modifier ou supprimer une propriété. En particulier, il gère la cohérence des propriétés entre elles mais aussi par rapport aux objets de la scène. La Figure 111 résume les différentes fonctions que nous venons d'évoquer.

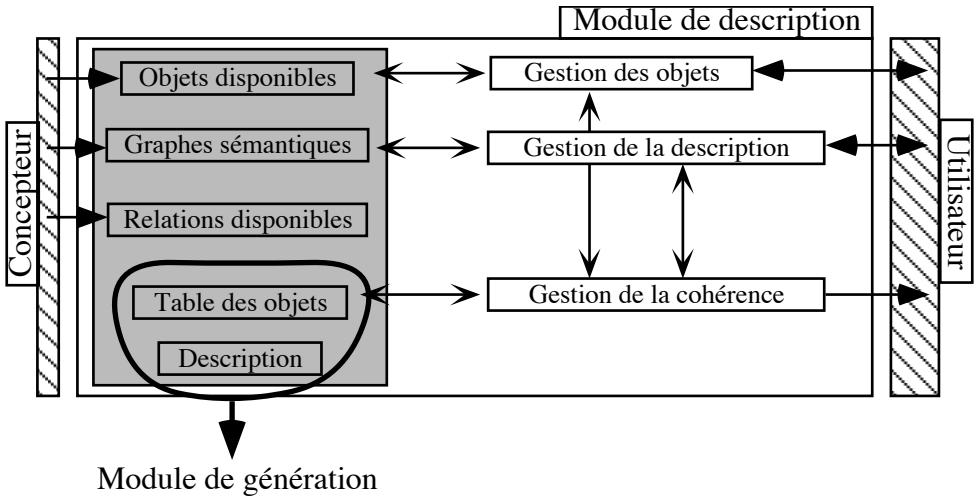

Figure 111. Le module de description

Ce module propose des éléments utiles pour les éventuels outils d'interface. Il fournit, par exemple, la liste des modificateurs et des opérateurs flous. Il indique aussi les concepts, donnés par le concepteur, susceptibles de décrire des relations entre des objets. Il possède des méthodes permettant de créer les propriétés à partir des éléments qui les définissent (les opérateurs utilisés, les concepts mis en cause…). Ce module possède notamment une méthode pour convertir des chaînes de caractères respectant une forme donnée en la propriété correspondante. Il prend aussi en compte la gestion des propriétés de modification. Nous avons signalé, au chapitre I.5, le problème de l'application successive de propriétés de modification. Nous avons montré que l'ajout d'une propriété sur un domaine en comportant déjà une autre, pose certains problèmes. Généralement, il est préférable d'effacer la ou les propriétés précédentes.

Remarque : La table des objets et la description forment le modèle de description. Ce dernier est fourni au module de génération afin que celui-ci génère les scènes correspondantes.

3. Module de génération

Le module de génération prend en charge tous les éléments concernant la phase de génération du modeleur. Il comprend principalement un chef d'orchestre s'occupant d'organiser les tâches pour générer la scène. Ce module permet d'obtenir la première scène solution ainsi que la suivante. Il permet aussi de générer un objet précis de la scène et de changer le groupe de

génération courant. Si des informations manquent, il récupère les compléments d'information nécessaires pour débloquer la situation. La Figure 112 résume les différentes fonctions que nous venons d'évoquer.

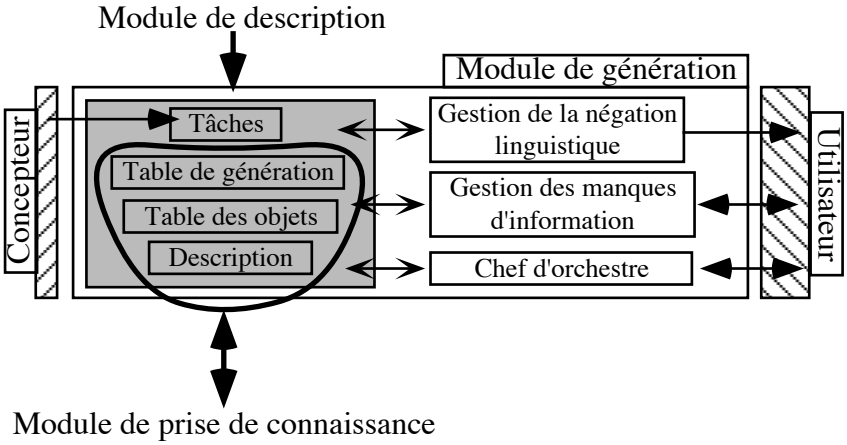

Figure 112. Module de génération

Remarque : Le modèle de description et la table de génération forment le modèle de génération. Ce dernier est utilisé par le module de prise de connaissance pour proposer les solutions à l'utilisateur.

4. Gestion de la négation linguistique

Nous avons vu dans le chapitre I.4 que la négation logique standard n'est pas suffisante. Nous avons alors présenté une méthode de négation linguistique. Le principe est le suivant : lorsqu'une négation est détectée, le système détermine toutes les propriétés plausibles comme négation implicite. L'utilisateur sélectionne celle qui lui semble la plus proche de son idée.

En première analyse, ce traitement est réalisé au moment de la gestion de la description, c'est-à-dire dans le module de description. Or, ce n'est possible que dans certains cas particuliers. En effet, il n'est pas possible dans le cas général d'effectuer cette étude avant de commencer la génération. La détermination des propriétés plausibles s'effectue en construisant toutes les propriétés élémentaires possibles puis en éliminant celles qui sont trop similaires à la propriété niée. Or cette construction n'est possible que lorsque le domaine est définitivement initialisé, c'est-à-dire lorsque toutes les contraintes d'initialisation ont été appliquées. L'application de ces contraintes n'est effectuée que pendant la génération et, pour certains objets, située profondement dans la hiérarchie, autrement dit « longtemps » après le début de la génération. Par conséquent, la gestion de la négation linguistique ne peut se faire qu'au cours de la génération (Figure 112).

Remarque : Une solution est néanmoins possible pour placer cette étude au niveau de la description. Il suffit de travailler les propriétés élémentaires sur un domaine $[0,1]_u$. Ainsi, la propriété choisie sera réajustée lorsque le domaine sera devenu définitif. Cependant, cette méthode ne fonctionne que dans un cas particulier. En effet, il faut que la construction des fonctions d'appartenance soit faite en fonction de la « forme » du domaine et soit produite de manière automatique. A partir du moment où cette construction fait référence à une valeur réelle, la méthode n'est plus envisageable. De plus, si on veut pouvoir proposer aussi des propriétés paramétrées comme propriétés plausibles, elle n'a pas de sens. Il faut éviter de proposer une propriété comme « entre 0.1239564 et 0.536985 » en expliquant à l'utilisateur que ce ne sont que des valeurs provisoires (!). Enfin, cette méthode ne peut gérer la négation de propriétés paramétrées comme « *La hauteur n'est pas entre 10 et 50 mètres* » avec seulement des valeurs comprises entre 0 et 1.

5. Module de prise de connaissance

5.1. La prise de connaissance dans les modeleurs déclaratifs

En phase de prise de connaissance, nous trouvons plusieurs classes d'outils ([CDMM97b], Figure 113). Il existe un ensemble d'outils permettant d'explorer l'ensemble des solutions. L'utilisateur explore cet ensemble :

- en spécifiant un *ordre de parcours* des solutions particulier ;
- en filtrant certaines solutions selon des critères spécifiques (*raffinement des solutions*) ;
- en modifiant une solution par *retouche contrôlée* (la modification d'une solution, « manuelle » mais contrôlée, est vue comme un moyen de parcours des solutions) ;
- en modifiant une solution par *modification déclarative* en utilisant les propriétés modificatrices.

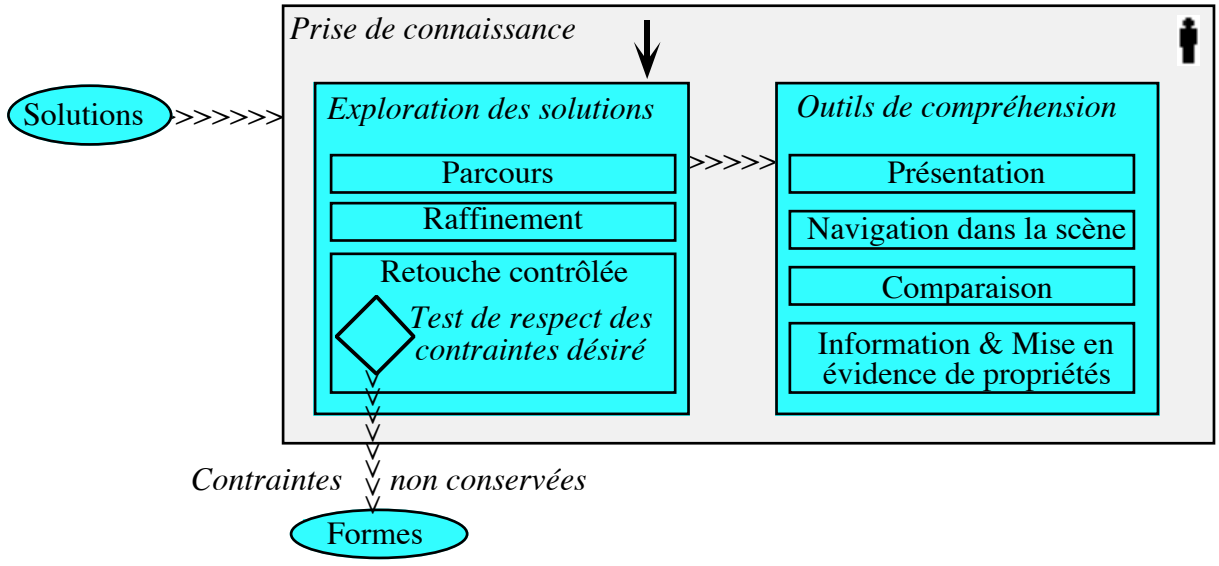

Figure 113. La phase de prise de connaissance [CDMM97b]

S'il connaît parfaitement leur description, puisqu'il l'a donnée au modeleur, l'utilisateur n'en connaît pas pour autant leur aspect. Il est très difficile de manipuler, d'appréhender, avec des outils conventionnels des formes que l'on ne connaît pas. De façon générale, comme avec tout système de haut niveau, présenter une solution sans explication n'est pas suffisant. Aussi, les modeleurs déclaratifs mettent à la disposition du concepteur une panoplie d'outils de compréhension de haut niveau pour l'aider à appréhender la structure et les propriétés (demandées ou non) des solutions ([CDMM97b]). Ces outils permettent de présenter une solution, de mettre en évidence les propriétés de la description ou d'autres informations pertinentes (bon point de vue, bons modes de visualisation…) et de la comparer avec d'autres déjà rencontrées.

Remarque : L'ensemble des solutions n'est pas toujours connu entièrement au moment de la prise de connaissance. Ces outils ne sont donc pas toujours exploitables à moins que le module de calcul puisse s'adapter dynamiquement en fonction des demandes de la phase de prise de connaissance.

5.2. La présentation des solutions

Le module de prise de connaissance présente donc, comme son nom l'indique, les solutions à l'utilisateur. Pour cela, il fournit les informations nécessaires et, en particulier, le modèle géométrique de la scène ainsi que des informations supplémentaires issues du processus déclaratif. A partir des valeurs des mesures des différents objets de la scène qui ne sont pas des ébauches (par rapport au niveau d'avancement de la génération), il construit la scène selon un modèle géométrique donné. Les algorithmes de passage des modèles de représentation des objets au modèle géométrique global sont déterminés par le concepteur. En plus du modèle géométrique, le module fournit des informations supplémentaires spécifiques au processus déclaratif. Il propose en particulier :

- la description ;
- les degrés d'appartenance de la scène (des objets de la scène) aux diverses propriétés ;
- les mesures des concepts faisant l'objet d'une description ;
- le degré d'appartenance global de la scène à la description…

Enfin, ce module s'occupe, pour le groupe courant, de la validation des objets présentés. En effet, l'utilisateur intervient au niveau de la prise de connaissance en indiquant les objets qu'il désire conserver et ceux qu'il refuse. Il peut aussi donner son avis sur la scène proposée. Cet avis peut être exploité par le concepteur afin de mettre en place des optimisations et, éventuellement, des méthodes d'apprentissage. La Figure 114 résume les différentes fonctions que nous venons d'évoquer.

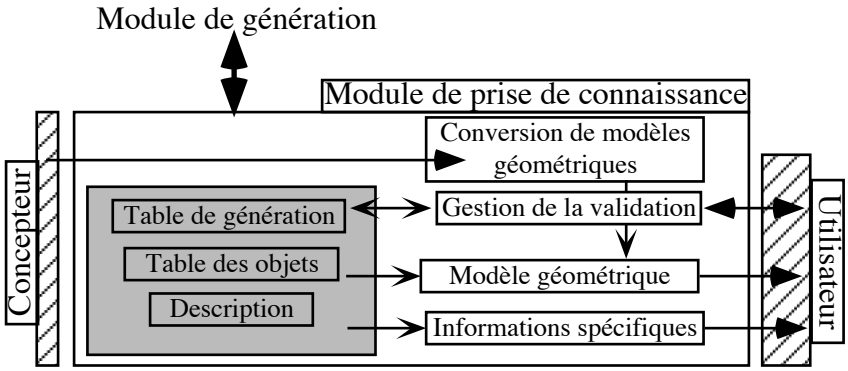

Figure 114. Module de prise de connaissance

Remarques :

- Le modèle géométrique produit n'est pas forcément toujours le même. Il dépend de la scène à afficher et de l'état d'avancement de la génération. En effet, lorsque le modeleur travaille par niveau de détails, le modèle associé à un niveau donné est parfois différent de celui du niveau précédent ou du niveau suivant.
- Tout ce qui concerne les fonctions de présentation de la prise de connaissance concerne plutôt la coche d'interface.

6. La base de connaissances

La base de connaissances est aussi un élément important du noyau. Elle contient en particulier un certain nombre de concepts « classiques » que l'on retrouve souvent lors de la constitution de modeleurs et dans les différents domaines d'application. Cette connaissance est appelée « l'ontologie ». Elle permet d'améliorer la rapidité de développement en proposant un ensemble de concepts courants. Le concepteur peut ainsi les utiliser directement, modifier certains éléments ou les enrichir. A priori, nous avons, à travers les différentes applications développées ou en cours de développement en modélisation déclarative, un certain nombre de connaissances qui peuvent être introduites dans cette base : modélisation géométrique, CAO, architecture, modélisation déclarative…

Les principaux types d'éléments qu'on peut y trouver sont (Figure 115) :

- des concepts, des tâches de génération spécifiques et des graphes sémantiques ;
- des outils d'interface et de dialogue permettant de saisir des descriptions ou de présenter des scènes selon des modes spécifiques ;
- une multitude d'autres « objets » comme des modèles géométriques importants, des tables de correspondance permettant de faire différentes conversions, des outils pour mettre en place d'autres méthodes de génération (en particulier les arbres d'énumération vus au chapitre II.2)…

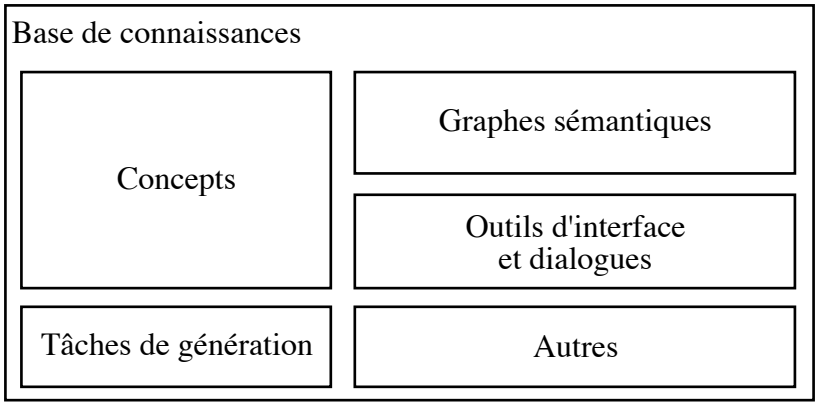

Figure 115. Éléments de la base de connaissances

Par exemple, il est indispensable d'avoir dans cette base les concepts de positionnement dans des espaces 2D et 3D. Elle propose donc des concepts correspondant aux boîtes englobantes ainsi que les graphes sémantiques associés. Ces graphes contiennent en particulier les contraintes d'initialisation des objets que représentent ces boîtes (elles ne sont que des ébauches d'autres objets). De plus, pour prendre connaissance des scènes géométriques en trois dimensions, la base propose des outils de visualisation et d'exploration d'univers en 3D comme la possibilité de générer des fichiers au format VRML, l'utilisation d'algorithmes de visualisation… Tous ces éléments pourront ensuite être exploités par les autres couches de la plate-forme.

Cette base ne doit, bien évidemment, pas rester figée. Le concepteur doit pouvoir la mettre à jour en y incluant les éléments qu'il utilise beaucoup. Il dispose d'outils pour explorer cette base, classer les éléments et y ajouter ses propres connaissances.

7. Conclusion

Nous avons présenté d'abord les trois modules déclaratifs puis la base de connaissances. Les premiers permettent de manipuler les concepts, tâches et autres contraintes par rapport au modeleur déclaratif et la seconde apporte un ensemble de connaissances permettant de construire des concepts complexes plus rapidement. Ces modules sont des éléments importants du noyau. Ils proposent des méthodes de plus haut niveau pour manipuler les connaissances. Nous avons présenté ici l'ensemble minimal des actions qu'ils doivent pouvoir effectuer. Il serait intéressant de compléter ces listes afin de proposer des modules plus complets. Maintenant, nous intéressons nous aux deux autres couches de CordiFormes, c'est-à-dire la couche d'interface et la couche de prototype.

CHAPITRE II.5 : DIALOGUES ET OUTILS PROTOTYPES

1. Introduction

Après avoir étudié le noyau de CordiFormes, nous intéressons nous maintenant aux deux autres couches. Nous aborderons d'abord la seconde couche de la plate-forme où des éléments d'interface sont proposés au concepteur pour gérer les éléments du noyau (section 2). Puis, nous examinerons la troisième couche, c'est-à-dire la constitution de l'application prototype permettant d'évaluer un modeleur déclaratif (section 3).

2. Utilisation de dialogues standard

2.1. Introduction

La seconde couche de CordiFormes propose donc les outils d'interface avec l'utilisateur. Ces derniers se regroupent en trois catégories :

1. les outils de description ;
2. les outils de génération ;
3. les outils de prise de connaissance.

Ces outils sont de deux niveaux : les outils d'interface (éléments de dialogues) d'une part et les fenêtres et dialogues standard d'autre part. Bien évidemment, les seconds s'utilisent sur les premiers ! Après la présentation de quelques exemples de dialogues pour chacune des catégories d'outils, nous étudierons les relations qui existent entre ces outils ainsi que les modules déclaratifs du noyau.

Remarque : Nous proposons ici des outils de relativement bas niveau. Cependant, cette couche devra aussi pouvoir proposer des outils de description et de prise de connaissance de plus haut niveau présentés dans beaucoup de travaux effectués en synthèse d'images et en CAO tels que [SDS93], [GCR93], [ZHH96], [RMS96], [RMD96], [KGC97] ou [PRJ97]).

2.2. Saisie de la description

Les outils de saisie de la description sont de deux catégories : la saisie des objets de la scène et la saisie des propriétés. Ils sont l'intermédiaire entre la description de l'utilisateur et le module de description du noyau.

2.2.1 Éléments d'interface

A ce niveau, les dialogues utilisent plusieurs outils d'interfaces classiques : la liste des concepts disponibles, la liste des objets déjà décrits et la liste des propriétés déjà énoncées. Les concepts disponibles sont les objets qui peuvent être présents dans la scène. Toutes ces informations sont fournies par le module de description et plus précisément par la table des objets et la liste des propriétés.

2.2.2 Gestion des objets

Pour qu'une description soit possible, il faut des objets dans la scène. La seconde couche propose un dialogue permettant d'ajouter ces objets. Les informations nécessaires sont uniquement le nom de l'objet, son « père » dans la hiérarchie de la scène (par défaut l'objet « scène ») et le concept qu'il représente (Figure 116).

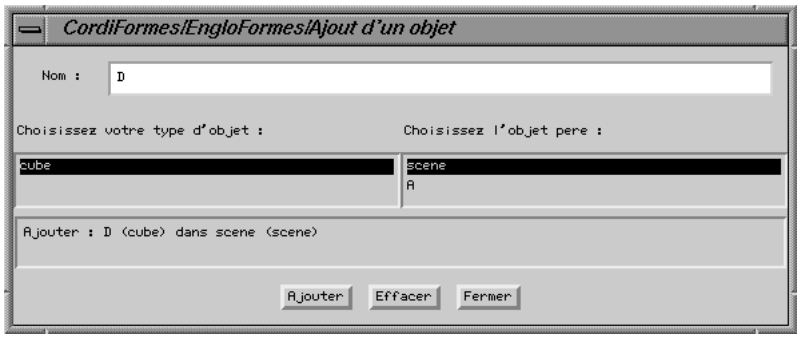

Figure 116. Exemple de fenêtre de gestion des objets

2.2.3 Gestion des propriétés

Le concepteur dispose d'un ensemble d'outils permettant de gérer la description, c'est-à-dire d'ajouter, de modifier ou de supprimer des propriétés de la description (Figure 117).

Figure 117. Dialogue de gestion des propriétés de la description

Pour ajouter une nouvelle propriété, le concepteur dispose d'un module d'analyse syntaxique permettant de récupérer une propriété à partir de son énoncé pour peu qu'il respecte les formes standard. Cependant, il peut aussi utiliser un ensemble de dialogues spécialisés pour construire les propriétés de façon interactive. Il dispose d'autant d'outils que de types de pro-

priétés recensées. Ces outils sont appelés en fonction de la forme de la propriété choisie (Figure 118).

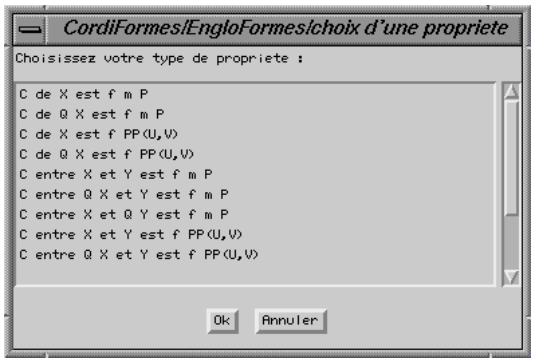

Figure 118. Choix du type de la nouvelle propriété

La saisie d'une propriété est fonction des éléments qui la composent. Il y a donc autant d'outils que de formes de propriétés. Par exemple, une propriété élémentaire est composée à partir d'un concept, d'un objet, d'un opérateur flou, d'un modificateur et d'une propriété de base. Nous retrouvons donc tous ces éléments dans le dialogue qui lui est affecté (Figure 119). Nous avons aussi la possibilité de choisir entre la forme affirmative et la forme négative.

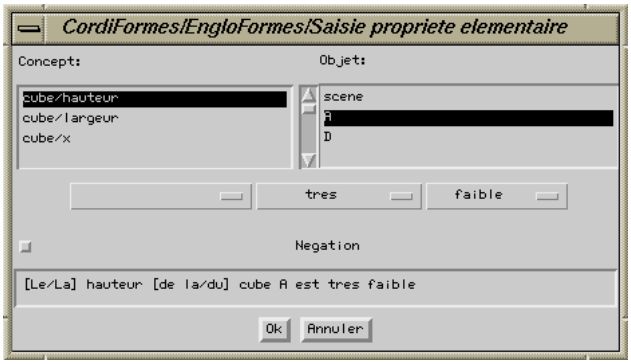

Figure 119. Construction d'une propriété élémentaire

Remarques :

- Les objets proposés par ces outils sont uniquement ceux déjà décrits, car l'utilisateur ne doit pas décrire des objets absents.
- La saisie des propriétés quantifiées est un peu différente de celle des propriétés non-quantifiées équivalentes. En effet, outre le choix du modificateur, l'objet sur lequel porte la quantification n'est pas connu. La liste des objets est alors remplacée par la liste des concepts présents dans la scène.
- Pour certains dialogues (comme celui de la saisie d'une propriété élémentaire), il serait intéressant de visualiser la propriété construite parmi les propriétés de base et, éventuellement, parmi les propriétés déjà décrites dans ce domaine. Or, dans le cas général,

ce n'est pas possible. En effet, les bornes du domaine n'étant pas fixées, il n'est pas possible de construire effectivement la propriété.

2.3. Gestion de la négation

En ce qui concerne la gestion de la négation, nous reprenons les éléments présentés dans le chapitre I.4. Le dialogue propose la liste ordonnée des propriétés plausibles correspondant à la négation (Figure 120). Éventuellement, l'utilisateur peut demander de simplifier les propriétés qui lui sont proposées. L'outil informe alors le module de description du choix de l'utilisateur. Il sagit d'une propriété équivalente ou simplement de la solution « tout sauf » (sans avis). Dans ce dialogue, pour aider le concepteur dans sa mise au point, il est possible d'afficher les fonctions d'appartenance de la propriété niée avec les propriétés plausibles ou de la propriété sélectionnée. Ceci est possible car ce dialogue est proposé uniquement lorsque le domaine est connu. En effet, nous le reverrons plus loin, le calcul des propriétés plausibles n'est effectué que lorsque l'on connaît le domaine.

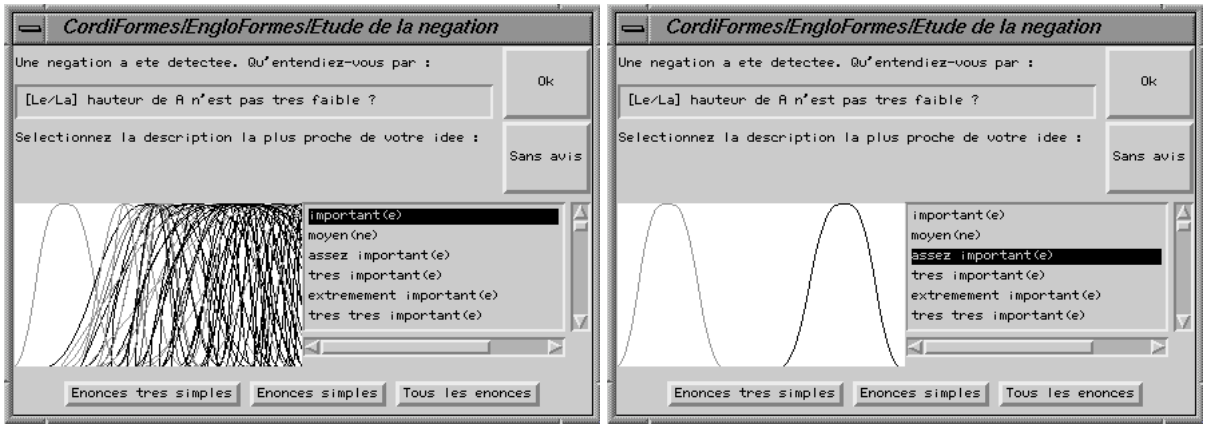

Figure 120. Fenêtres de gestion de la négation linguistique

2.4. Génération

Au niveau de la génération, il y a peu de relations avec l'utilisateur. Les seules informations qui s'échangent sont des ordres (« une autre solution », « passer au groupe suivant »…). Il n'y a donc qu'un seul élément d'interface possédant les boutons relatifs à ces ordres. Celui-ci peut être indépendant dans un petit dialogue ou directement placé dans l'application prototype (Figure 124). En dehors ces échanges et de la gestion de la négation, le module de génération a des contacts avec les outils d'interface uniquement s'il manque des informations. Par le biais des outils de prise de connaissance, ce module informe l'utilisateur du ou des problèmes. Ensuite, des outils de description sont exploités pour saisir les renseignements nécessaires. Ces outils sont alors spécifiques puisque l'information ne fait pas partie des informations habituelles de la description.

2.5. Prise de connaissance

Les outils de prise de connaissance présentent la scène en cours de génération et toutes les informations utiles pour bien la comprendre. Ils donnent aussi la possibilité à l'utilisateur d'intervenir sur la génération en lui permettant de valider tout ou une partie seulement des objets et en lui donnant des informations pour améliorer la génération. Notons qu'il est difficile de proposer des outils complexes de prise de connaissance. En effet, celle-ci dépend étroitement du domaine d'application du modeleur déclaratif. Les outils proposés par CordiFormes sont donc assez élémentaires. Le concepteur, s'il désire des outils plus performants, devra alors ajouter lui-même ses propres outils.

2.5.1 Présentation de la solution

La Figure 121 propose la liste des propriétés sous forme textuelle avec les mesures, les degrés d'appartenance et les différentes vues de la scène.

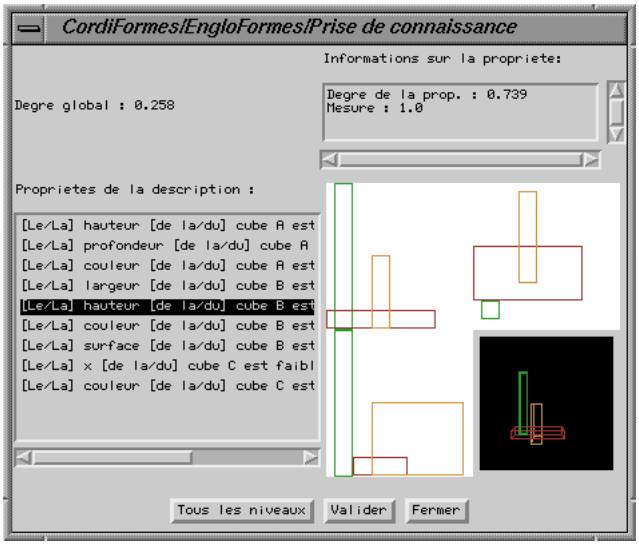

Figure 121. Fenêtre de prise de connaissance élémentaire

Un dialogue simple comme celui de la Figure 121 reste malgré tout assez souple. En effet, le concepteur doit simplement s'occuper de la visualisation de la scène (« Zone libre » dans le schéma de la Figure 122).

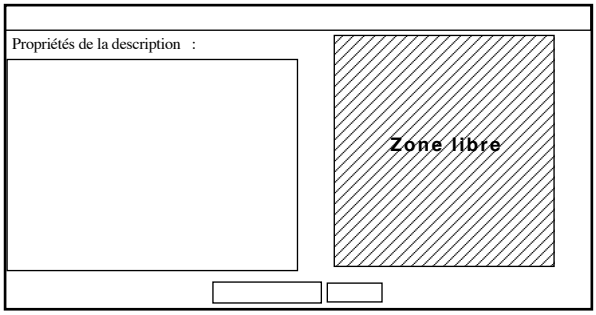

Figure 122. Dialogue élémentaire de prise de connaissance

Ce dialogue est assez représentatif de ce que l'on désire au niveau des outils de la couche interface. D'une part les outils de base doivent être disponibles et d'autre part l'utilisateur ne doit avoir à s'occuper que de ce qui est spécifique au domaine d'application du modeleur déclaratif qu'il veut construire. Notons que, pour l'application présentée, cette méthode de visualisation des boîtes englobantes est déjà présente dans les outils de la base de connaissance.

2.5.2 Validation de la solution

En phase de prise de connaissance, la présentation des résultats ne suffit pas. L'utilisateur doit pouvoir intervenir pour valider des objets, en refuser d'autres, critiquer la solution ou modifier la description. Les outils de prise de connaissance proposent alors des éléments de validation de la scène où l'utilisateur peut indiquer les objets qu'il désire garder et, éventuellement, ceux qu'il préfère explorer en premier. De plus, l'utilisateur peut avoir accès depuis ces outils à ceux du module de description. Ceci lui permet alors d'énoncer des propriétés de modification sur les concepts des objets qu'il trouve « à peu près » corrects à un ou deux détails près.

2.6. Relation entre modules et outils de dialogue

Les outils de la couche interface sont très étroitement liés aux modules du noyau. Ceux de description travaillent principalement avec le module de description. Ils sont aussi parfois utilisés par le module de génération et par certains outils de prise de connaissance. Il existe très peu d'outils liés à la génération. Ces derniers concernent uniquement les ordres de progression dans l'espace des solutions. En effet, il s'agit d'une phase de calcul dont l'utilisateur n'a que très peu conscience, si ce n'est pas du tout si on considère que les ordres de progression font partie de la prise de connaissance ([CDMM97b]). Les outils de prise de connaissance sont, quant à eux, étroitement associés au module de prise de connaissance. Ils sont aussi utilisés par le module de génération, principalement pour gérer tout manque d'informations. Pour organiser la présentation de ces outils ainsi que les communications avec les modules du noyau, il faut intégrer les outils de MultiVues ([Paj94b] et [Luc97]). Ceux-ci sont particulièrement utiles pour la prise de connaissance. Ils permettent de visualiser plusieurs solutions simultanément et, plus généralement, de prendre connaissance efficacement d'un grand nombre de solutions. Avec ces outils, le concepteur peut décrire de manière déclarative la disposition des dialogues dans l'écran. Comme pour les éléments du noyau, les éléments de dialogues, ainsi que les dialogues eux-mêmes, proposés dans cette couche peuvent être modifiés et enrichis par le concepteur. Tout ceci est en effet fonction des problèmes spécifiques liés au domaine d'application et aux habitudes des utilisateurs.

Les relations entre les outils d'interface et les modules du noyau que nous venons de décrire sont reprises par la Figure 123.

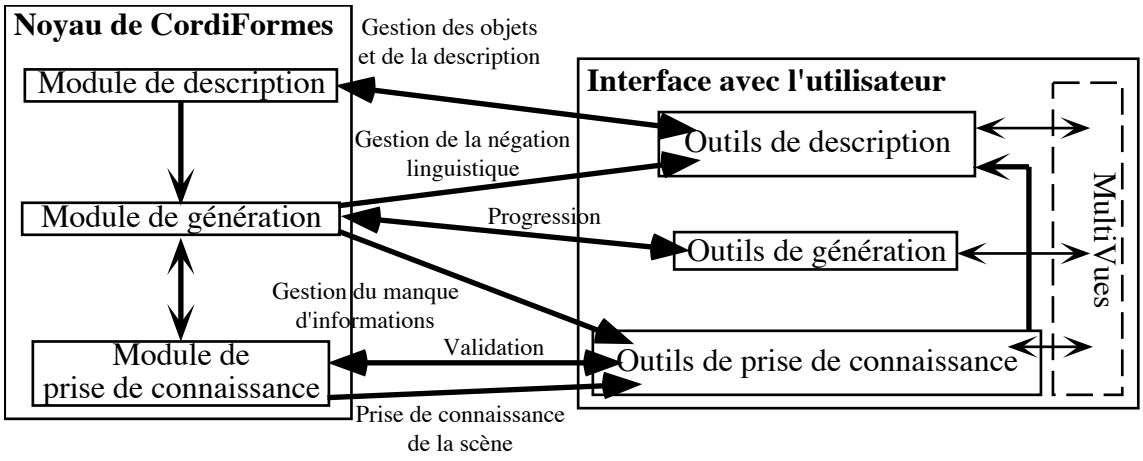

Figure 123. Relations entre modules et outils d'interface

3. Développer une maquette rapidement

Nous arrivons désormais à la dernière couche de CordiFormes : la couche application prototype. L'idée est assez simple : une fois tout le système de concepts, de tâches et de contraintes mis en place, le concepteur peut alors lancer une application (Figure 124) où il peut :

- créer des objets ;
- faire une description (conjonction de propriétés de différents types) ;
- lancer les calculs ;
- prendre connaissance des scènes produites.

Figure 124. Application prototype avec outils de gestion de la génération

Cette application utilise les outils les plus génériques de la couche interface. Cependant, le concepteur doit malgré tout s'occuper du problème de la prise de connaissance des solutions. En effet, il n'est pas possible de prévoir la méthode associée aux objets produits. Pour cela, l'application utilisera un dialogue de prise de connaissance comme celui présenté à la Figure 121. L'utilisateur ne s'occupe alors que de la visualisation.

Remarques :

- A défaut, les solutions pourraient être proposées sous forme textuelle en indiquant les valeurs des différents concepts des différents objets de la scène. Cependant, cette solution n'est plus envisageable dès que le nombre de concepts manipulés devient relativement important.

- D'un point de vue plus général, cette application ne se veut pas une application pour l'utilisateur. Elle est destinée au concepteur. L'objectif est de lui permettre d'évaluer son modeleur. Par conséquent, il n'est pas nécessaire que les outils de description et de prise de connaissance fournis soient très puissants. La priorité doit plutôt être donnée à leur aspect générique.

Les dialogues proposés par défaut ne sont, en effet, pas très sophistiqués mais suffisent pour l'évaluation de l'application. Cependant, ces dialogues peuvent être remplacés ou modifiés par le concepteur.

4. Conclusion

Outre le noyau, la plate-forme CordiFormes possède deux couches liées à l'interface avec l'utilisateur. La couche interface propose trois ensembles d'outils concernant la description, la génération et la prise de connaissance. Ces outils servent de base aux dialogues nécessaires pour utiliser un modeleur déclaratif de manière interactive. La couche prototype propose, quant à elle, une application « clef en main ».

Les outils que nous venons de présenter dans ce chapitre sont assez simples et peu nombreux. Il apparaît alors indispensable de les enrichir. Il faut d'une part les diversifier, et d'autre part, proposer des outils de plus haut niveau. Nous allons maintenant présenter trois exemples d'application contruites à partir de CordiFormes.

CHAPITRE II.6 : EXEMPLES D'APPLICATIONS AVEC CORDIFORMES

1. Introduction

Dans ce chapitre, nous allons présenter quelques applications développées à l'aide de CordiFormes. Ces applications sont simples mais elles ont permis de valider la démarche et d'illustrer la présentation de CordiFormes. Nous distinguerons les projets suivants :

- EngloFormes, modélisation déclarative de boîtes englobantes de bâtiments (section 2) ;
- LinéaFormes, modélisation déclarative de segments dans un plan (section 3) ;
- ChromoFormes, modélisation déclarative de couleurs (section 4).

Le schéma de présentation des projets sera le suivant :

- Description générale du projet ;
- Utilisation de l'application (ce que l'utilisateur peut de décrire et ce qu'il obtient) ;
- Implémentation de l'application ;
- Propriétés utilisables et exemple de description.
- Apport de ce projet pour la mise en valeur des éléments de la plate-forme.

2. EngloFormes

2.1. Présentation du projet

Le projet *EngloFormes* consiste à disposer de manière déclarative des boîtes englobantes de bâtiments ou de parties de bâtiments ([DeM97b]). Il constitue un cas particulier du projet VoluFormes (C*f*. [Chau92] et [Chau94b]). Dans ce dernier les boîtes sont définies dans l'espace à trois dimensions. En particulier, leur position n'est pas contrainte. Par contre, dans le projet EngloFormes, les boîtes sont « posées » sur un plan représentant un sol. Autrement dit, leur position est telle que l'une des faces se trouve systématiquement dans le plan horizontal.

Remarques :

- Les dimensions des objets sont souvent données par rapport à leur boîte englobante appelée aussi *Boîte de Manhattan*. Les bords de la boîte englobante sont parallèles aux axes du repère propre. Ce type de boîte englobante est appelé par [Mac92] *Boîte de Manhattan positionnée* (et aussi boîte orthogonale, orthomorphe ou orthotopique). Par extension, il définit les *Objets de Manhattan* qui sont des solides dont chaque partie

(face, arête, axe...) est parallèle ou perpendiculaire aux axes du repère absolu. Il montre que tout objet connexe de Manhattan peut se décomposer en plusieurs boîtes adjacentes de Manhattan. Par conséquent, l'étude des Boîtes de Manhattan suffit.

- [Mou94] montre qu'il n'est pas toujours facile de définir la boîte englobante d'un objet. Le problème est particulièrement délicat lorsqu'il faut déterminer la boîte englobante d'un objet dont une ou plusieurs dimensions sont infinies ou nulles (plan, quadriques, quartiques...). Ces dimensions extrêmes posent aussi des problèmes quand il faut faire des opérations booléennes (union, intersection, différence...) sur ces boîtes.

2.2. Utilisation d'EngloFormes

Dans ce projet, l'utilisateur décrit des boîtes englobantes de bâtiments dans un univers dont les dimensions sont fixées a priori. Il peut ainsi décrire les dimensions de cette boîte et sa position dans le plan horizontal. La position en hauteur est calculée pour que la boîte soit posée sur le plan inférieur de l'univers.

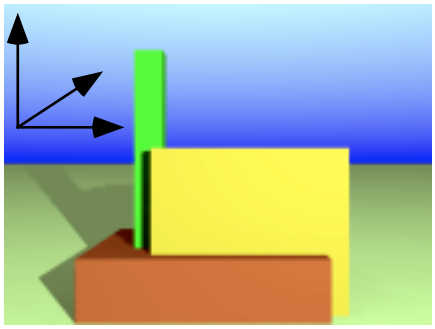

Figure 125. Description de boites englobantes

2.3. Implémentation à l'aide de CordiFormes

La boîte englobante est donc définie à partir du concept de boîte en trois dimensions présente dans la bibliothèque. La définition est modifiée afin de la « poser » sur le sol (le concept de position en « y » n'est plus générateur mais est calculé à partir de la hauteur). De plus, on ajoute un nouveau concept, non-générateur, pour la description de la surface au sol de la boîte. Il est spécifié aussi que le cube est généré par énumération à l'exception de la hauteur qui sera produite par tirage aléatoire.

Algorithme 11. Définition d'un cube (boîte englobate)

```
import PlateForme.Concept.CConceptTerminal;
import PlateForme.Concept.CConcept;
import PlateForme.Domaine.CDomaine;
import PlateForme.Divers.CIntervalleClassique;
import PlateForme.Contrainte.CContrainteMesure;
import PlateForme.Contrainte.CContrainte;
import PlateForme.Tache.Tache;
import PlateForme.Bibliotheque.CConceptBoite3D;
```

```
//Definition de la mesure pour le calcul de la surface d'une boite
final class CMesureSurface extends CContrainteMesure {
 public CMesureSurface(CConcept leConcept) { super(leConcept);}
 protected CIntervalleClassique CalculerDomaine() {
   CDomaine d1 = Domaine[0];
   CDomaine d2 = Domaine[1];
   CIntervalleClassique i = new CIntervalleClassique(
                     d1.Gauche()*d2.Gauche(), d1.Droite()*d2.Droite());
   return i;
 }

 protected Object Calculer() {
   float l = ((Float)ValeurSource).floatValue();
   float p = ((Float)ValeurSourceBis).floatValue();
   return new Float(l*p);
 }
}

//Definition du calcul de l'altitude pour que la boite soit posee
final class CMesureYY extends CContrainteMesure {
 public CMesureYY(CConcept leConcept) { super(leConcept);}
 protected Object Calculer() {
   float h = ((Float)ValeurSource).floatValue();
   return new Float(h/2+conceptCible.leDomaineDeTravail.a);
 }
}

public final class cube extends CConceptBoite3D {
 public cube() {
   super(8,1);
   Nom = "cube";
   //Pour ici, YY (position en hauteur) n'est pas a utiliser
   ContrainteYY.Desactiver();
   UnsetGenerateur(YY);

   //Determination du mode de generation du cube
   maTacheGeneratrice = Tache.ENUMERATION;
   Hauteur.maTacheGeneratrice = Tache.TERMINALE_ALEATOIRE;

   //Installation de la mesure de hauteur
   CMesureYY MesureYY = new CMesureYY(YY);
   MesureYY.AjouterReference(Hauteur);
   YY.AjouterContrainteMesure(MesureYY);
   MesureYY.Activer();

   // Exemple de domaine non generateur : mesure de la surface
   CConceptTerminal Surface = new CConceptTerminal(this,"surface",
                                          new CDomaine(1,8*8,1f),null);
   AjouterConcept(Surface,7,false);

   CMesureSurface MesureSurface = new CMesureSurface(Surface);
   MesureSurface.AjouterReference(Largeur);
   MesureSurface.AjouterReference(Profondeur);
   Surface.AjouterContrainteMesure(MesureSurface);
 }}
```

L'univers est une boîte en trois dimensions dont le centre est placé à l'origine du repère absolu et les dimensions sont fixées par le concepteur.

Algorithme 12. Définition de l'univers pour EngloFormes

```
import PlateForme.Bibliotheque.CConceptBoite3D;

public final class univers extends CConceptBoite3D {
 public univers() {
    super(8,0);
    Nom = "scene";
    InitialiserTout();
    Hauteur.SetMesure(new Float(8));
    XX.SetMesure(new Float(0));
    Largeur.SetMesure(new Float(8));
    YY.SetMesure(new Float(0));
    Profondeur.SetMesure(new Float(8));
    ZZ.SetMesure(new Float(0));
    Couleur.SetMesure("Gris");
    SetMesure();
 }
}
```

Finalement, l'application consiste simplement à fournir le concept lié à la scène ainsi que les concepts liés aux objets et aux caractéristiques de la scène.

Comme la prise de connaissance est particulière, le concepteur propose son propre dialogue, issu du dialogue standard dont il a spécifié la zone libre en proposant une vue 3D et trois vues selon les axes principaux « x », « y » et « z » (Figure 126).

Lorsqu'une scène est validée, l'utilisateur peut enregistrer sa solution dans un format spécifique afin de créer une image plus réaliste. Nous avons choisi ici de produire une fichier reconnaissable par un programme de visualisation réaliste par lancer de rayon (POV-Ray).

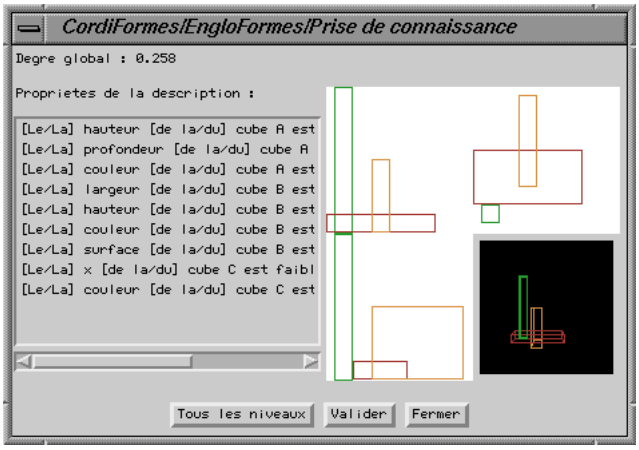

Figure 126. Dialogue de prise de connaissance dans EngloFormes

Normalement, l'application peut en rester là, car l'application standard possède des dialogues de description. Cependant, il est possible que le concepteur ait envie de proposer une description par défaut (pratique pour la mise au point en particulier). Nous proposons donc ici une description par défaut qui nous permet de mettre en évidence certaines caractéristiques de CordiFormes. Le concepteur propose trois objets nommés respectivement "A", "B" et "C". Ce dernier objet à la particularité d'être produit par tirage aléatoire pour tous ses concepts composants. Le concepteur propose aussi une description formée de propriétés élémentaires sur chacun de ces objets. Pour cela, il utilise des méthodes proposées par le module de description pour créer les propriétés élémentaires (par la simple donnée de ses éléments constitutifs que sont le concept, l'objet visé, affirmation ou négation, l'opérateur flou, le modificateur et la propriété de base) et ajouter une propriété à la description.

Algorithme 13. L'application EngloFormes

```
import PlateForme.Interfaces.IModeleurDeclaratif;
import PlateForme.Propriete.CProprieteElementaire;
import PlateForme.Propriete.CPropriete;
import PlateForme.Contrainte.CContrainte;
import PlateForme.Tache.Tache;
import cube;

import java.awt.*;

public class EngloFormes extends IModeleurDeclaratif {
  // Le dialogue de prise de connaissance
  PdC_EngloFormes DPdC;

public EngloFormes() {
  super("EngloFormes");

  //Concepts de l'application
  MD.ModuleDescription.SetScene(new univers());
  MD.ModuleDescription.AddConcept("cube");

  //Mise en place du dialoge de prise de connaissance specifique
  SetDialPdC(new PdC_EngloFormes(this,MD.ModulePdC));

 //Exemple de description par defaut
  CProprieteElementaire pp;

  MD.ModuleDescription.AddObject("A", "cube", "scene");
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/hauteur","A",
                              false,"","tres tres","important(e)");
  MD.ModuleDescription.AjouterPropriete(pp);
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/profondeur",
                              "A", false,"","extremement","faible");
  MD.ModuleDescription.AjouterPropriete(pp);
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/couleur","A",
                              false,"", "de", "VertSombre",null);
  MD.ModuleDescription.AjouterPropriete(pp);

  MD.ModuleDescription.AddObject("B", new cube(), "scene");
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/largeur","B",
```

```
                                false,"","tres","important(e)");
  MD.ModuleDescription.AjouterPropriete(pp);
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/hauteur","B",
                                false,"plus ou moins","tres","faible");
  MD.ModuleDescription.AjouterPropriete(pp);
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/couleur","B",
                                false,"", "de", "RougeBrique",null);
  MD.ModuleDescription.AjouterPropriete(pp);
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/surface","B",
                                false,"","","faible");
  MD.ModuleDescription.AjouterPropriete(pp);

  cube cube = new cube();
  cube.maTacheGeneratrice = Tache.ALEATOIRE;
  MD.ModuleDescription.AddObject("C", cube, "scene");
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/x","C",
                                false,"","","faible");
  MD.ModuleDescription.AjouterPropriete(pp);
  pp = MD.ModuleDescription.CreerProprieteElementaire("cube/couleur","C",
                                false,"", "de", "Or",null);
  MD.ModuleDescription.AjouterPropriete(pp);

  Init(); pack(); show();
}

public static void main(String[] argv) {me = new EngloFormes();}
}
```

2.4. Les propriétés et les concepts dans EngloFormes

Ainsi construit, EngloFormes permet de décrire des cubes bien spécifiques. Les concepts et les propriétés de base disponibles sont donnés par le Tableau 14.

Tableau 14. Concepts pour un cube dans EngloFormes

Concept	Propriétés
Hauteur	bas(se) ; moyen(ne) ; haut(e)
Largeur	étroit(e) ; moyen(ne) ; large
Profondeur	faible ; moyen(ne) ; profond(e)
x	faible ; moyen(ne) ; important(e)
y	faible ; moyen(ne) ; important(e)
z	faible ; moyen(ne) ; important(e)
Couleur	C*f*. ChromoFormes (mode Liste)
Surface	faible ; moyen(ne) ; important(e)

L'utilisateur peut alors fournir une description comme « *Il y a trois cubes. Soit le cube A. A est vert sombre. A est grand. A n'est pas profond. Soit le cube B. B est rouge brique. La largeur de B est très importante. B n'est pas grand. Soit le cube C. C est or. C est derrière A. Le centre de C est à droite. C est plus grand que B.* » Les scènes de la Figure 127 sont des solutions possibles à cette description.

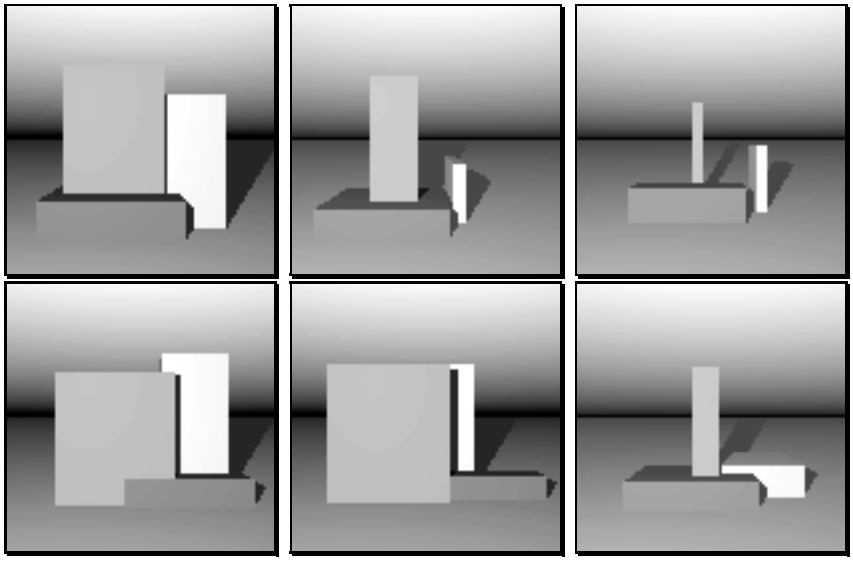

Figure 127. Solutions possibles à une description dans EngloFormes

2.5. Spécificités de ce projet

Ce projet nous a permis de vérifier un certain nombre d'éléments de CordiFormes. En particulier, nous avons utilisé au maximum l'application prototype et les éléments du noyau. En conséquence, notons que le nombre de lignes spécifiques à cette application est vraiment faible. Le plus coûteux en programmation a été l'implémentation des éléments de prise de connaissance spécifiques (moins de 250 lignes). Cette économie est principalement liée à l'utilisation de la bibliothèque de CordiFormes qui propose un concept de boîte englobante ainsi qu'un ensemble d'outils de visualisation en 3D. Nous avons aussi pu mettre en pratique le panachage des modes de génération entre objets ("A" et "B" sont énumérés alors que "C" est généré par tirage aléatoire) et entre concepts d'un même objet (le concept de "Hauteur" est produit par tirage aléatoire alors que les autres concepts le sont par énumération).

3. LinéaFormes

3.1. Présentation du projet

Comme nous l'avons déjà vu dans la partie I, dans le projet FiloFormes ([Paj94]), les scènes constituant l'univers sont des *configurations de segments de droite dans un plan*. Ce plan est représenté par une grille n×m. Les segments sont disposés dans cette grille. Un des objectifs de ce projet est de produire des jeux d'essai raisonnés pour des algorithmes de visualisation en synthèse d'images afin de contrôler les évaluations et mesurer l'impact des différentes améliorations que l'on peut mettre en œuvre [Luc94]. Une scène composée de segments dans un plan peut être caractérisée par : le nombre de segments qu'elle comporte, leur longueur, leur pente, la longueur de la configuration ou des composantes verticales et horizontales, le nombre d'intersections, la densité, le degré moyen de recouvrement…

Nous nous proposons de reprendre une grande partie de ce projet en utilisant CordiFormes. Le projet *LinéaFormes* consiste donc à décrire un ensemble de segments du plan. Nous allons nous intéresser surtout aux caractéristiques des segments de la scène, c'est-à-dire leur position, leur longueur et leur pente.

3.2. Utilisation de LinéaFormes

La construction d'une scène se déroule en plusieurs phases en fonction de la constitution de la scène. La méthode de conception est celle par défaut, c'est-à-dire par niveau de la hiérarchie de la scène. Une scène est composée de zones dans lesquelles se trouvent des segments (graphes sémantiques). Par conséquent, l'utilisateur choisit d'abord la zone (place et forme) puis la disposition des segments et leurs caractéristiques.

3.3. Implémentation à l'aide de CordiFormes

Le segment 2D (comme le segment 3D) fait partie de la base de connaissances. Le concepteur n'a donc rien à faire. De même, une zone n'est en réalité qu'une boîte englobante 2D comme la scène dont les dimensions sont fixées par le concepteur.

Remarque : Le segment et la zone n'ont pas forcément besoin d'être redéfinis car ils sont utilisés sans modification par l'application. La seule qu'on ait faite consiste à changer le nom afin que la description soit plus naturelle.

Algorithme 14. Définition d'un segment

```
import PlateForme.Bibliotheque.CConceptSegment2D;
import PlateForme.Tache.Tache;
public final class CSegment extends CConceptSegment2D {
 public CSegment() {
    super(8,0);
    Nom = "segment";
}}
```

Algorithme 15. Définition d'une zone

```
import PlateForme.Tache.Tache;
import PlateForme.Bibliotheque.CConceptBoite2D;
public final class CZone extends CConceptBoite2D {
 public CZone() {
    super(8,0);
    Nom = "zone";
}}
```

Algorithme 16. Définition de l'univers pour LinéaFormes

```
import PlateForme.Bibliotheque.CConceptBoite2D;

public final class CUniversSegment extends CConceptBoite2D {
 public CUniversSegment() {
```

```
    super(8,0);
    Nom = "scene";
    InitialiserTout();
    Hauteur.SetMesure(new Float(8));
    XX.SetMesure(new Float(0));
    YY.SetMesure(new Float(0));
    Largeur.SetMesure(new Float(8));
    Couleur.SetMesure("gris");
    SetMesure();
}}
```

Le dialogue de prise de connaissance est standard. Le concepteur ne propose qu'une méthode de visualisation de la scène (rectangles et segments). Ce dialogue sert aussi bien pour le choix des zones que pour le choix des configurations de segments (Figure 128).

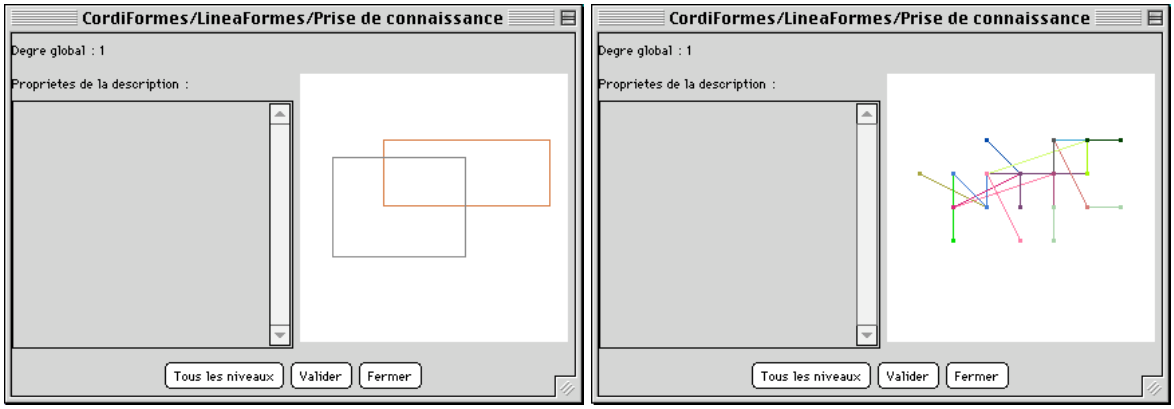

Figure 128. Dialogues de prise de connaissance de LinéaFormes lors du choix des zones puis du choix des configurations de segments

Le dialogue standard propose aussi de visualiser tous les niveaux ensembles (Figure 129).

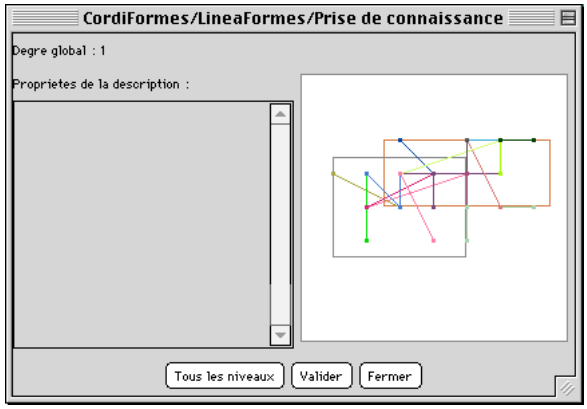

Figure 129. Dialogues de prise de connaissance de LinéaFormes

Finalement, l'application consiste à définir les concepts utilisés, le graphe sémantique spécifiant l'organisation de la scène et les contraintes qui sont associées (segment dans sa zone…).

Algorithme 17. L'application LinéaFormes

```
import PlateForme.Interfaces.IModeleurDeclaratif;
import PlateForme.Propriete.CProprieteElementaire;
import PlateForme.Propriete.CPropriete;
import CZone;
import CSegment;
import java.awt.*;

public class LineaFormes extends IModeleurDeclaratif {
public LineaFormes() {
  super("LineaFormes");
  //Concepts de l'application
  MD.ModuleDescription.SetScene(new CUniversSegment());
  MD.ModuleDescription.AddConcept("CZone");
  MD.ModuleDescription.AddConcept("CSegment");
  CRelationSemantique rs;
  CGrapheSemantique gs = new CGrapheSemantique();
  rs = new CRelationSemantique("CZone",
                          CRelationSemantique.EST_COMPOSE_DE,"CSegment");
  gs.AddRelation(rs);
  rs = new CRelationSemantique("CUniversSegment",
                          CRelationSemantique.EST_COMPOSE_DE,"CZone");
  gs.AddRelation(rs);
  gs.AddContrainte("CContrainteSegZone",
                            "segment/origine_x","zone/largeur|zone/x");
  gs.AddContrainte("CContrainteSegZone",
                            "segment/origine_y","zone/hauteur|zone/y");
  gs.AddContrainte("CContrainteSegZone",
                            "segment/extremite_x","zone/largeur|zone/x");
  gs.AddContrainte("CContrainteSegZone",
                            "segment/extremite_y","zone/hauteur|zone/y");

  //Mise en place du dialoge de prise de connaissance specifique
  SetDialPdC(new LineaFormes_PdC(this, MD.ModulePdC));
//Exemple de description d'un ensemble de 20 segments
//repartis sur deux zones
  MD.ModuleDescription.AddObject("Zone1", new CZone(), "scene");
  MD.ModuleDescription.AddObject("Zone2", new CZone(), "scene");
  int i;
  String Nom;
  for(i=0;i<10;i++) {
    Nom = "S"+i;
    MD.ModuleDescription.AddObject(Nom, new CSegment(), "Zone1");
  }
  for(i=10;i<20;i++) {
    Nom = "S"+i;
    MD.ModuleDescription.AddObject(Nom, new CSegment(), "Zone2");
  }

  Init(); pack(); show();
}

public static void main(String[] argv) { me = new LineaFormes();}
}
```

3.4. Les propriétés et les concepts dans LinéaFormes

Ainsi défini, LinéaFormes possède donc deux concepts représantant des objets de la scène : la zone et le segment. L'utilisateur peut donc décrire des propriétés se rapportant aux différents concepts composants de ces deux objets. Le Tableau 15 et le Tableau 16 proposent ces concepts et les propriétés de base associées.

Tableau 15. Concepts pour une zone dans LinéaFormes

Concept	Propriétés
Hauteur	bas(se) ; moyen(ne) ; haut(e)
Largueur	étroit(e) ; moyen(ne) ; large
x	faible ; moyen(ne) ; important(e)
y	faible ; moyen(ne) ; important(e)
Couleur	vert sombre ; rouge brique ; or

Tableau 16. Concepts pour un segment dans LinéaFormes

Concept	Propriétés
Origine x	faible ; moyen(ne) ; important(e)
Origine y	faible ; moyen(ne) ; important(e)
Extrémité x	faible ; moyen(ne) ; important(e)
Extrémité y	faible ; moyen(ne) ; important(e)
Pente	faible ; moyen(ne) ; important(e)
Longueur	court(e) ; moyen(ne) ; long(ue)
Couleur	C*f*. LinéaFormes (en mode RVB)

L'utilisateur peut alors donner la description suivante : « *Le segment S1 est très long. Au moins 6 segments ont une longueur assez peu importante* ». La Figure 130 propose une solution à cette description.

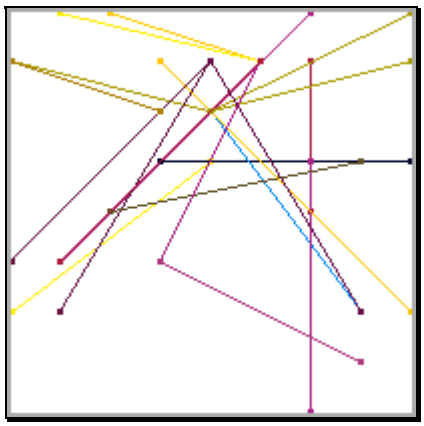

Figure 130. Solutions possibles à une description dans LinéaFormes

Par rapport à ce que nous avons présenté ici, des descriptions comme « *Le nombre de verticales est faible. La longueur de recouvrement est très importante. La proportion de segments parallèles est vraiment faible. Le degré moyen de recouvrement est plus ou moins assez important* » (Figure 131a) ou « *Il y a vraiment peu de segments verticaux et très peu de segments horizontaux. Le recouvrement est assez long. Le degré moyen de recouvrement n'est*

pas extrêmement important. Il n'y a pas beaucoup de parallèles » (Figure 131b et c) vues dans la première partie, ne sont pas possibles. Il faut ajouter pour cela les concepts « longueur de recouvrement », « degré moyen de recouvrement » et la relation « parallèles à ».

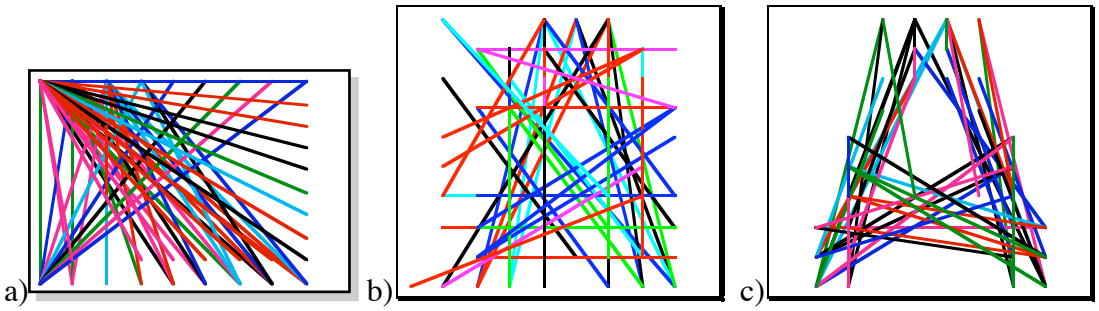

Figure 131. Autres solutions possibles à une description dans LinéaFormes

3.5. Spécificités du projet

Ce projet est intéressant, car il permet de travailler la gestion des propriétés quantifiées. En effet, une application de la génération de configurations de segments consiste à valider des algorithmes de visualisation de scènes par balayage par des plans (méthodes issues de celle de Watkins). L'intérêt est donc d'avoir des configurations dont les caractéristiques sont basées sur celles des segments (longueur, pente, position…). Ce projet permet aussi de proposer une hiérarchie de scène qui n'est pas seulement à deux niveaux. Nous avons donc eu besoin d'utiliser les graphes sémantiques pour gérer les éléments et montré qu'il est donc possible d'appliquer des méthodes de conception différentes sur des objets qui ne sont pas totalement triviaux.

4. ChromoFormes

4.1. Présentation du projet

Habituellement, la détermination d'une couleur n'est pas toujours facile ([Hic81], [Leg90], [Cou92]). L'utilisateur est souvent obligé de donner une série de valeurs pour des paramètres dont il ne contrôle pas vraiment le comportement (différents modèles RVB, TSL…). Il peut rechercher aussi son bonheur dans de grandes tables (couleurs Pantone et autres nuanciers). Actuellement, son choix est facilité par des techniques graphiques permettant d'évoluer dans les modèles. Cependant, l'exploration n'est pas toujours facile ou naturelle. L'objectif du projet *ChromoFormes* est donc de proposer un outil (ici sous forme d'application indépendante) permettant de déterminer une couleur de façon déclarative.

Les modèles de représentation des couleurs sont très nombreux ([Hic81], [Leg90], [Cou92]). Nous avons par exemple les modèles :

- Rouge-Vert-Bleu (RVB), modèle utilisé par les moniteurs couleurs (TV ou écrans d'ordinateurs), il définit les couleurs par l'addition de trois couleurs primaires (rouge, vert et bleu) selon la synthèse additive (reconstitution de la couleur par combinaison des radiations de lumières rouge, verte et bleue telles que R+V+B=blanc) ;
- Cyan-Magenta-Jaune (CMJ) ou Cyan-Magenta-Jaune-Noir (CMJN), modèle utilisé par les systèmes d'impression, il définit les couleurs selon la synthèse soustractive (reconstruction de la couleur par superposition de couches de colorants cyan, magenta et jaune telle que C+M+J=noir) ;
- YIQ, une variante du modèle RVB ;
- Teinte-Saturation -Luminance (TSL), modèle particulier, il est issu des travaux du peintre Albert H. Munsell ;
- …

4.2. Utilisation de ChromoFormes

L'utilisateur décrit alors ce qu'il désire et c'est au modeleur d'explorer l'espace des couleurs selon le modèle approprié pour déterminer celles susceptibles d'être intéressantes.

4.3. Implémentation à l'aide de CordiFormes

Le modèle « le plus intuitif et le plus proche de la perception naturelle des couleurs » ([Cou92]) est incontestablement le modèle Teinte-Saturation-Luminance ou TSL (Figure 132).

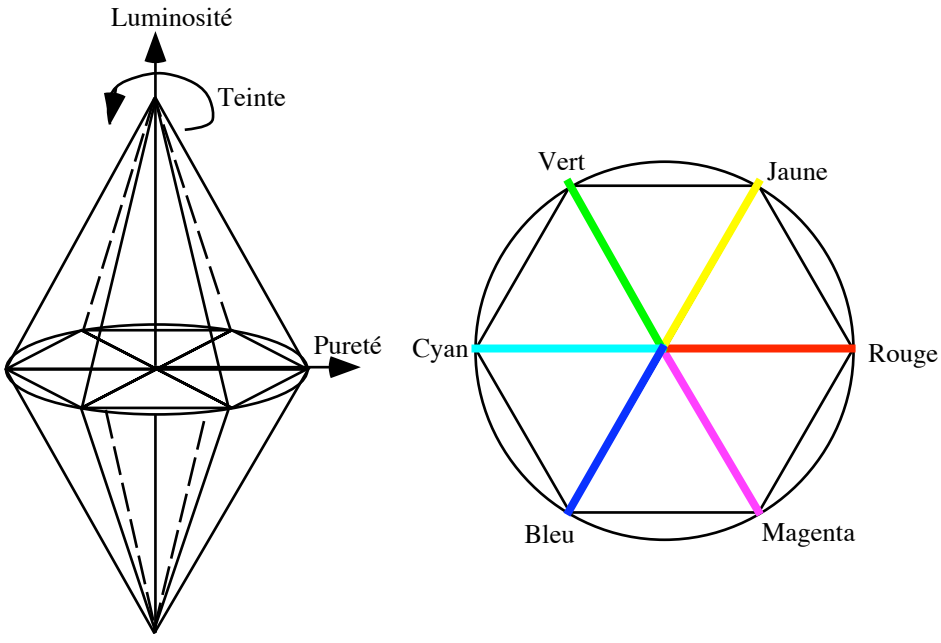

Figure 132. Modèle TSL

La *teinte* détermine la couleur souhaitée. La *luminance* définit la part du noir ou du blanc dans la couleur sélectionnée. Elle permet de distinguer une couleur claire d'une couleur sombre. La *saturation* mesure la pureté des couleurs, c'est-à-dire le pourcentage de couleur pure par rapport au blanc. Elle permet de distinguer les couleurs « vives » des couleurs « pastel » ou « délavées ». Liés aux couleurs, il existe un grand nombre de termes correspondant à des couleurs précises. Le concept couleur possède donc une liste de couleurs « nommées » permettant d'atteindre directement une couleur en fonction de son nom commun (« chocolat », « bleu azur », « or », « rouge brique », « vert sombre »…).

Ce modèle peut être défini par un espace en forme de double cône (Figure 132) dont l'axe représente la luminosité ou l'intensité de la couleur (sombre vers le bas et claire vers le haut). Elle est souvent donnée en pourcentage comme « totalement sombre »=0% et « totalement claire »=100%. La base commune des cônes représente les différentes teintes. Elles sont habituellement repérées sur ce cercle de base par une valeur angulaire (0° à 360°). Les teintes fondamentales sont disposées selon un hexagone régulier comme le rouge=0°, le jaune=60°, le vert=120°, le cyan=180°, le bleu=240° et le magenta=300°. Le degré de pureté de la couleur est représenté par la distance par rapport à l'axe vertical du cône. Elle est souvent donnée en pourcentage telle que « saturation maximale »=100% (surface du cône) et « saturation minimale »=0% (axe du cône)

Une couleur sera donc composée de trois concepts :

- la luminosité avec les trois propriétés de base classiques renommées {« sombre », « moyenne », « claire »} ;
- la pureté avec la propriété de base unique classique renommée {« pure »} ou les trois propriétés de base classiques renommées {« pastel », « moyenne », « vive »} ;
- la teinte qui comporte les six propriétés de base {« rouge », « jaune », « vert », « cyan », « bleu », « magenta »} (Figure 133).

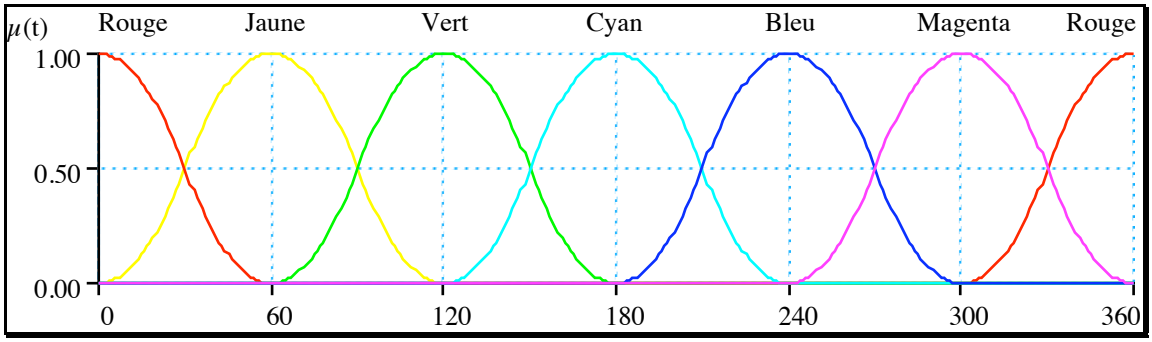

Figure 133. Définition des teintes dans le modèle TSL

Remarque : La définition des fonctions d'appartenance pour le concept « teinte » est intuitive. Elle n'est basée sur aucun principe et sera peut-être à affiner.

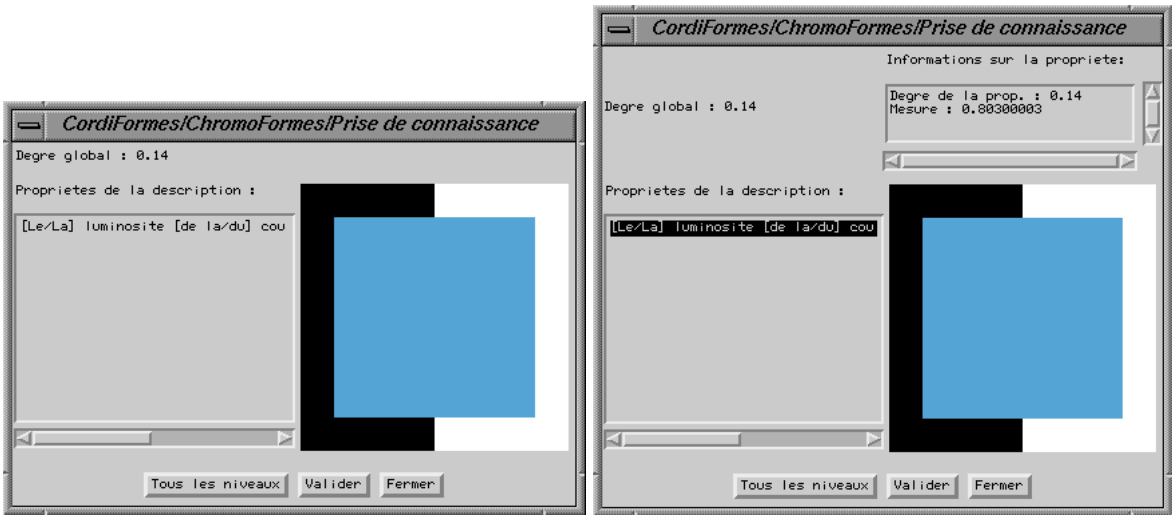

Figure 134. Dialogue de prise de connaissance dans ChromoFormes

D'un point de vue pratique, aussi bien pour la visualisation que pour la construction ou même parfois pour la description, nous pouvons ajouter les concepts « rouge », « vert », « bleu » pour le modèle RVB (visualisation) et les concepts « jaune », « cyan », « magenta », « noir » pour le modèle CMJN (impression). Tous ces concepts comportent les trois propriétés de base standard.

D'un point de vue de définition des concepts, le concepteur n'a, dans ce cas, aucun travail à faire car le concept de « couleur » fait bien évidemment partie de l'ontologie en synthèse d'image. L'objectif visé est ici de définir une seule couleur. Par conséquent, le concept univers est vide. Une autre solution est de considérer la couleur comme l'univers.

Algorithme 18. L'application ChromoFormes

```
import PlateForme.Interfaces.IModeleurDeclaratif;
import PlateForme.Propriete.CProprieteElementaire;
import PlateForme.Bibliotheque.CConceptCouleur;

public class ChromoFormes extends IModeleurDeclaratif {
  // Le dialogue de prise de connaissance
  PdC_ChromoFormes DPdC;

public ChromoFormes() {
  super("ChromoFormes");

  //Concepts de l'application
  //1 seul objet possible
  MD.ModuleDescription.SetScene(new CConceptCouleur());


  //Mise en place du dialoge de prise de connaissance specifique
  SetDialPdC(new PdC_ChromoFormes(this,MD.ModulePdC));

 //Exemple de description par defaut
  CProprieteElementaire pp;
```

```
  pp = MD.ModuleDescription.CreerProprieteElementaire(
          "couleur/luminosite","A", false,"","tres tres","important(e)");
   MD.ModuleDescription.AjouterPropriete(pp);

   Init(); pack(); show();
}

public static void main(String[] argv) {me = new ChromoFormes();}
}
```

Finalement, dans cette application, le concept essentiel est le concept de couleur de la bibliothèque (décrit dans l'annexe 4).

4.4. Les propriétés et les concepts dans ChromoFormes

Les concepts pouvant faire l'objet d'une description sur une couleur sont donnés par le Tableau 17. Selon ce que désire le concepteur, les concepts générateurs sont :

- {Teinte, Saturation, Luminosité} (mode TSL) ;
- {Rouge, Vert, Bleu} (mode RVB) ;
- la liste de couleurs {« vert sombre » ; « rouge brique » ; « or » ; …} (mode Liste).

Tableau 17. Concepts pour une couleur dans ChromoFormes

Concept	Propriétés
Teinte	rouge ; jaune ; vert(e) ; cyan ; bleu(e) ; magenta
Saturation	pastel ; moyen(ne) ; pur(e)
Luminosité	sombre ; moyen(ne) ; clair(e)
Rouge	faible ; moyen(ne) ; important(e)
Vert	faible ; moyen(ne) ; important(e)
Bleu	faible ; moyen(ne) ; important(e)
Couleur	vert sombre ; rouge brique ; or ; …

L'utilisateur peut alors fournir une description comme « *La scène est magenta, très claire et de saturation entre 0,7 et 0,9* ». Quelques solutions possibles sont (Annexe 5) :

A. μ_A=0,142 (r=174 ; v=60 ; b=241) ;

B. μ_B=0,170 (r=245 ; v=31 ; b=229) ;

C. μ_C=0,646 (r=187 ; v=37 ; b=171) ;

D. μ_D=0,944 (r=202 ; v=60 ; b=191) ;

E. μ_E=0,680 (r=167 ; v=24 ; b=203) ;

F. μ_F=0,316 (r=179 ; v=43 ; b=144).

Il peut aussi donner : « *La teinte est vraiment bleue, la luminosité très très claire et la saturation plus ou moins pastel* ». Des solutions possibles, avec l'ajout progressif des trois propriétés, sont proposées en Annexe 5.

4.5. Spécificités du projet

Ce projet est intéressant en soi, car il propose une nouvelle façon d'aborder les couleurs (l'exploration de l'univers des couleurs) en exploitant le modèle le plus proche de l'intuition de l'utilisateur. Il est aussi intéressant vis-à-vis de CordiFormes du fait de la multiplicité des modèles possibles et donc des ensembles de concepts générateurs possibles. De plus ce projet utilise directement un objet de la base de connaissances.

5. Conclusion

Nous venons de présenter trois projets de modélisation déclarative EngloFormes, Linéa-Formes et ChromoFormes. Ils sont tous basés sur l'application prototype de CordiFormes. Ils ont permis de mettre en évidence les différents éléments de la plate-forme.

Maintenant, il faudra encore mettre en place d'autres projets mais aussi une grosse application pour continuer d'évaluer les capacités ainsi que les limites de CordiFormes.

Nous terminerons cette étude par une présentation des grands thèmes qui n'ont pas encore été abordés et qu'il sera intéressant de développer autour de CordiFormes.

CHAPITRE II.7 : CONCLUSION

1. Introduction

Tout au long de cette seconde partie, nous avons présenté les différentes couches de CordiFormes et, principalement le noyau. Nous nous intéresserons ici à un certain nombre d'axes de recherches en cours d'étude liés au projet CordiFormes.

Nous nous intéresserons d'abord aux outils d'aide à la conception du modeleur déclaratif qui peuvent faciliter le travail du concepteur (section 2). Nous examinerons ensuite la possibilité d'introduire des méthodes d'apprentissage liées à l'utilisation d'une plate-forme et d'outils génériques (section 3). Enfin, avant de conclure cette partie, nous aborderons quelques notions de parallélisme (section 4).

2. Vers des outils d'aide à la conception

Concevoir un modeleur déclaratif demande la connaissance et la maîtrise d'un grand nombre de concepts (dans le sens général du terme) et d'outils. Le noyau proposé comporte beaucoup d'éléments différents qui ne sont pas toujours aussi importants les uns que les autres. L'idée est donc de proposer des « assistants » dont les objectifs sont :

- aider le concepteur dans sa réflexion ;
- le guider dans les choix des outils et des structures à utiliser ;
- préparer le code du futur modeleur.

Présentons d'abord quelques grandes idées d'outils puis intéressons nous à l'exemple d'un outil d'aide à la détermination des modes de génération.

2.1. Conception assistée

Pour définir tous ses éléments, le concepteur doit manipuler beaucoup de notions et, en particulier, définir un grand nombre de paramètres principalement liés aux concepts, aux domaines et aux propriétés de base. Le système doit aider le concepteur dans la définition de ses valeurs afin qu'il maîtrise les conséquences sur le fonctionnement du futur modeleur.

Ainsi, le système l'aide en proposant des valeurs par défaut. Par exemple, nous avons montré dans la partie I et dans l'annexe 3 comment déterminer automatiquement les différents coefficients liés à la définition d'un concept et, plus précisément, à la définition des propriétés de base de ce concept. Cependant, la détermination automatique n'est pas suffisante. Selon le

domaine d'application, le concepteur peut vouloir modifier certains paramètres. Par conséquent, il faut lui proposer des outils permettant de modifier les valeurs tout en visualisant l'impact de ces modifications.

Il dispose par exemple d'un outil permettant de visualiser une propriété de base d'un concept dans son domaine (si le domaine n'est pas défini sur $[0,1]_u$). L'ensemble des paramètres ($<\alpha,a,b,\beta>$, les fonctions L et R et les coefficients τ et γ) peut être modifié par le concepteur afin d'ajuster au mieux cette propriété. De plus, il peut la visualiser parmi les autres propriétés de base du concept. Il peut aussi prendre connaissance des propriétés élémentaires résultant de l'application des modificateurs et des opérateurs flous sur sa propriété de base.

CordiFormes propose aussi des outils permettant de visualiser les structures des objets (hiérarchie des concepts). Plus généralement, il est intéressant de disposer d'un environnement de haut niveau permettant de visualiser graphiquement les éléments introduits (concepts, contraintes…), de les modifier en ajoutant éventuellement le code nécessaire…

2.2. Aide à la recherche des modes de génération

Un Système à Base de Connaissances (SBC) se différencie d'un système informatique conventionnel de part la capacité de rendre compte de manière intelligible des différents raisonnements qu'il met en œuvre. La construction d'un tel système nécessite l'utilisation d'architectures informatiques spécialisées permettant de représenter de façon explicite les différentes connaissances manipulées et les différents mécanismes de raisonnement associés. DSTM ([TrT97]) est un environnement informatique répondant à cet objectif.

DSTM permet de représenter une démarche de résolution de problème modélisée en termes de *tâches* (identifiant un problème à résoudre) et de *méthodes* (identifiant un moyen de résoudre un problème) et propose des mécanismes permettant la mise en oeuvre de raisonnements opportunistes (sélection dynamique de la tâche la plus pertinente à réaliser, puis de la méthode la plus favorable à sa réalisation). Le paradigme de modélisation Tâche-Méthode est actuellement un dénominateur commun à la plupart des méthodologies d'acquisition des connaissances actuelles ([DKS93]) et de nombreux langages d'opérationnalisation ont été définis dans ce cadre. La spécificité de DSTM par rapport à ces nombreux langages dits de haut niveau est que les primitives de modélisation *tâche* et *méthode* et les mécanismes de sélection dynamique associés peuvent être adaptés en fonction de l'expertise étudiée. En d'autres termes, DSTM ne force pas à une représentation particulière de ce que doit être une tâche et/ou une méthode mais offre la possibilité d'adapter les structures de représentation et les différents mécanismes de sélection (i.e. comment sont sélectionnées les tâches et les méthodes, comment est réalisé le choix entre plusieurs méthodes possibles pour une même tâche, etc.).

Dans le cadre de l'intégration de ces travaux au projet CordiFormes, nous cherchons à construire un SBC dédié au conseil dans le processus de construction de modeleurs déclaratifs. Il est important de noter que le SBC recherché n'est pas un tuteur intelligent au sens utilisé en Environnement Interactif d'Apprentissage avec Ordinateurs : l'objectif n'est pas l'apprentissage d'une méthode de construction de modeleur déclaratif. Nous cherchons simplement à guider le concepteur dans ces choix de méthodes de génération associées aux différents objets acceptés par le futur modeleur.

Une première étude a mis en évidence l'intérêt d'une modélisation de type Tâche-Méthode et d'une Base de Connaissances explicite sous-jacente. Cette Base de Connaissances (en cours de définition) a pour buts (1) de représenter explicitement une ontologie des objets classiquement utilisés dans le domaine de l'image et (2) d'associer pour chacun des objets identifiés une description des différentes méthodes de génération connues. L'objectif à terme est d'appliquer une sélection dynamique des méthodes de génération les plus pertinentes à utiliser en fonction de critères aussi variés que les caractéristiques intrinsèques des objets considérés, les préférences du concepteur, l'environnement dans lequel sera intégré le futur modeleur (c'est-à-dire les habitudes des utilisateurs potentiels et du domaine étudié), etc.

Ce travail est en cours ([DeT97]) et relève d'une véritable ingénierie des connaissances. Ce constat est à l'origine du choix d'utiliser les techniques offertes par l'Intelligence Artificielle et plus particulièrement des travaux réalisés en acquisition des connaissances considérant le processus de construction d'un SBC comme une activité de modélisation. En effet, les notions de Tâche et de Méthode adoptées étant en cours de définition et susceptible d'évoluer, la flexibilité offerte par l'environnement DSTM va nous permettre de réaliser conjointement les phases de modélisation et d'implantation du SBC final.

Nous nous intéressons donc ici au guidage des choix des meilleurs modes d'exploration pour chacun des objets de la scène. Ceci s'effectue en fonction de leurs caractéristiques, des désirs du concepteur et de ceux du futur utilisateur.

Ce processus se décompose en cinq étapes :

- saisie des objets de la scène et de leurs caractéristiques (lois de composition par définition) (Figure 135A) ;
- saisie des graphes sémantiques des futures scènes (lois de composition par construction) (Figure 135B) ;
- détermination d'une première base de connaissance sur les objets (Figure 135C) ;
- saisie des critères attendus par le concepteur et l'utilisateur (Figure 135D) ;
- construction de graphes sémantiques enrichis (Figure 135E).

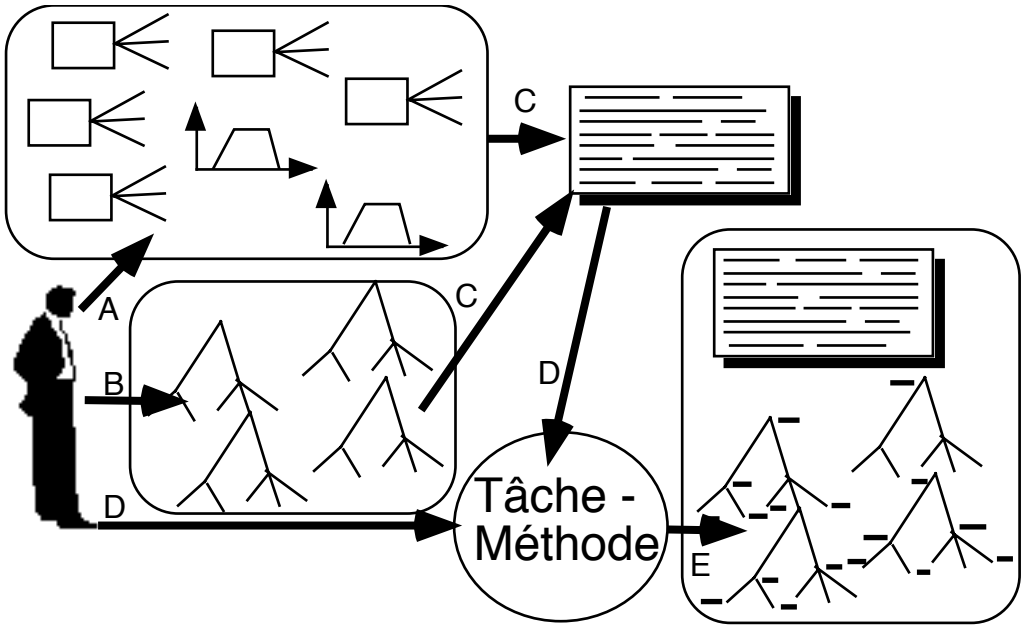

Figure 135. Recherche des bons modes d'exploration

3. Outils d'apprentissage

3.1. Introduction

Certaines études ont déjà été effectuées sur l'apprentissage en modélisation déclarative ([Des95a], [ChC94], [Cham95], [Cham96]). Les méthodes d'optimisation de la génération n'ont à ce jour pas donné de résultats significatifs dans le cas général. Certaines sont d'ailleurs encore en cours d'étude ([Cham97a]). Elles seront à inclure dans le noyau au même titre que les algorithmes classiques.

Ce qui nous intéresse plus particulièrement ici, ce sont les méthodes d'apprentissage liées à l'utilisation de la plate-forme. CordiFormes comporte un certain nombre de procédures d'apprentissage, non seulement au niveau de l'amélioration des temps de calcul mais aussi au niveau de l'adaptation du système à l'utilisateur. En effet, le concepteur propose des solutions générales, classiques pour le domaine d'application mais il faut que le modeleur s'ajuste en fonction de l'utilisateur. Cet apprentissage concerne essentiellement la connaissance du modeleur déclaratif, c'est-à-dire la sémantique liée aux différents concepts. Néanmoins, il s'intéresse aussi aux habitudes liées à la prise de connaissance et aux choix de génération.

[Chau92] propose une méthode d'apprentissage qu'on peut appeler « faible ». Elle consiste à définir une nouvelle propriété (un nouveau concept) comme une combinaison de concepts liés entre eux par des opérateurs. Cette méthode reste inapplicable tant qu'on ne maîtrise pas davantage l'utilisation de ces opérateurs (chapitre I.6). [Chau92] (puis [Chau94b]) avance l'idée d'une méthode d'apprentissage plus « forte » où l'utilisateur peut reprendre une pro-

priété pour modifier ses paramètres soit « à la main » soit par apprentissage par l'exemple. Cette dernière solution évite à l'utilisateur d'entrer dans les détails d'implémentation.

Dans CordiFormes, l'apprentissage de la sémantique se situe principalement à quatre niveaux :

1. modification de propriétés existantes ;
2. ajout de nouvelles propriétés ;
3. modification de concepts existants ;
4. ajout de nouveaux concepts.

Nous avons commencé à nous intéresser au premier niveau. Une première idée est de proposer des outils de modification de la fonction d'appartenance et des différents coefficients. Nous pourrions par exemple, mettre à disposition de l'utilisateur un outil identique à celui proposé au concepteur. Cette solution de facilité n'apparaît vraiment pas satisfaisante pour les raisons que nous venons d'évoquer pour le concepteur. Par contre, il serait intéressant d'étudier un module d'apprentissage ajustant de lui-même les connaissances en fonction du comportement de l'utilisateur.

3.2. Un exemple

3.2.1 Adaptation des connaissances d'un modeleur déclaratif aux différents utilisateurs

Lorsque le concepteur construit un modeleur déclaratif, il doit prendre des décisions concernant la forme et le comportement des propriétés de base des concepts, c'est-à-dire sur la fonction d'appartenance et les paramètres tels que le coefficient de translation élémentaire et le coefficient d'asymétrie. Le concepteur définit les connaissances en "standard". Or, ces valeurs ne sont pas forcément celles que l'utilisateur a en tête au cours de l'utilisation du modeleur. Par exemple, sa notion de boîte « grande » n'est pas forcément exactement la même que celle du concepteur.

En fait, ces différences ne sont malgré tout pas énormes (heureusement pour le concepteur). L'objectif est donc de mettre en place une méthode d'apprentissage ajustant les connaissances en fonction de l'utilisateur. Cet ajustement s'effectue en fonction des données, des résultats, des modifications … c'est-à-dire en fonction de toutes les actions effectuées par l'utilisateur sur le modeleur déclaratif comme l'énoncé des propriétés (description), la sélection et la "critique" des solutions ainsi que la modification de la description.

Tout ceci, nous amène à proposer l'ajout d'un module d'apprentissage travaillant à partir de toutes les informations récoltées tout au long des différentes phases du modeleur, c'est-à-dire la description de l'utilisateur, les solutions obtenues par la génération, les critiques et les modifications de l'utilisateur en fonction de ces solutions (Figure 136).

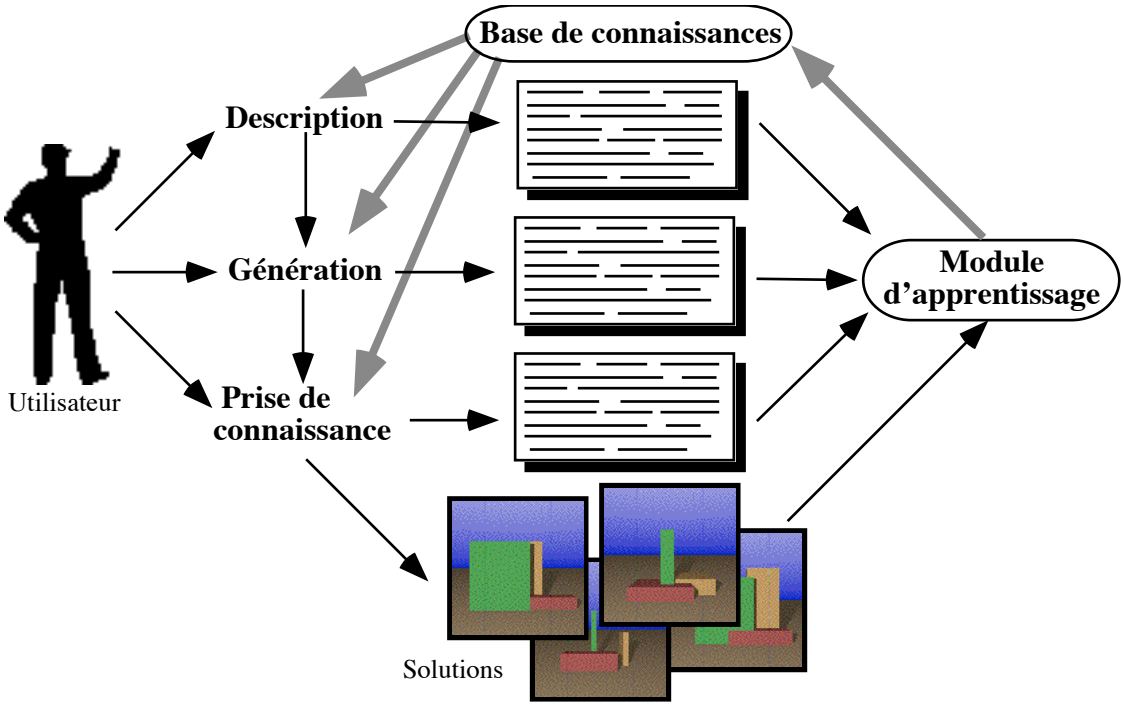

Figure 136. Le module d'apprentissage du modeleur déclaratif

Ce module travaille donc sur la base de connaissances du modeleur, c'est-à-dire sur toutes les informations définies dès le départ par le concepteur comme les concepts, les propriétés de base, les contraintes… Il ajuste les connaissances du modeleur en fonction de l'utilisateur.

3.2.2 Apprentissage sur les propriétés de base

Une des fonctions essentielles de ce module est l'ajustement des propriétés de base des différents concepts du modeleur ainsi que des opérateurs utilisés si nécessaire.

Intéressons nous en particulier à l'apprentissage des fonctions d'appartenance des propriétés de base. Il porte sur tous les paramètres liés à la représentation des propriétés de la description et, plus particulièrement : la forme (fonctions L et R), le quadruplet $<\alpha, a, b, \beta>$, la position dans le domaine… Le but est de trouver de "bonnes valeurs" pour la fonction d'appartenance selon les critères de l'utilisateur, c'est-à-dire de déterminer la nouvelle "forme" à partir de celle de la propriété de base et des informations tirées d'une session.

Le système EAGLE ([Mart97]) récupère donc toutes les informations (description, solutions, modifications…). Il propose une nouvelle fonction d'appartenance et de nouveaux coefficients prenant en compte les « remarques » de l'utilisateur (Figure 137).

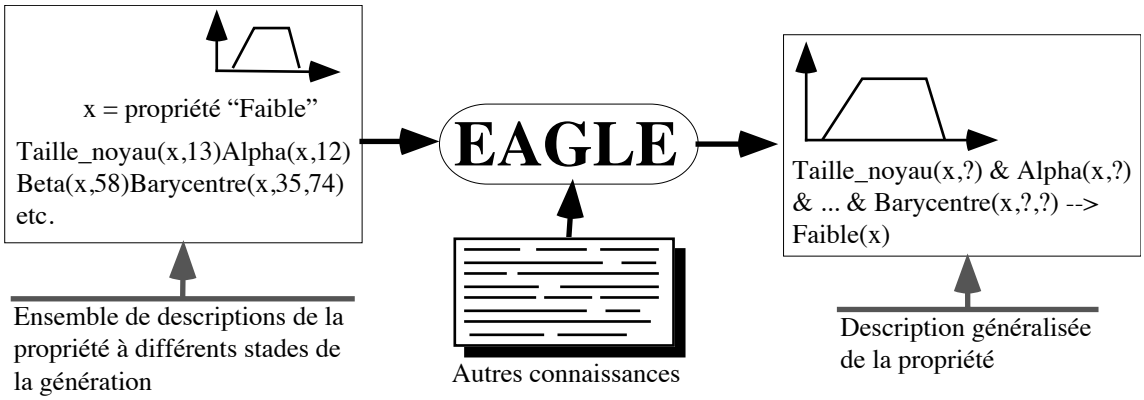

Figure 137. Principe d’apprentissage d’une propriété de base

EAGLE est un système d’apprentissage basé sur le cadre théorique de la Programmation Logique Inductive ou PLI ([BeG96]). L’objectif de la PLI est de former des définitions intentionnelles des concepts d’un domaine, à partir d’une description extensionnelle de ces concepts, contenant principalement des exemples représentatifs (dits exemples « positifs ») et non représentatifs (dits exemples « négatifs ») de ces concepts. Par exemple, à partir d’un ensemble contenant des descriptions de voitures du monde entier, le système doit induire une définition de la voiture anglaise comme étant "une voiture dont le volant est à droite". Pour un concept donné, le problème posé au système est donc de trouver une caractérisation générale de ce concept qui inclut les exemples positifs et exclut les exemples négatifs. Cette caractérisation doit ensuite pouvoir être utilisée pour la classification de nouveaux exemples dans un concept particulier. Dans le cadre de la plate-forme CordiFormes, ce système d’apprentissage peut permettre l’extraction de définitions des fonctions d’appartenance conformes à la vision de l’utilisateur, à partir de leurs différentes descriptions correspondant à des instants différents de la phase de génération.

Ce système est en cours d’étude mais l’idée d’effectuer l’apprentissage des fonctions d’appartenance en général semble prometteuse.

4. Parallélisation

Nous avons cherché à constituer le noyau de la plate-forme afin qu’il puisse être adapté à un fonctionnement en parallèle. En effet, la phase de calcul d’un modeleur déclaratif semble pouvoir être adaptée à une parallélisation.

L’idée de base consiste à considérer chaque tâche de génération comme un processus. La construction d’une scène est donc répartie sur autant de processus que de concepts qui la composent. Ces processus sont organisés de manière arborescente. Cette organisation est celle de l’arborescence des concepts. Par conséquent, la structure du réseau de processus change pour chaque objet. Ils ne peuvent évidemment pas fonctionner simultanément. Un processus

ne peut commencer à proposer des valeurs que lorsque toutes les contraintes d'initialisation ont été appliquées. De plus, certains auront un comportement différent. En effet, un processus associé à un concept composant d'un autre concept devra attendre les valeurs de ses propres concepts composants avant de générer lui-même sa valeur. Par contre, les processus associés aux objets de la scène seront seulement dépendants des contraintes d'initialisation.

Il reste un ensemble de problèmes en particulier ceux posés par la nécessité d'énumérer les concepts. Cette voie pourrait être intéressante pour améliorer l'efficacité de la phase de calcul et demandera une étude et une expérimentation poussée.

Remarques :

- Le choix du langage Java comme langage de programmation est aussi intéressant sur ce point. En effet, la notion de tâche doit permettre de tester cette parallélisation sans avoir à changer de langage.
- Les travaux sur la parallélisation des modeleurs déclaratifs sont très peu nombreux. On trouve simplement :
  - [BoP96] qui propose des parallélisations de l'évaluation des règles de production associées à la scène et à sa génération,
  - [Cham97a] qui propose une résolution de contraintes parallèle.

5. Pour finir

CordiFormes est donc une plate-forme pour la construction de modeleurs déclaratifs. Cette plate-forme est constituée de trois couches : le noyau, la couche interface et la couche prototype. Nous nous sommes principalement intéressés à la constitution du noyau et, plus particulièrement, à la création de la connaissance liée à l'application. Ainsi, nous avons implémenté les éléments du formalisme présenté dans la seconde partie : les *concepts*, les *domaines*, les *propriétés de base* et les *propriétés de la description*. Nous avons aussi proposé une méthode de *génération systématique* basée sur une génération récursive sur les concepts. Cette méthode utilise la notion de *tâche*. Pour améliorer cette génération, nous avons introduit la notion de *contraintes*. Celles-ci permettent d'initialiser les domaines, d'optimiser la génération et de calculer les mesures des concepts qui ne sont pas générateurs. Nous avons présenté ensuite les différents modules permettant d'organiser l'utilisation de ce noyau. Puis nous avons abordé les deux autres couches axées sur l'interface avec l'utilisateur. Nous avons terminé la présentation de CordiFormes par l'introduction d'études portant sur la constitution d'outils d'aide à la conception de modeleur, sur l'utilisation de techniques d'apprentissage permettant d'adapter le modeleur produit aux exigences de l'utilisateur et, enfin, sur la possibilité de paralléliser la génération systématique.

Le projet CordiFormes n'en est qu'à son début. Nous avons posé les premières pierres à l'édifice. De nombreuses perspectives sont envisageables.

Au niveau du noyau, la structuration des connaissances liées au domaine d'application est satisfaisante.

Il s'agit maintenant d'enrichir la base de connaissances en introduisant les concepts importants et, éventuellement, les tâches de génération qui leur sont spécifiques. Il sera intéressant d'ajouter, par exemple, des concepts liés aux matrices de voxels, d'affiner ceux sur les boîtes englobantes…

Il conviendra aussi d'étudier précisément les méthodes de génération basées sur les CSP afin d'affiner notre génération systématique et, surtout, notre gestion des contraintes.

Comme nous l'avons vu, le travail s'est porté ici essentiellement sur le noyau de CordiFormes. Il sera important d'affiner les éléments d'interface. En premier lieu, il faut se pencher sur l'intégration des éléments du projet MultiVues. Il faut aussi proposer, aussi bien pour la saisie de la description que pour la prise de connaissance, de nouveaux dialogues de plus haut niveau pour compléter ceux qui existent. D'une manière plus globale, il faut aussi réfléchir sur la constitution d'un environnement de programmation de plus haut niveau proposant outils et assistants.

L'essentiel du travail a porté sur les éléments de description et de génération. Tout ce qui concerne la prise de connaissance des solutions, aussi bien pour le noyau que pour la couche interface, demande à être approfondi.

Dans son état actuel, CordiFormes ne comprend aucun module avancé pour le raisonnement sur la description proposée par l'utilisateur ([GAT96]), c'est-à-dire sur la gestion poussée de la cohérence, sur la résolution d'énoncés ambigus et sur l'utilisation de connaissances par défaut.

Enfin, il faudrait prolonger l'étude des outils d'aide à la conception (assistant) ainsi qu'enrichir les méthodes d'apprentissage pour l'adaptation du modeleur à l'utilisateur.

CONCLUSION

Suite à un important travail de recherches bibliographiques, nous nous sommes rendus compte qu'il était temps de faire une synthèse et de regrouper ces études afin de proposer à un concepteur de modeleur déclaratif tous les outils nécessaires.

Nous avons alors proposé un nouveau formalisme basé sur les ensembles flous. Ce formalisme apparaît d'une part comme une synthèse et une unification de travaux existants et d'autre part apporte des éléments nouveaux comme la logique floue, la gestion linguistique de la négation, la réflexion sur l'interprétation de propriétés encore peu étudiées (propriétés quantifiées).

Après avoir présenté les notions de base (concept, domaine, propriété de base), nous nous sommes intéressés plus particulièrement aux propriétés les plus simples, appelées les propriétés élémentaires, aussi bien pour des énoncés affirmatifs que pour des négations. Ces propriétés sont construites à partir d'une propriété de base d'un concept (ensemble flou sur le domaine associé au concept) ou de valeurs de référence (pour les propriétés élémentaires paramétrées), d'un opérateur flou et d'un modificateur. Nous avons détaillé la méthode d'application des opérateurs sur la propriété de base ou l'intervalle utilisé. Nous avons aussi proposé des règles de construction automatique des propriétés de base.

Le traitement que nous avons choisi pour la gestion de la négation d'une propriété élémentaire est original. L'idée de base est de considérer que l'utilisation de la négation logique n'est pas suffisante et n'a pas réellement de sens. Nous nous basons sur l'idée que, lorsque l'utilisateur nie une propriété, il se réfère en fait à une autre propriété élémentaire sur le même concept. Nous avons donc proposé une méthode permettant de déterminer toutes les propriétés plausibles comme négation. Celles-ci pouvant être très nombreuses, nous avons aussi mis au point des règles pour les ordonner de façon à proposer d'abord celles qui ont le plus de chance d'être choisies.

Ensuite, après avoir recensé les différentes formes de propriétés qu'on peut trouver en modélisation déclarative et les différentes classifications qu'on peut en tirer, nous avons proposé des solutions de traitement.

Tous les travaux de modélisation déclarative ont en commun de traiter un sujet spécifique et ne sont pas directement adaptables à d'autres sujets. Cependant, la plupart d'entre eux mettent en oeuvre des techniques similaires. A partir de ces travaux et du formalisme flou, nous avons développé le projet CordiFormes, une plate-forme de programmation visant à fa-

ciliter la mise en œuvre de futurs modeleurs déclaratifs. Ses caractéristiques sont la simplicité, la souplesse de programmation, l'efficacité, l'extensibilité, la réutilisabilité et le prototypage rapide du modeleur.

CordiFormes est composée de trois couches. La principale couche est le noyau. Celui-ci implémente le formalisme flou. Il propose des structures de données génériques que le concepteur d'application adapte à son problème. De plus, il propose une méthode de génération, dite génération récursive, exploitant ces structures (concepts, domaines, propriétés…). Cette méthode est donc indépendante du domaine d'application. Cependant, le concepteur n'est pas obligé de l'utiliser et peut proposer sa propre méthode. Pour rendre cette génération utilisable et efficace, nous avons introduit la notion de contrainte. Elle permet d'ajuster un domaine ou une mesure d'un concept en fonction de la valeur d'autres concepts.

Le noyau comporte aussi une base de connaissances possédant un certain nombre d'objets. Le concepteur peut ainsi les utiliser directement. Néanmoins, il peut aussi créer ses propres objets en modifiant ou en composant ceux de la base. Ces objets constituent l'ontologie de la modélisation (géométrique en particulier).

Tous ces éléments sont exploités par trois modules se rapportant aux trois phases de la modélisation déclarative. Le module de description est chargé de construire le modèle de la scène décrite par l'utilisateur et doit gérer la description des propriétés à respecter. Le module de génération s'occupe de produire des solutions conformes à la description. Enfin, le module de prise de connaissance propose les informations utiles pour prendre connaissance des solutions.

CordiFormes comporte deux autres couches basées sur les problèmes de communication avec l'utilisateur. La couche interface propose un ensemble d'éléments d'interface et de dialogues permettant de mettre en place une application déclarative. La couche application donne la possibilité au concepteur d'évaluer son modeleur grâce à une application générique. Celle-ci comporte des outils de description et de prise de connaissance rudimentaires mais assez généraux.

Les trois projets développés avec CordiFormes (EngloFormes, FiloFormes et ChromoFormes) ont montré la justesse des structures et des algorithmes choisis. Cependant, il reste encore un grand nombre de points à préciser et à développer.

D'un point de vue formel, seules les propriétés élémentaires (paramétrées ou non) ont été étudiées précisément. Pour les autres, nous avons proposé des solutions plus ou moins satisfaisantes. Il faut désormais s'intéresser particulièrement aux propriétés quantifiées. Bien que très courantes (gestion des propriétés locales), leur formalisation n'est encore pas suffisante. Pour les autres, il faudra affiner le modèle afin de prendre en compte toutes les situations. Il

faudra aussi s'intéresser à la gestion de la négation pour ces propriétés. En effet, comment interpréter des énoncés comme « *La pente d'une dizaine de segments n'est pas très importante* » ou « *Le cube A n'est pas plus grand que le cube B* » ? En première analyse, il semble que la méthode appliquée aux propriétés élémentaires est adaptable mais cela demande à être confirmé. Enfin, il reste aussi à définir un bon modèle pour représenter et manipuler les propriétés de composition ("Et", "Ou"…}.

Du point de vue implémentation, CordiFormes demande encore à être enrichie. En effet, aussi bien en description qu'en prise de connaissance, les outils d'interface proposés sont assez rudimentaires. Il faut proposer des outils de niveau beaucoup plus haut tout en gardant le plus possible l'aspect générique. Nous avons vu aussi que plusieurs axes de recherches sont à explorer au niveau d'outils d'aide à la conception, de modules d'apprentissage puissants et de méthodes de parallélisation performantes. De plus, la base de connaissances est actuellement assez pauvre. Il est important d'effectuer une étude avancée de l'ontologie en modélisation déclarative et en modélisation géométrique. Cela nous permettra de déterminer et de concevoir les concepts à ajouter dans cette base.

Tout au long de ce travail, nous avons parlé d'intervalles, d'ensembles flous, d'intervalles flous et de contraintes. Il est donc important de s'intéresser de près aux techniques développées pour l'arithmétique des intervalles ainsi que pour les CSP et leurs applications aux ensembles flous (CSP flous). Notre traitement s'est naturellement approché de ces techniques. CordiFormes peut énormément profiter de ces travaux. Cependant, il convient d'être prudent car, en CSP par exemple, les techniques sont souvent proposées pour des contraintes symétriques entre deux concepts terminaux. Or, nous proposons des contraintes n-aires. De plus, elles ne sont pas toujours inversibles.

BIBLIOGRAPHIE

All83 J. F. Allen, "Maintaining Knowledge about Temporal Intervals", Communications of the ACM, Vol. 26, n°11, novembre 1983, pp. 832-843

ArG96 K. Arnold, J. Gosling, "*Le langage Java*", International Thomson Publishing France, Paris, 1996, 330 pages

ADG84 G. Adorni, M. DiManzo, F. Giunchiglia. "From Descriptions to Images: What Reasoning in between?", T. O'Shea (ed.), Proceedings of the 5th European Conference on Artificial Intelligence: Advances in Artificial Intelligence (ECAI-84), Elsevier, 1984, pp. 359-368.

AMF83 G. Adorni, M. Di Manzo, G. Ferrari, "Natural Language Input for Scene Description", European Chapter of ACL'83, Pisa, Italie, 1983, pp. 175-182

Bea89 C. Beasse, "Description formelle de formes", Rapport de DEA, Rennes, 1989, 74 pages

BeF91 E. Benoit, L. Foulloy, "Symbolic Sensors", AIMaC'91, Kyoto, 1991, pp 131-136

BeG96 Bergadano F., Gunetti D., "Inductive Logic Programming: from Machine Learning to Software Engineering", MIT Press, 1996

BLP96 P. Bosc, L. Liétard, H. Prade, "On fuzzy queries involving fuzzy quantifiers", ECAI'96.

BoP96 P.-F. Bonnefoi, D. Plemenos, "Propositions pour une modélisation déclarative parallèle", AFIG'96, Dijon, 27-29 novembre 1996, pp. 152-161

Bouc93 B. Bouchon-Meunier, "*La logique floue*", Presses Universitaires de france, Coll. Que sais-je, 1993, 128 pages

Bour97 M. Bourneton, "Formalisation des relations binaires dans le cadre de la modélisation déclarative, en utilisant la théorie des sous-ensembles flous", Rapport de DEA, Nantes, Septembre 1997, 53 pages

Buh94 H. Bühler, "*Réglage par logique floue",* Presses polytechniques et universitaires romandes, Lausanne, 1994, 181 pages

Bro43 V. Brondal, "*Essais de linguistique générale*", Copenhague, 1943

Bro48 V. Brondal, "*Les parties du discours : Etude sur les catégories linguistiques*", Copenhage, 1948, 173 pages

CBB96 J. Charlet, B. Bachimont, J. Bouaud, P. Zweigenbaum, "Ontologie et réutilisabilité : expérience et discussion" dans "*Acquisition et ingénierie des connaissances*", Cépadues-Editions, Toulouse, 1996, pp. 69-87

Cham95 L. Champciaux, "Apprentissage et Modélisation Déclarative", RR-95-8-INFO, Ecole des Mines de Nantes, Nantes, décembre 1995, 104 pages

Cham96 L. Champciaux, "De l'applicabilité des techniques d'apprentissage en Modélisation Déclarative", 3IA'96, Limoges, 3-4 avril 1996, pp 163-185

Cham97 L. Champciaux, "Declarative Modelling : Speeding Up the Generation", CISST'97, Las Vegas, USA, 30 juin - 2 juillet 1997, pp. 120-129

Cham97b L. Champciaux, "Classification : A Basis for Understanding Tools in Declarative Modeling", CompuGraphics'97, Villamoura, Portugal, GRASP, 15-18 décembre 1997, pp. 106-116

Chau92 D. Chauvat, "Modélisation déclarative : Etude de techniques de contrôle spatial", Rapport de DEA, Nantes, 1992, 139 pages

Chau94a D. Chauvat, "Création de paysages imaginaires par contrôle spatial", 2èmes journées de l'AFIG, Toulouse, 1994, pp 105-116

Chau94b D. Chauvat, "Le projet VoluFormes : un exemple de modélisation déclarative avec contrôle spatial", Thèse de doctorat, Nantes, Décembre 94, 225 pages

ChC94 L. Champciaux, C. Colin, "Etude d'une méthode d'apprentissage en modélisation géométrique déclarative", 2èmes journées de l'AFIG, Toulouse, 1994, pp 129-140

ChM94a D. Chauvat, P. Martin, "Contraintes en modélisation géométrique. Etude bibliographique", RR-IRIN-47, Nantes, février 1994, 29 pages

ChM94b D. Chauvat, P. Martin, "Un résolveur de contraintes adapté à la génération déclarative de scènes", RR-IRIN-60, Nantes, septembre 1994, 20 pages

Col90 C. Colin, "Modélisation déclarative de scènes à base de polyèdres élémentaires", Thèse de doctorat, Rennes, Décembre 1990, 266 pages

Col92 C. Colin, "Les propriétés dans le cadre d'une modélisation géométrique déclarative", MICAD 92, Paris, 1992, pp 75-94

Cou92 J.-P. Couwenberg, "*L'indispensable pour maîtriser la couleur*", Marabout, Alleur (Belgique), 1992, 471 pages

DAG86 M. DiManzo, G. Adorni, F. Giunchiglia, "Reasoning about scene descriptions" IEEE Proceedings - Special Issue on Natural Language, Vol. 74, n°7, 1986, pp. 1013-1025.

Dan95 M. Daniel, “Une première approche de la modélisation déclarative de courbes”, RR-IRIN-81, Nantes, Janvier 1995, 11 pages

DaL96 M. Daniel, M. Lucas, “Towards Declarative Geometric Modeling in Mechanics”, IDMME’96, 15-17 avril 1996, Nantes, pp. 455-464

Dem94 C. Demoisson, “Vers la modélisation déclarative de courbes”, Rapport de DEA, Nantes, Septembre 1994, 80 pages

DiK94 P. Diamond, P. Klaeden, “*Metric Spaces of Fuzzy Sets*”, World Scientific, Pergamon, 1994

Dje91 N. Djedi, “Modélisation en synthèse d’images : utilisation d’une méthodologie déclarative”, Thèse de doctorat, Toulouse, Novembre 1991, 196 pages

DKS93 David J.M., Krivine J.P., Simmons R., “*Second Generation Expert Systems*” (David J.M., Krivine J.P., Simmons R. eds), Springer-Verlag, 1993

DPS93 D. Dubois, H. Prade, S. Sandri, “On Possibility/Probability Transformations”, R. Lowen and M. Roubens (eds.), Fuzzy Logic, pp. 103-112

DPT88 D. Dubois, H. Prade, C. Testemale, “Weighted Fuzzy Pattern Matching”, Fuzzy Sets and Systems, vol. 28, n° 3, Décembre 1988, pp 313-331

DuP80a D. Dubois, H. Prade, “*Fuzzy Sets and Systems - Theory and Applications*”, Academic Press, New York, 1980

DuP80b D. Dubois, H. Prade, “Unifying View of Comparison Indices in a Fuzzy Set-Theoretic Framework”, Fuzzy Set and Possibility Theory, Ronald R. Yager (Ed.), Pergamon Press, 1980, pp. 3-13

DuP85 D. Dubois, H. Prade, “*Théorie des possibilités : Application à la représentation des connaissances en informatique”*, MASSON, Paris, 1985, 248 pages

DuP90 D. Dubois, H. Prade, “Measuring Properties of Fuzzy Sets : A General Technique and Its Use in Fuzzy Query Evaluation”, Fuzzy Sets and Systems, Vol. 38, 1990, pp. 137-152

DuP93 D. Dubois, H. Prade, “Ensembles flous, raisonnement et décision”, Rapport IRIT/93-52-R, Toulouse, Décembre 1993, 75 pages

Don93 S. Donikian, “Une approche déclarative pour la création de scènes tridimensionnelles : application à la conception architecturale”, Thèse de Doctorat, Rennes, 1993, 149 pages

DuS95 O. Ducrot, J.-M. Schaeffer et al., “*Nouveau dictionnaire encyclopédique des sciences du langage”*, Editions du Seuil, Paris, 1995, 670 pages

Elk89 G. Elkharroubi, “Etude des propriétés attachées à un arbre de modélisation”, Rapport de DEA, Rennes, 1989, 63 pages

FDP95 H. Fargier, D. Dubois, H. Prade, “Problèmes de satisfaction de contraintes flexibles : une approche égalitariste”, Revue d’intelligence artificielle, Vol. 9, n°3, 1995, pp. 311-354

Fau95 D. Faucher, “Définition déclarative d'ensembles de plans de l'espace”, Rapport de DEA, Nantes, Juin 1995, 86 pages

Fla96 D. Flanagan, “*Java in a Nutshell*”, O’Reilly International Thomson, Paris, 1996, 492 pages

Fron94 A. Fron, “*Programmation par contraintes*”, Addison-Wesley, Paris, 1994, 393 pages

GAT96 E. Giunchiglia, A. Armando, P. Traverso, A. Cimatti, “Visual Representation of Natural Language Scene Description”, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 26, n°4, 1996, pp. 575-589

GCR93 V. Gaildrat, R. Caubet, F. Rubio, “Conception d’un modeleur déclaratif de scènes tridimensionnelles pour la synthèse d’images”, MICAD’93, Paris, Hermès, 1993, pp. 265-284

GFT92 F. Giunchiglia, C. Ferrari, P. Traverso, E. Trucco, “Understanding Scene Descriptions by Integrating Different Sources of Knowledge”, International Journal of Man Machine Studies, Vol. 37, 1992, pp. 47-81

GoY97a J. Gosling, F. Yellin, “*Les API de Java : Le noyau*”, International Thomson Publishing France, Paris, 438 pages

GoY97b J. Gosling, F. Yellin, “*Les API de Java : AWT et Applets*”, International Thomson Publishing France, Paris, 349 pages

Gru93a T. R. Gruber, “A translation approach to portable ontology specifications”, in Knowledge Acquisition, vol. 5, 1993, pp. 199-220

Gru93b T. R. Gruber, “Toward Principle for the Design of Ontologies Used for Knowledge Sharing”, in Guarino, N. and Poli, R. (eds), “*Formal Ontology in Conceptual Analysis and Knowledge Representation*”, 1993

GuG95 N. Guarino, P. Giaretta, “Ontologies and Knowledge Bases”, N.J.I. Mars (ed.), “*Toward Very Large Knowledge Bases*”, IOS Press, Amsterdam, 1995

Hall66 E. T. Hall, “*La dimension cachée*”, Presse Universitaire de France, collection Que sais-je?, 1966, 254 pages

HBC91 J.-P. Haton, N. Bouzid, F. Charpillet, M.-C. Haton, B. Lâasri, H. Lâasri, P. Marquis, T. Mondot, A. Napoli, "*Le raisonnement en intelligence artificielle. Modèles, techniques et architectures pour les systèmes à base de connaissance*", InterEdition, Paris, 1991, 475 pages

Hic81 A. Hickethier, "*Le cube des couleurs*", Dessain & Tolra, Paris, 1981

Hje35 L. Hjelmslev, "*La catégorie des cas (1)*", Acta Jutlandica, 1935

Hje37 L. Hjelmslev, "*La catégorie des cas (2)*", Acta Jutlandica, 1937

Hor89 L. R. Horn, "*A Natural History of Negation"*, The University of Chicago Press, Chicago, 1989, 637 pages

Jim97 M. Jimenez, "*La psychologie de la perception*", Flamarion, Col. Dominos, Paris, 1997, 127 pages

KaB92 D. Kalra, A. H. Barr, "A Unified Framework for Constraint-based Modeling", Actes de Computer Graphics International'92, 1992, pp. 675-695

Khe95 L. Khemlani, "GENWIN : A Generative Computer Tool For Window Design in Energy-Conscious Architecture", Building and Environment, vol. 30, n°1, 1995, pp. 73-81

KGC97 G. Kwaiter, V. Gaildrat, R. Caubet, "DEM^{2}ONS : A High Level Declarative Modeler For 3D Graphics Applications", CISST'97, Las Vegas, USA, 30 juin--2 juillet 1997, pp. 149-154

Koc94 S. Kochhar, "CCAD : A Paradigm for Human-Computer Cooperation in Design", IEEE Computer Graphics and Applications, vol. 14, n°3, mai 1994, pp. 54-65

Koc97 S. Kochhar, "The WWW and Innovative Product Development for the Discrete Manufacturing Industry : A Vision and Research Agenda", CISST'97, Las Vegas, USA, 30 juin--2 juillet 1997, pp. 78-87

Leg90 D. Legrand, "*La couleur imprimée : mode d'emploi*", Trait d'union graphique, Paris, 1990

LeG90 D. Le Goff, "Modélisation déclarative et morphologie urbaine", Rapport de DEA, NANTES, Octobre 1990, 107 pages

Lie96 S. Liege, "La Modélisation Déclarative Incrémentale : Application à la Conception Urbaine", Thèse de Doctorat, Nantes, Décembre 1996, 204 pages

LMM89 M. Lucas, D. Martin, P. Martin et D. Plemenos, "Le projet ExploFormes : quelques pas vers la modélisation déclarative de formes", Journées AFCET-GROPLAN, Strasbourg, 1989, publié dans BIGRE, no 67, janvier 1990, pp. 35-49

Luc91 M. Lucas, “Equivalence Classes in Object Shape Modeling”, IFIP TC5/WG 5.10 Working Conference on Modeling in Computer Graphics, Tokyo, Japan, avril 1991

Luc93 M. Lucas, “Conception assistée par ordinateur et modélisation déclarative de formes”, Colloque PRIMECA, Chatenay-Malabry, Novembre 1993, pp. 93-98

Luc94 M. Lucas, “A propos de la notion de complexité d'un ensemble de segments de droite dans le plan”, RR-IRIN-45, Nantes, Janvier 1994, 57 pages

Luc97 M. Lucas, “Declarative Modelling : Examples of Application Domains”, CISST’97, Las Vegas, USA, 30 juin - 2 juillet 1997, pp. 526-535

LuP94 M. Lucas, F. Poulet, “Modélisation déclarative de monuments mégalithiques : le projet MégaFormes”, RR-IRIN-58, Nantes, Septembre 1994, 67 pages

Mac92 R. Maculet, “ARCHIPEL : Intelligence Artificielle et Conception Assistée par Ordinateurs en Architecture”, Thèse de Doctorat, Paris, Juillet 1992, 212 pages

MaM89 P. et D. Martin, “Un système de génération déclarative de polyèdres”, RR-LIST 89-02, Nantes, juin 1989, 76 pages

MaM90 P. et D. Martin, “Systèmes à base de règles pour la génération et l'exploration d'univers de formes”, RR-LIST 90-12, Nantes, 1990, 22 pages

MaM93 P. et D. Martin, “Declarative Generation of a Family of Polyhedra”, GraphiCon'93, St Petersburg, Russia, septembre 1993

MaM96 P. et D. Martin, “PluriFormes : un environnement pour le développement de modeleurs déclaratifs”, RR-IRIN-144, Décembre 1996, 50 pages

Mart97 E. Martienne, “EAGLE : un système d'apprentissage inductif”, Actes des Journées Ingénierie des Connaissances et Apprentissage Automatique (JICAA'97), Roscoff, 20-22 mai 1997, pp. 41-51

Mar90 J.-Y. Martin, “Synthèse d'images à l'aide d'automates cellulaires”, Thèse de doctorat, Rennes, Décembre 1990, 228 pages

Mat93 L. Mastyska, “Logic Programming with Fuzzy Sets”, Technical Report, City University, London, December 1993, 20 pages

MBF96 G. Mauris, E. Benoit, L. Foulloy, “Fuzzy sensors : an overview”, A paraître dans Fuzzy Set Methods in Information Engeneering : A Guided Tour of Application, John Wiley, 1996, 20 pages

Mou94 J.-P. Mounier, “Modélisation déclarative de quelques paramètres de production d'images réalistes”, Rapport de DEA, Nantes, 1994, 150 pages

Mou96 J.-P. Mounier, “Modélisation déclarative de parcours pour l’analyse des ambiances architecturales et urbaines”, AFIG’96, Dijon, 27-29 novembre 1996, pp. 162-172

Mul91 C. Muller, “*La négation en français”*, Publications romanes et françaises, Genève, 1991, 470 pages

Oft94 Observatoire français des techniques avancées, “*Logique floue”*, Masson, Paris, 1994, 295 pages

Pac92 D. Pacholczyk, “Contribution au traitement logico-symbolique de la connaissance”, Thèse d’état, Paris6, 1992, 628 pages.

Pac97 D. Pacholczyk, “A Fuzzy Analysis of Linguistic Negation of Nuanced Property in Knowledge-Based Systems”, ECSQARU/FAPR’97, 9-12 juin, Seminaris, Bad Honnef, 1997, à paraître.

Paj94 L. Pajot-Duval, “Modélisation déclarative de configurations de segments de droite”, Thèse de doctorat, Nantes, Juin 1994, 142 pages

Ple91 D. Plemenos, “Contribution à l'étude et au développement des techniques de modélisation, génération et visualisation de scènes : le projet MultiFormes“, Thèse de Doctorat d'état, Nantes, 1991, 308 pages

Ple94 D. Plemenos, “La modélisation déclarative en synthèse d’images, tendances actuelles et perspectives”, RR-MSI94-03, Limoges, mars 1994, 56 pages

PoL96 F. Poulet, M. Lucas, “Modelling Megalithic Sites”, Proceedings of Eurographics'96, septembre 1996, pp. 279-288

PRJ97 P. Poulin, K. Ratib, M. Jacques, “Sketching Shadows and Highlights to Position Lights”, Computer Graphics International’97, 1997, pp. 56-63

Pou94a F. Poulet, “Modélisation déclarative de scènes tridimensionnelles : Le projet spatioformes Vocabulaire sur quelques propriétés, Méthodes de génération”, 3IA'94, Limoges, 1994, pp 51-74

Pou94b F. Poulet, “Modélisation déclarative de scènes tridimensionnelles par énumération spatiale : le projet SpatioFormes“, Thèse de doctorat, Rennes, Juin 1994, 135 pages

Pou96 F. Poulet, “Le projet MégaFormes : réalisations, résultats”, RR-IRIN-112, Nantes, Janvier 1996, 33 pages

RMD96 A. Rau-Chaplin, B. MacKay-Lyons, T. Doucette, J. Gajewski, X. Hu, P. F. Spierenburg, “Graphics Support for a Worl-Wide-Web Based Architectural Design Service”, Compugraphics'96, 1996, pp. 83-92

RMS96 A. Rau-Chaplin, B. MacKay-Lyons, P. F. Spierenburg, “The LaHave House Project : Towards an Automated Architectural Design Service”, Cadex'96, 1996, pp. 24-31

RRL94 L. Rondeau, R. Ruelas, E. Levrat, G. Ris, “Modélisation de surfaces par règles floues”, Revue internationale de CFAO et d’infographie, Vol. 9 - n°5, 1994, pp. 633-644

SBR94 J. J. Shah, G. Balakrishanam, M. T. Rogers, S. D. Urban, “Comparative Study Of Procedural and Declarative Feature Based Geometric Modeling”, IFIP’94 - Feature modelling and recognition in advanced CAD/CAM systems, n°2, Valenciennes, France, 24-26 mai 1994, pp. 647-671

Sch81 P. Scheffe, “On Foundations of Reasoning with Uncertain Facts and Vague Concepts”, dans “Fuzzy Reasoning and its Applications”, 1981, pp. 189-216

SDS93 C. Schoeneman, J. Dorsey, B. Smits, J. Arvo, D. Greenberg, “Painting with Light”, Actes de Siggraph’93 (Anaheim, California, USA, 1-6 août 1993), *Computer Graphics Proceedings*, Annual Conference Series 1993, ACM SIGGRAPH, New York, 1993, pp. 143-146

Sir97 D. Siret, “Propositions pour une approche déclarative des ambiances dans le projet architectural. Application à l’ensoleillement”, Thèse de doctorat, Nantes, juin 1997, 323 pages

SnK92 John M. Snyder, James T. Kajiya, “Generative Modeling : A Symbolic System for Geometric Modeling”, Actes de Siggraph'92 (Chicago, USA , 26-31 juillet 1992), *Computer Graphics Proceedings*, Annual Conference Series 1992, ACM SIGGRAPH, New York, 1992, pp. 369-378

Sny92 John M. Snyder, “Interval Analysis for Computer Graphics”, Actes de Siggraph'92 (Chicago, USA , 26-31 juillet 1992), *Computer Graphics Proceedings*, Annual Conference Series 1992, ACM SIGGRAPH, New York, 1992, pp. 121-130

TcT97 Tchounikine P., Trichet F., “DSTM: a Framework to Support Modelling Problem-Solving within the Task-Method Paradigm”, RR-IRIN-157, Nantes, juillet 1997

Tor96 V. Torra, “Negation Functions Based Semantics for Ordered Linguistic Labels”, Int. Jour. of Intelligent Systems, 1996, p. 975-988.

Ton95 J.-R. Tong-Tong, “*La logique floue*”, Hermès, Paris, 1995, 160 pages

TrT97 Trichet F., Tchounikine P., “Reusing a Flexible Task-Method Framework to Prototype a Knowledge Based System”, Actes de la 9ième conférence internationale Software Engineering and Knowledge Engineering (SEKE'97), Madrid, Espagne, juin 1997, pp. 192-200

Tsa93 E. Tsang, "*Foundations of Constraint Satisfaction*", Academic Press, 1993, 421 pages

Tve77 A. Tversky, "Features of Similarity", Psychological Review, 1977, pp. 4-84

Woo91 R. F. Woodbury, "Searching for Designs : Paradigm and Practice", Building and Environment, vol. 26, n°1, 1991, pp. 61-73

Yag82 R. R. Yager, "Level Sets of Membership Evaluation of Fuzzy Subsets", Fuzzy Sets and Possibility Theory : Recent Developments (R.R. Yager, eds), Pergamon Press, Oxford, 1982, pp. 90-97

Yag84 R. R. Yager, "General Multiple-Objective Decision Functions and Linguistically quantified Statement", Int. J. of Man-Machine Studies, Vol. 21, 1984, pp. 389-400

Zad65 L. A. Zadeh, "Fuzzy Sets, Information and Control", vol. 8, 1965, pp 338-353

Zad83 L. A. Zadeh, "A Computational Approach to Fuzzy Quantifiers in Natural Languages", Computer Mathematics with Applications, Vol. 9, 1983, pp. 149-183

Zad87 L. A. Zadeh, "Similarity relations and Fuzzy ordering", Selected Papers of L.A. Zadeh, 3 (4), 1987, pp. 387-420

ZCB87 R. Zwick, E. Carestein, V. Bubescu, "Measure of Similarity among Fuzzy Concepts - a Comparative Analysis", Int. J. of Approximate Reasonning, Vol. 1, 1987, pp. 221-242

ZHH96 C. Zeleznik, K. P. Herndon, J. F. Hughes, "SKETCH : An Interface for Sketching 3D Scene", Actes de Siggraph'96 (août 1996), *Computer Graphics Proceedings*, Annual Conference Series 1996, ACM SIGGRAPH, New York, 1996, pp. 163-170

Contributions personnelles

CDMM97a C. Colin, E. Desmontils, J.-Y. Martin, J.-P. Mounier, "Modélisation déclarative : la vision de l'utilisateur", RR-IRIN-158, Nantes, mai 1997, 12 pages.

CDMM97b C. Colin, E. Desmontils, J.-Y. Martin, J.-P. Mounier, "Modèle Utilisateur d'un Modeleur Déclaratif", Journées "Modeleurs Géométriques", Grenoble, 17-19 septembre 1997

CDMM97c C. Colin, E. Desmontils, J.-Y. Martin, J.-P. Mounier, "Working Modes with a Declarative Modeler", CompuGraphics'97, Villamoura, Portugal, GRASP, 15-18 décembre 1997, pp. 117-126

Des95a E. Desmontils, "Les modeleurs déclaratifs", RR-IRIN-95, Nantes, septembre 1995, 150 pages

Des95b E. Desmontils, “Formalisation des propriétés en modélisation déclarative à l’aide des sous-ensembles flous”, RR-IRIN-106, Nantes, décembre 1995, 41 pages

Des96 E. Desmontils, “Une formalisation des propriétés en modélisation déclarative à l’aide des ensembles flous”, 3IA’96, Limoges, 3-4 avril 1996, pp 87-105

DeM97a E. Desmontils, J.-Y. Martin, “La propriété dans tous ses états”, RR-IRIN-148, Nantes, mars 1997, 34 pages

DeM97b E. Desmontils, J.-Y. Martin, “Properties Taxonomy in Declarative Modeling”, CISST’97, Las Vegas, 30 juin - 2 juillet 1997, pp. 130-138

DeM98 E. Desmontils, J.-Y. Martin, “Une plate-forme pour la construction de modeleurs déclaratifs”, RR-IRIN-167, Nantes, janvier 1998, 22 pages

DeP96a E. Desmontils, D. Pacholczyk, “Modélisation déclarative en synthèse d’images : traitement semi-qualitatif des propriétés imprécises ou vagues”, AFIG’96, Dijon, 27-29 novembre 1996, pp. 173-181

DeP96b E. Desmontils, D. Pacholczyk, “Apport de la théorie des ensembles flous à la modélisation déclarative en synthèse d’images”, LFA’96, Cépaduès Editions, Nancy, 4-5 décembre 1996, pp. 333-334

DeP97a E. Desmontils, D. Pacholczyk, “Interprétation du vague dans la description linguistique d’une scène en Modélisation Déclarative”, RR-IRIN-150, Nantes, avril 1997, 27 pages

DeP97b E. Desmontils, D. Pacholczyk, “Vers un traitement linguistique des propriétés en modélisation déclarative”, Revue Internationale de CFAO et Infographie, Vol. 12, n°4, septembre 1997, pp. 351-371

DeT97 E. Desmontils, F. Trichet, “Introduction d'une Base de Connaissances de type tâche/méthode pour assister la conception de modeleurs déclaratifs”, RR-IRIN-166, Nantes, décembre 1997, 24 pages

LuD95 M. Lucas, E. Desmontils, “Les modeleurs déclaratifs”, Revue Internationale de CFAO et Infographie, Vol. 10, n°6, 1995, pp. 559-585

PaD97a D. Pacholczyk, E. Desmontils, “A Qualitative Approach to Fuzzy Properties in Scene Description”, CISST’97, Las Vegas, 30 juin - 2 juillet 1997, pp. 139-148

PaD97b D. Pacholczyk, E. Desmontils, “Vers une description nuancée des scènes en synthèse d’images”, LFA’97, Cépaduès Editions, Lyon, 3-4 décembre 1997, pp. 185-192

ANNEXES

ANNEXE 1 : LES SOUS-ENSEMBLES FLOUS

1. Introduction

L'objectif de cette annexe est de présenter rapidement les principales notions et résultats en théorie des sous-ensembles flous. Les principaux documents qui ont servi à construire cette annexe sont : [Zad65] , [DuP85], [DPT88] , [HBC91], [Bou93], [DuP93], [Mat93], [Oft94], [Buh94] et [Tong95].

2. Les sous-ensembles flous

Définition A1.1 : Un *ensemble flou* A sur le *domaine* (appelé aussi *référentiel*) T est défini par la donnée d'une fonction d'appartenance μ_a à valeur dans [0,1]. $\mu_a(t)$ est le *degré d'appartenance* de $t \in T$ à A.

μ_a : $T \rightarrow [0,1]$

$t \rightarrow \mu_a(t)$

Définition A1.2 : On appelle *Support de A* noté Supp(A), le sous-ensemble non flou des valeurs de T telles que : $Supp(A) = \{t \in T : \mu_a(t) \neq 0\}$

Définition A1.3 : On appelle *Noyau de A* noté Noy(A), le sous-ensemble non flou des valeurs de T telles que : $Noy(A) = \{ t \in T : \mu_a(t) = 1\}$

Remarque : $Noy(A) \subseteq Supp(A) \subseteq T$

Définition A1.4 : La *hauteur* d'un ensemble flou A est le plus fort degré d'appartenance d'un élément du domaine appartenant à A. Elle est notée h(A) telle que : $h(A) = \sup_{t \in T}(\mu_a(t))$

Définition A1.5 : Un ensemble flou est dit *normalisé* si : $h(A) = 1$, autrement dit si $Noy(A) \neq \emptyset$

Définition A1.6 : La *cardinalité* de A notée |A| évalue le degré global avec lequel les éléments de T appartiennent à A. Soit : $|A| = \int_T \mu_a(t)dt$

Remarques :

- On a, si T est fini : $|A| = \sum_{t \in T} \mu_a(t)$
- Pour la suite, dans ce genre de formule, nous supposerons être dans les bonnes conditions de convergence (support compact).

Définition A1.7 : L'ensemble A est dit *précis* si : $\exists\, t \in T, \forall\, u \in T : \mu_a(t) = 1$ et $\mu_a(u) = 0$ si $u \neq t$ d'où : Supp(A) = Noy(A) = {t}

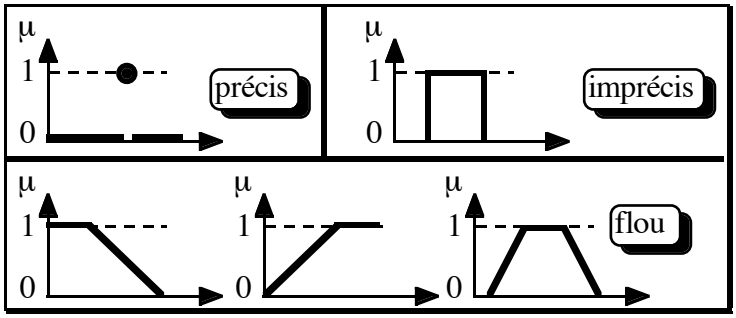

Figure 138 : Propriétés précises, imprécises et floues

Définition A1.8 : L'ensemble A est dit *imprécis* si : $\forall\, t \in T : \mu_a(t) \in \{0,1\}$ d'où : Supp(A) = Noy(A)

Définition A1.9 : L'ensemble A est dit *flou* si : $\exists\, t \in T : \mu_a(t) \in]0,1[$ d'où : $Noy(A) \subset Supp(A)$

Les fonctions d'appartenance peuvent être classées en deux groupes divisibles eux-mêmes en trois formes :

- les ensembles classiques :
  - les valeurs précises (Figure 139 : C1),
  - les intervalles classiques (Figure 139 : C2),
  - les ensembles classiques quelconques (Figure 139 : C3) ;
- les ensembles flous :
  - les valeurs floues (Figure 139 : F1),
  - les intervalles flous de forme Z, Π ou S (Figure 139 : resp. F2a, F2b et F2c),
  - les ensembles flous quelconques (Figure 139 : F3).

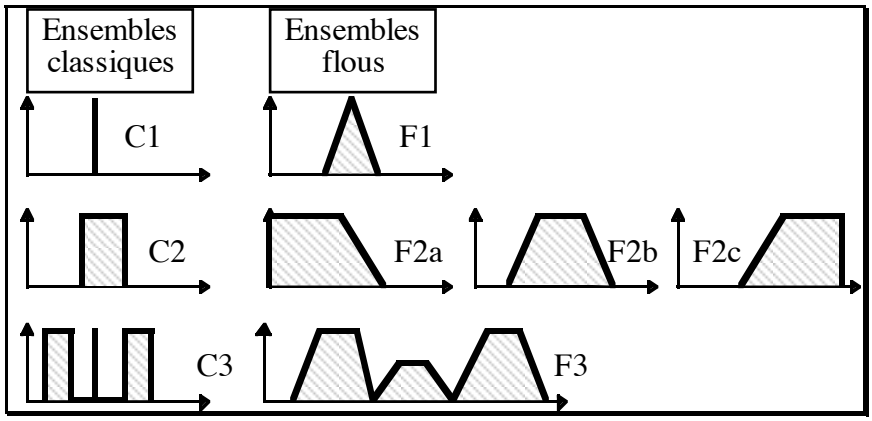

Figure 139 : Classement des différentes formes de fonctions d'appartenance en théorie des ensembles flous

Nous pouvons noter que les formes Z, Π ou S ne sont parfois que partiellement entre les bornes du domaine (Figure 140). Nous pouvons associer dans cette figure les formes A à la forme Π, les formes B au complémentaire d'une forme Π, les formes C1 et D2 à la forme Z et, enfin, les formes C2 et D1 à la forme S.

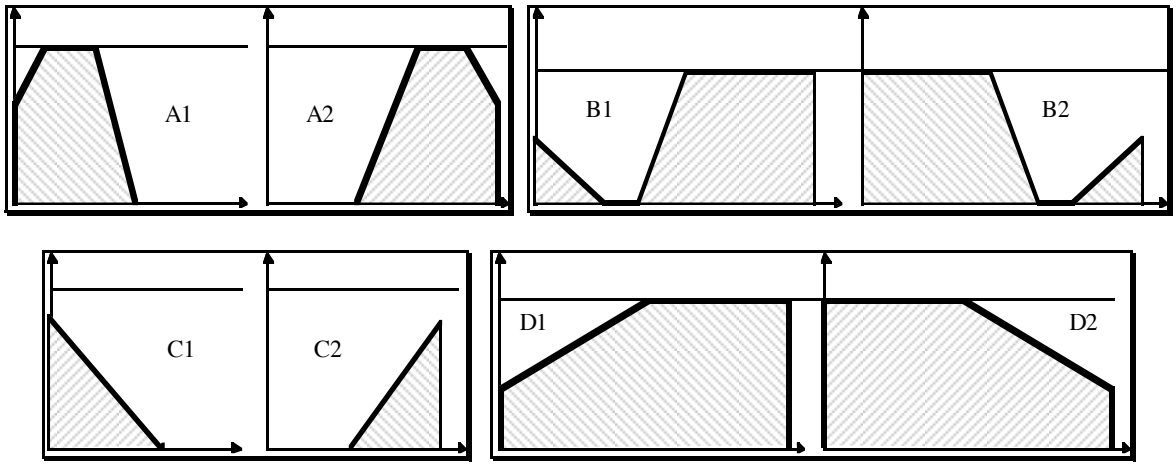

Figure 140 : Exemples de formes classiques « dégénérées »

Définition A1.10 : On appelle *coupe de niveau* α ou *α-coupe* de l'ensemble flou A pour une valeur donnée $\alpha \in [0,1]$, le sous-ensemble non flou A_α de T défini par : $A_\alpha = \{ t \in T : \alpha \in [0,1], \mu_a(t) \geq \alpha \}$

Définition A1.11 : On appelle *α-niveau* une α-coupe telle que : $A^\alpha = \{ t \in T : \alpha \in [0,1], \mu_a(t) = \alpha \}$

Définition A1.12 : On appelle *barycentre* ou *centre de gravité* d'un ensemble flou A de T, l'élément g de T tel que :

$$g = \frac{\int_T t * \mu(t) dt}{\int_T \mu(t) dt} \text{ ou, si T est discret, } g = \frac{\sum_T t * \mu(t)}{\sum_T \mu(t)}$$

3. Opérations sur les ensembles flous

3.1. Relations entre deux ensembles flous A et B de T

Soit $t \in T$, k un entier

- l'inclusion : $A \subseteq B \Leftrightarrow \mu_a(t) \leq \mu_b(t)$
- l'égalité : $A = B \Leftrightarrow \mu_a(t) = \mu_b(t)$
- la distance : $d(A,B) = \sum_{t \in T} \left(\left| \mu_a(t) - \mu_b(t) \right| \right)^{\frac{1}{k}}$ avec souvent k = 2 (distance Euclidienne)

3.2. Composition de deux ensembles flous A et B de T

Soit $Max(a,b) \equiv a \vee b$ et $Min(a,b) \equiv a \wedge b$

Soit $t \in T$, $\alpha \in [0,1]$, C un ensemble flou de T :

- l'intersection : $C = A \cap B \Leftrightarrow \mu_c(t) = \min(\mu_a(t), \mu_b(t)) = \mu_a(t) \wedge \mu_b(t)$ (voir les t-normes)
- l'union : $C = A \cup B \Leftrightarrow \mu_c(t) = \max(\mu_a(t), \mu_b(t)) = \mu_a(t) \vee \mu_b(t)$ (voir les t-conormes)
- l'addition ou somme bornée : $C = A \oplus B \Leftrightarrow \mu_c(t) = \min(1,\mu_a(t)+\mu_b(t)) = 1 \wedge (\mu_a(t) + \mu_b(t))$
- la somme algébrique (probabiliste) : $C = A + B \Leftrightarrow \mu_c(t) = \mu_a(t) + \mu_b(t) - \mu_a(t) . \mu_b(t)$
- la multiplication : $C = A \otimes B \Leftrightarrow \mu_c(t) = \max(0,\mu_a(t)+\mu_b(t)-1) = 0 \vee (\mu_a(t) + \mu_b(t) - 1)$
- la multiplication algébrique (probabiliste) : $C = A . B \Leftrightarrow \mu_c(t) = \mu_a(t) . \mu_b(t)$
- la multiplication par une constante : $C = \alpha . A \Leftrightarrow \mu_c(t) = \alpha . \mu_a(t)$
- la soustraction : $C = A - B \Leftrightarrow \mu_c(t) = \mu_a(t) \wedge (1 - \mu_b(t)$
- la soustraction restreinte ou soustraction bornée : $C = A \langle - \rangle B \Leftrightarrow \mu_c(t) = 0 \vee (\mu_a(t) - \mu_b(t))$
- la soustraction absolue : $C = |A - B| \Leftrightarrow \mu_c(t) = |\mu_a(t) - \mu_b(t)|$
- la soustraction symétrique : $C = A \Delta B \Leftrightarrow X = (A - B) \cup (B - A)$
- le complément de degré λ : $C = \overline{A}^{\lambda} \Leftrightarrow \mu_c(t) = \dfrac{1 - \mu_a(t)}{1 + \lambda\mu_a(t)}$

3.3. Complémentation d'un ensemble flou A de T

Soit $t \in T$, C un ensemble flou de T : $C = \overline{A} \Leftrightarrow \mu_c(t) = 1 - \mu_a(t)$

C est appelé le *complémentaire* de A dans T.

3.4. T-Normes et T-Conormes

Définition A1.13 : une *norme triangulaire* (*t-norme*) est une fonction T de $[0,1]x[0,1] \rightarrow [0,1]$ qui vérifie pour tous x, y, t et z de [0,1] :

1. $T(x,y)=T(y,x)$ (commutativité) ;
2. $T(x,T(y,z))=T(T(x,y),z)$ (associativité) ;
3. $T(x,y) \leq T(z,t)$ si $x \leq z$ et $y \leq t$ (monotonie) ;
4. $T(x,1) = x$ (élément neutre 1)

Définition A1.14 : de façon analogue, une *conorme triangulaire* (*t-conorme*) est une fonction $\perp$ de $[0,1]x[0,1] \rightarrow [0,1]$ qui vérifie pour tous x, y, t et z de [0,1] :

1. $\perp(x,y)=\perp(y,x)$ (commutativité) ;
2. $\perp(x,\perp(y,z))=\perp(\perp(x,y),z)$ (associativité) ;
3. $\perp(x,y) \leq \perp(z,t)$ si $x \leq z$ et $y \leq t$ (monotonie) ;
4. $\perp(x,0) = x$ (élément neutre 0)

Remarques :

- N'importe quelle t-norme est un opérateur d'intersection (Tableau 18).
- N'importe quelle t-conorme est un opérateur d'union (Tableau 18).
- Leurs propriétés montrent que t-normes et t-conormes peuvent être considérées comme duales. On peut passer de l'un à l'autre par l'intermédiaire d'une négation (fonction n de [0,1] → [0,1] telle que n(0)=1, n(1)=0 et n(x)≤n(y) si x≥y). La négation n(x)=1-x est la plus classique.

Tableau 18 : Exemples de t-normes et de t-conormes

t-norme	t-conorme	Nom
min(x,y)	max(x,y)	Zadeh
x . y	x + y - x . y	Probabiliste
max(x + y - 1, 0)	min(x + y, 1)	Lukasiewicz

3.5. Moyennes

La notion de moyenne est intermédiaire entre t-norme et t-conorme. Les moyennes ont la forme suivante : $M_f(a_1,\ldots,a_m) = f^{-1}(\frac{1}{m}\sum_{j=1}^{m} f(a_j))$

Remarque : Les moyennes usuelles sont définies avec $f(x)=x^{\alpha}$

Il est possible aussi de définir une moyenne pondérée :

$$M^f_{w_1,\ldots,w_m}(a_1,\ldots,a_m) = f^{-1}(\frac{1}{m}\sum_{j=1}^{m} w_j f(a_j))$$

$$avec : \sum_{j=1}^{m} w_j = 1$$

4. Relations floues

4.1. Produit cartésien de deux ensembles flous A de T et B de S

Soit $t \in T$, Soit $s \in S$,

Soit C tq :

μ_c : TxS → [0,1]

t,s → $\mu_c(t,s)$

$C = A \times B \Leftrightarrow \mu_c(t,s) = \mu_{a\times b}(t,s) = \min(\mu_a(t), \mu_b(s))$

4.2. Relation floue

Soit A_i des ensembles flous de T ($i \in [1,n]$) , $(t_1, ..., t_n) \in T^n$

Soit C tq :

μ_c : $T_1 x ... x T_n \rightarrow [0,1]$

$t_1, ..., t_n \rightarrow \mu_c(t_1, ..., t_n)$

$C = R(A_1, \ldots, A_n) \Leftrightarrow \mu_c(t_1, ..., t_n) = \mu_R(t_1, ..., t_n)$

4.3. Composition de relations floues binaires A et B

Soit x, y et v ∈ T, Soit C une relation binaire tq :

μ_c : $TxT \rightarrow [0,1]$

$x,y \rightarrow \mu_c(x,y)$

$C = A \circ B \Leftrightarrow \mu_c(x,y) = \mu_{a \circ b}(x,y) = Sup_v \min(\mu_a(x,v), \mu_b(v,y))$

5. Cas particuliers d'ensembles flous

Définition A1.15 : La *convexité* d'un ensemble flou A de T est telle que A est convexe si et seulement si $\forall \lambda \in [0,1], \forall x_1, x_2 \in T, \mu_a(\lambda x_1 + (1-\lambda)x_2) \geq \min(\mu_a(x_1), \mu_a(x_2))$

Remarque : si A et B sont deux ensembles flous convexes alors A ∩ B est convexe.

Définition A1.16 : Une *quantité floue* est un ensemble flou dans l'univers IR des nombres réels. Une quantité floue est supposée normalisée.

Définition A1.17 : Une quantité floue est dite *convexe* si : $\forall [a,b] \subset IR \quad \forall x \in [a,b]$: $\mu(x) \geq \min(\mu(a), \mu(b))$

Définition A1.18 : Un *intervalle flou* est une quantité floue convexe.

Remarque : lorsque la fonction d'appartenance d'un intervalle flou est semi-continue supérieurement (toute α-coupe est un intervalle fermé), cet intervalle est une généralisation de l'intervalle fermé. Nous supposerons dans la suite que les intervalles flous possèdent cette propriété.

Définition A1.19 : Un *Nombre flou* est un intervalle flou semi-continu supérieurement dont la fonction d'appartenance est à support compact (fermé et borné) dans r. Le noyau est réduit à un élément (valeur modale).

Remarques sur les quantités floues :

- Les opération habituelles sur les nombres réels sont définies aussi sur les quantités floues (opposé, inverse, puissance, exponentielle, signe, addition, soustraction, multiplication, division...).

- Une représentation très utilisée est la représentation L-R et en particulier la représentation trapézoïdale (voir annexe 2).

6. Variables et modificateurs linguistiques

6.1. Notion de variable linguistique

Définition A1.20 : A un domaine est associée une *variable linguistique* qui permet l'assertion suivante : « X est A » X est une variable linguistique associée au domaine T et A un ensemble flou définit sur le domaine T

Exemple : « *Jean est grand* » = « Taille(Jean) est grand » = « X est grand ». X est définie sur le domaine « Taille » (elle est de type « Taille » et prend pour valeur « grand ».

6.2. Notion de modificateur linguistique

Définition A1.22 : les *modificateurs linguistiques* permettent d'amplifier ou d'atténuer la signification initiale de certains attributs. C'est une fonction de [0,1] dans [0,1] qui s'applique sur la fonction d'appartenance d'un ensemble flou.

Opérations de modification d'un ensemble flou A de T (modificateurs linguistiques) :

Soit t ∈ T

- « très » ou Contraction : $cont(A) = \mu_a^2$
- « plus ou moins » ou Dilatation : $dil(A) = \sqrt{\mu_a}$
- « hautement » : $haut(A) = \mu_a^3$
- « intensément » : $Int(A) = \begin{cases} 2 * \mu_a si \mu_a \leq 0.5 \\ 1 - 2 * (1 - \mu_a) si \mu_a > 0.5 \end{cases}$
- « plutôt » : $Plut(A) = \begin{cases} 2 * \mu_a^2 si \mu_a \leq 0.5 \\ 1 - 2 * (1 - \mu_a)^2 si \mu_a > 0.5 \end{cases}$

ANNEXE 2 : LES FONCTIONS D'APPARTENANCE

1. Introduction

Nous allons parcourir quelques techniques fréquemment utilisées pour représenter les fonctions d'appartenance. Elles sont pratiques à manipuler et simples à mettre en place. Ces techniques sont tirées, entre autres, de : [DuP80a], [Bou93], [Buh94], [HBC91] et [Ton95].

2. Représentation par énumération

Cette solution consiste à définir une fonction d'appartenance en énumérant les degrés d'appartenance pour tous les éléments du domaine. Il est donc clair que cette solution n'est utilisable que pour un domaine discret (ou discrétisé) et dont le nombre d'éléments n'est pas trop important. Cette technique est applicable aussi pour les propriétés qui ne sont pas des intervalles flous.

2.1. Représentation par les parties croissantes et décroissantes

Pour simplifier un peu l'énumération, uniquement pour les intervalles flous et les nombres flous, une autre solution consiste à n'énumérer que les parties croissantes et décroissantes de la fonction. Entre les deux, le degré d'appartenance est forcément à 1.

3. Représentation L-R

3.1. Généralités

Pour faciliter les traitements, nous utilisons souvent les fonctions d'appartenance de type L-R ([DuP80a], [Tong95], [Bou93] et [HBC91]). Elles sont totalement définies par un quadruplet de réels $<\alpha,a,b,\beta>$ (avec $a \leq b$, α et $\beta > 0$), ou par quatre points $(a-\alpha, a, b, b+\beta)$, et par deux fonctions L et R appelées fonctions de forme. Celles-ci sont décroissantes et semi-continues supérieurement. Elles sont définies par :

L :	$\mathbb{R}^+$	$\rightarrow$	[0,1]	R :	$\mathbb{R}^+$	$\rightarrow$	[0,1]
	t	$\rightarrow$	L(t)		t	$\rightarrow$	R(t)

De plus :

- $\forall x<1$: $L(x)>0$ et $R(x)>0$; $\forall x>0$: $L(x)<1$ et $R(x)<1$;
- $L(0) = R(0) = 1$; $L(1) = R(1) = 0$;
- $L(+\infty) = R(+\infty) = 0$.

L'expression d'une fonction d'appartenance L-R définie par le quadruplet <α, a, b, β> et les fonctions L(x) et R(x) est :

$$\mu_{<\alpha,a,b,\beta>,L,R} : \mathbb{R} \rightarrow [0,1]$$

$$t \rightarrow \begin{cases} L((a-t)/\alpha) & \text{si } t < a \\ 1 & \text{si } a \leq t \leq b \\ R((t-b)/\beta) & \text{si } t > b \end{cases}$$

Remarque : Il existe un certain nombre de fonctions L-R, comme par exemple : $L(x) = R(x) = e^{-x}$ (voir Figure 141).

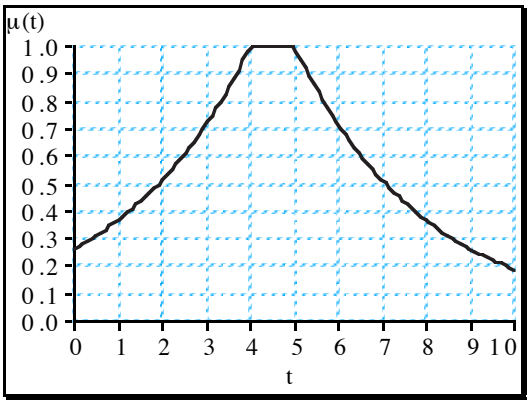

Figure 141. Exemple de fonction L-R

3.2. Cas particuliers de fonctions L-R

Nous choisirons des fonctions L(x) et R(x) avec des caractéristiques bien particulières. La première donne la forme de la courbe entre les deux premiers points (a-α et a) et la seconde entre le deux derniers (b et b+β). Entre les deux points intermédiaires (a et b), μ(t)=1. Avant le premier point (a-α) et après le dernier (b+β), μ(t)=0. Dans ce cas, nous autorisons des valeurs de α et β à 0 (les deux premiers points et/ou les deux derniers sont confondus).

L'expression d'une fonction d'appartenance L-R (Figure 142) définie par le quadruplet <α, a, b, β> (α, β ≥ 0) et les fonctions L(x) et R(x) ayant les caractéristiques ci-dessus se réécrit de la manière suivante :

$$\mu_{<\alpha,a,b,\beta>,L,R} : \mathbb{R} \rightarrow [0,1]$$

$$t \rightarrow \begin{cases} 0 & \text{si } t < a-\alpha \\ L((a-t)/\alpha) & \text{si } (\alpha \neq 0) \text{ et } (a-\alpha \leq t < a) \\ 1 & \text{si } a \leq t \leq b \\ R((t-b)/\beta) & \text{si } (\beta \neq 0) \text{ et } (b < t \leq b+\beta) \\ 0 & \text{si } t > b+\beta \end{cases}$$

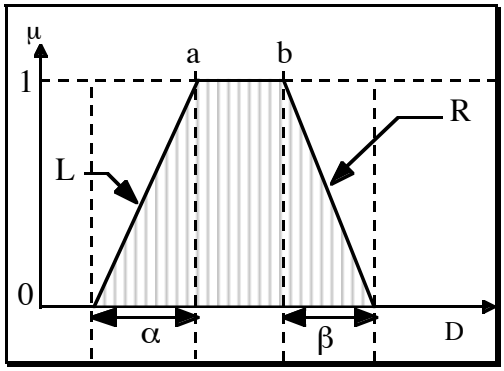

Figure 142 : expression d'une fonction d'appartenance L-R

Cette catégorie de fonction L-R possède un certain nombre de caractéristiques intéressantes. Soit la fonction d'appartenance d'un ensemble flou P définie par <α, a, b, β> et deux fonctions L(x) et R(x) bien choisies (comme celles présentées ci-dessus), on a alors :

- L'intervalle [a,b] est le noyau de la propriété ;
- α et β donnent la largeur des intervalles autour du noyau. Ils permettent de définir le segment support [a-α,b+β] ;
- α>0 ou β>0 si et seulement si P est floue ;
- α>0 ou β>0 et a≠b si et seulement si P est un intervalle flou ;
- α>0 ou β>0 et a=b si et seulement si P est un nombre flou ;
- α=0 et β=0 et a≠b si et seulement si P est imprécis (intervalle classique [a,b]) ;
- α=0 et β=0 et a=b si et seulement si P est précis (nombre classique {a}).

Les fonctions L(x) et R(x) les plus intéressantes ayant les caractéristiques que nous avons posées sont (voir Figure 143 et Figure 144) :

- $L_1(x) = R_1(x) =$ Max(0, 1-x) (ce sont les plus utilisées en T.S.E.F. et sont appelées fonctions trapézoïdales) ;
- $L_2(x) = R_2(x) =$ Max(0, $1-x^2$) ;
- $L_3(x) = R_3(x) =$ Max(0, $1-x)^2$;
- $L_4(x) = R_4(x) =$ 2*Max(0,$1-x)^2$ si x>0.5 et Max(0,$1-2*x^2$) sinon.

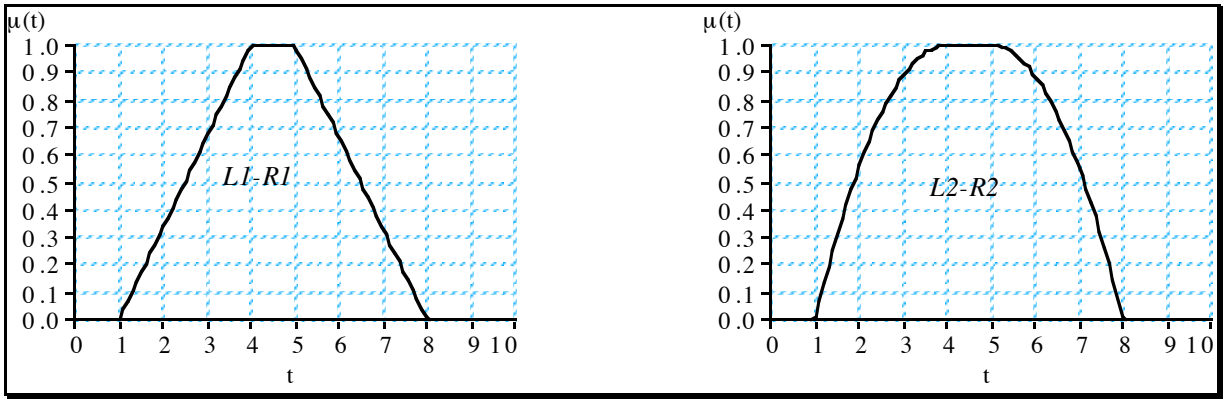

Figure 143. Fonctions L_1-R_1 et L_1-R_1

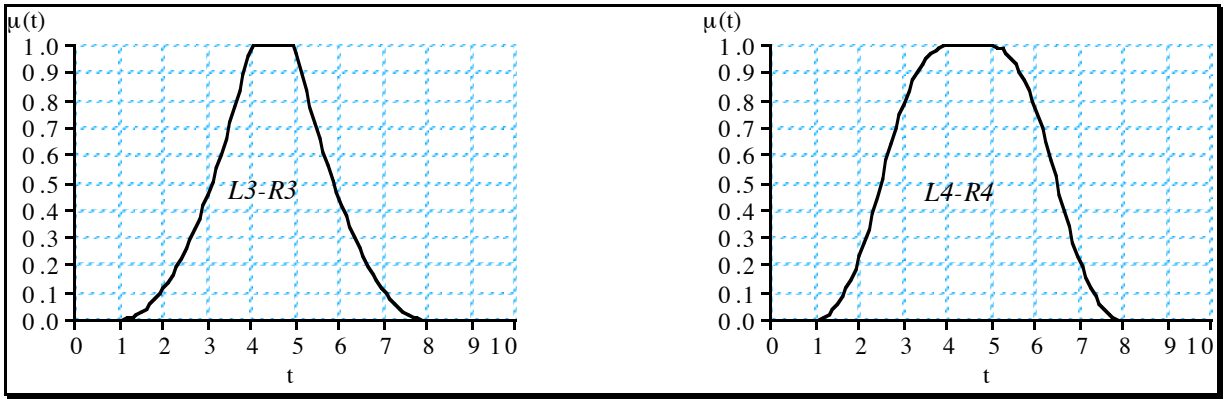

Figure 144. Fonctions L_3-R_3 et L_4-R_4

De plus, ces fonctions sont faciles à manipuler, simples à calculer et les fonctions pour modifier leur forme sont faciles à mettre en place. Elles permettent de faire des approximations satisfaisantes de fonctions complexes. C'est pour cela qu'elles sont très souvent utilisées en théorie des ensembles flous.

4. Représentation par une fonction

Enfin, il existe un certain nombre de fonctions utilisées pour n'importe quelle propriété. En voici quelques unes (voir Figure 145 et Figure 146) :

- Fonction 1 ; fonction en cloche, où t_0 détermine la position du sommet et a impose la largeur de la courbe : $\mu(t) = \frac{1}{1 + \left(\frac{t - t_0}{a}\right)^2}$;
- Fonction 2 ; fonction en cloche trigonométrique, où t_0 détermine la position du sommet et a impose la largeur de la courbe : $\mu(t) = \frac{1 + \cos\frac{\pi(t - t_0)}{2a}}{2}$ si $t_0-2a \le t \le t_0+2a$ et $\mu(t) = 0$ sinon ;
- Fonction 3 ; En extension de la fonction précédente, afin d'élargir le sommet : $\mu(t) = \frac{1 + \cos\frac{\pi(t - t_1)}{2a_1}}{2}$ si $t_1-2a_1 \le t \le t_1$, $\mu(t) = 1$ si $t_1 \le t \le t_2$, $\mu(t) = \frac{1 + \cos\frac{\pi(t - t_2)}{2a_2}}{2}$ si $t_2 \le t \le t_2+2a_2$, $\mu(t) = 0$ sinon ;
- Fonction 4 ; fonction composée par des morceaux de droite ;
- Fonction 5 ; fonction asymétrique où a et c déterminent la zone floue : $\mu(t) = 0$ si $x \le a$, $\mu(t) = \frac{2(t - a)^2}{(c - a)^2}$ si $a<x$ et $x<(c+a)/2$, $\mu(t) = 1 - \frac{2(t - c)^2}{(c - a)^2}$ si $(c+a)/2 \le x$ et $x<c$, $\mu(t) = 1$ sinon.

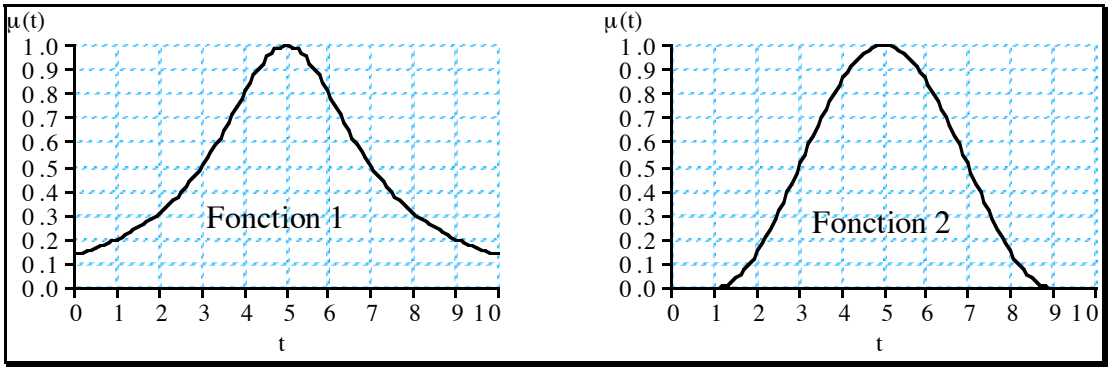

Figure 145. Exemples de fonctions

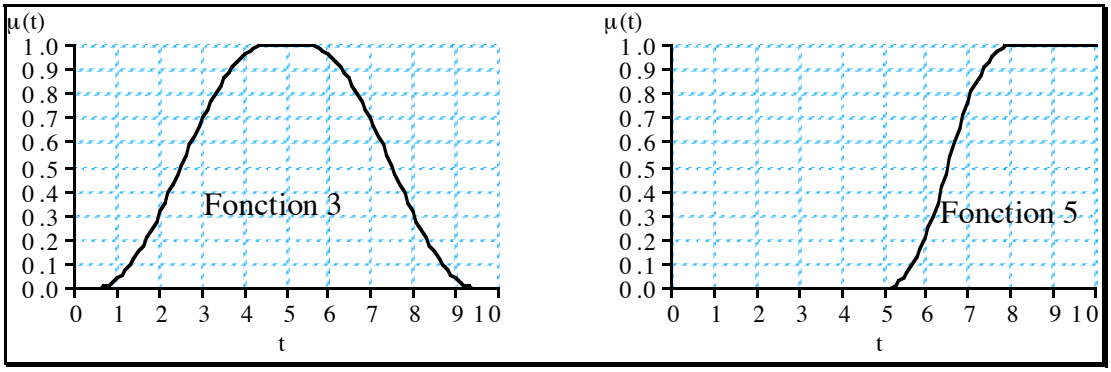

Figure 146. Autres exemples de fonctions

ANNEXE 3 : CALCUL AUTOMATIQUE DES PROPRIÉTÉS DE BASE

1. Construction automatique des propriétés de base d'un concept

L'objectif de ce paragraphe est de proposer des méthodes de calcul automatique des propriétés de base d'un concept permettant d'obtenir un premier ensemble. Nous supposerons dans la suite que les valeurs du domaine sont des valeurs numériques (sachant que l'on peut toujours s'y rapporter). Plus précisément, il s'agit de déterminer les fonctions d'appartenance des propriétés de bases ainsi que les coefficients de translation élémentaire et d'asymétrie.

1.1. Détermination de la fonction d'appartenance

La position et la forme des propriétés de la Figure 24 semblent correctes pour la majorité des concepts. Les critères de construction sont :

- la propriété « faible » dans la première moitié du domaine (autour du quart) ;
- la propriété « moyen » autour de la valeur médiane du domaine ;
- la propriété « important » dans la seconde moitié du domaine (autour du troisième quart) ;
- la valeur $(b+\beta)$ pour la propriété « faible » et $(a-\alpha)$ pour la propriété « important » se confondent et égale à la valeur médiane ;
- les propriétés « faible » et « important » ont une géométrie légèrement asymétrique et coupent la propriété « moyen » vers 0,5.

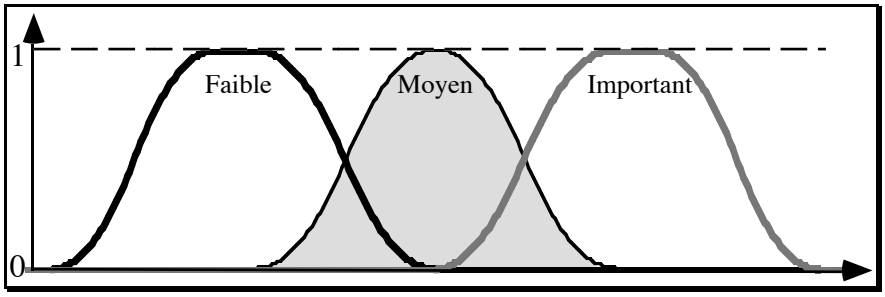

Figure 147. Propriétés de base d'un concept.

Remarque : a priori, le type des fonctions L et R utilisées importe peu ici. Nous proposons donc d'utiliser les fonctions les plus standard telles que les fonctions L_1-R_1 (linéaires) ou L_4-R_4 (avec un point d'inflexion comme celles de la Figure 24) présentées en annexe 2. Nous allons nous intéresser aux coefficients de la fonction d'appartenance $<\alpha, a, b, \beta>$.

La forme des propriétés dépend directement de la taille du domaine. Soit $[Bm,BM]_u$ le domaine, on a alors δ la taille du domaine telle que $\delta = BM-Bm$. Nous proposons de définir les

fonctions d'appartenance des propriétés de base comme indiquées dans le tableau 1 permettant d'obtenir la Figure 24.

Tableau 19. Définition automatique des fonctions d'appartenance

nom	α	a	b	β
faible	20% * δ	Bm + 25% * δ - 2% * δ	Bm + 25% * δ + 3% * δ	22% * δ
moyen	22% * δ	Bm + 50% * δ	Bm + 50% * δ	22% * δ
important	22% * δ	Bm + 75% * δ - 3% * δ	Bm + 75% * δ + 2% * δ	20% * δ

Nous pouvons noter que ces valeurs de coefficient ont permis d'obtenir les figures des paragraphes 1 à 7 du chapitre II.2 la Figure 148, la Figure 149 et la Figure 150 qui suivent.

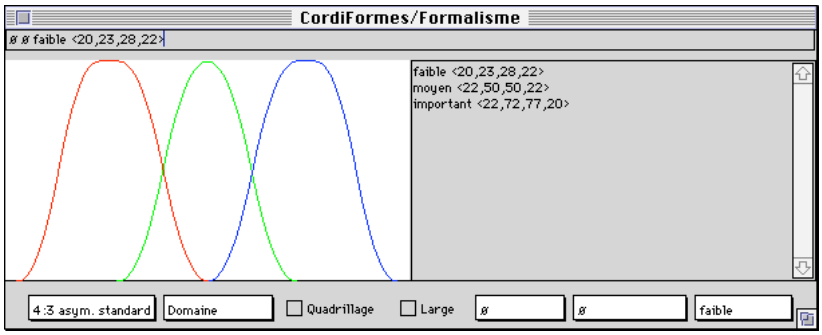

Figure 148. Domaine

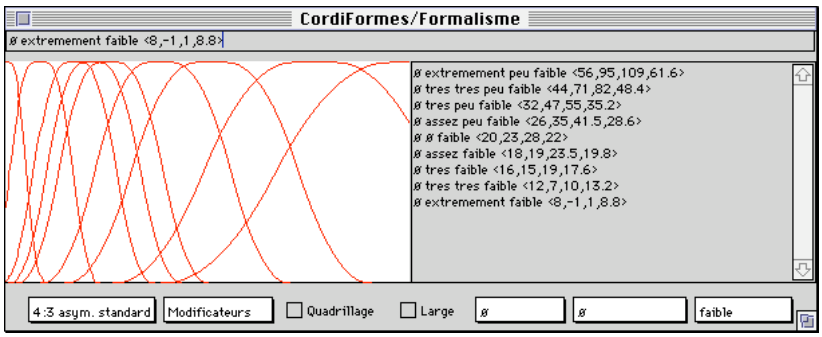

Figure 149. Modificateurs sur "faible" asymétrique

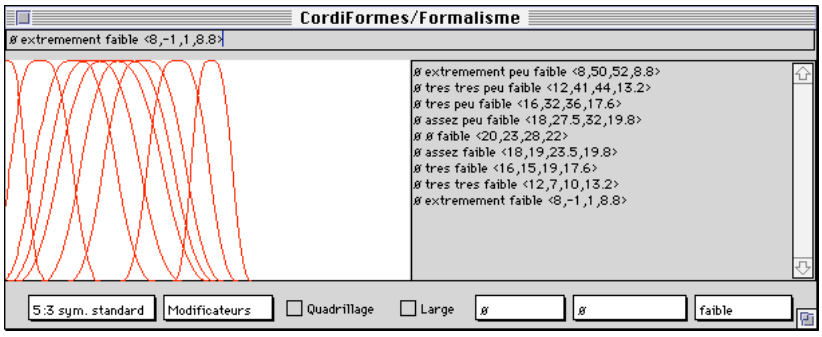

Figure 150. Modificateurs sur "faible" symétrique

1.2. Paramètres pour les modifications de la propriété de base asymétrique

Il reste maintenant à obtenir les deux coefficients associés à chacune des propriétés de base. Afin de faciliter les calculs, nous supposerons dans un premier temps les propriétés signées comme asymétriques. Plus précisément, les propriétés « faible » et « important » possèdent un même coefficient γ_{ik} négatif et supérieur à 1, c'est-à-dire que les translations sont augmentées lorsque l'on applique les modificateurs négatifs. Cette hypothèse semble cohérente avec les concepts asymétriques habituellement utilisés. Nous allons poser les contraintes suivantes :

1. lorsque l'on applique le modificateur « extrêmement », la fonction d'appartenance résultante doit être telle que BM (resp. Bm) soit dans le noyau pour une propriété positive (resp. négative) ;
2. lorsque l'on applique le modificateur « extrêmement peu », la fonction d'appartenance résultante doit être telle que Bm (resp. BM) soit dans le noyau pour une propriété positive (resp. négative).

Remarque : ces contraintes et ces hypothèses permettent d'assurer, pour chaque valeur du domaine un degré d'appartenance fort (souvent très proche ou égal à 1) à au moins une propriété issue de chaque propriété de base du concept. De plus, elles assurent que, pour les modificateurs les plus extrêmes, les bornes ont un degré d'appartenance maximum.

1.2.1 Détermination du coefficient de translation élémentaire

La contrainte 1 nous permet de calculer un τ_{ik} correct. Nous aurons pour une propriété positive : $b'_{ik} \geq BM$ et $a'_{ik} \leq BM$.

d'où $\tau_{ik} \geq \dfrac{BM - b_{ik} + |k| * 5\% * (b_{ik} - a_{ik})}{k}$ et $\tau_{ik} \leq \dfrac{BM - a_{ik} - |k| * 5\% * (b_{ik} - a_{ik})}{k}$

soit avec $b_{ik} = Bm + 75\% * \delta + 2\% * \delta$ et $a_{ik} = Bm + 75\% * \delta - 3\% * \delta$,

$$\boxed{\frac{\delta}{k}\left(\frac{23 + |k| * .25}{100}\right) \leq \tau_{ik} \leq \frac{\delta}{k}\left(\frac{28 - |k| * .25}{100}\right)}.$$

D'où, comme $k < 10$, nous avons : $\boxed{\tau_{ik} = \dfrac{\delta}{k}\left(\dfrac{25.5}{100}\right)}$.

Donc, si $k = k_{extrêmement}=6$: $\boxed{\tau_{ik} = \dfrac{\delta}{6}\left(\dfrac{25.5}{100}\right) = 4.25\% * \delta}$

Les propriétés négatives étant construites de manière symétrique, nous aurons alors comme coefficient de translation élémentaire $\tau_{ik} = -4.25\% * \delta$. Notons que le coefficient de translation élémentaire est proportionnel à la taille du domaine.

1.2.2 Détermination du coefficient d'asymétrie

La contrainte 2 nous permet de calculer un γ_{ik} correct. Nous aurons pour une propriété négative : $b'_{ik} \geq BM$ et $a'_{ik} \leq BM$

soit $|\gamma_{ik}| \leq \frac{BM - a_{ik}}{k * \tau_{ik} - |k|(b_{ik} - a_{ik}) * 5\%}$ et $|\gamma_{ik}| \geq \frac{BM - b_{ik}}{k * \tau_{ik} + |k|(b_{ik} - a_{ik}) * 5\%}$

donc : $\boxed{\frac{72\% * \delta}{k * \tau_{ik} + 0.25\% * \delta * |k|} \leq |\gamma_{ik}| \leq \frac{77\% * \delta}{k * \tau_{ik} - 0.25\% * \delta * |k|}}$

d'où, si on pose $\tau_{ik} = -4.25\% * \delta$, on a : $\boxed{\frac{72}{k * -4.25 + 0.25 * |k|} \leq |\gamma_{ik}| \leq \frac{77}{k * -4.25 - 0.25 * |k|}}$

d'où, avec $k = k_{\text{extrêmement peu}} = -6$, on a : $\boxed{\frac{72}{27} \leq |\gamma_{ik}| \leq \frac{77}{24}}$.

Soit, par approximation : $2{,}667 \leq |\gamma_{ik}| \leq 3{,}208$. Un coefficient d'asymétrie γ_{ik} pour une propriété négative à -3 semble donc une valeur correcte. Comme la propriété positive est construite de la même manière, elle aura le même coefficient d'asymétrie.

1.3. Paramétrage de la construction

Les calculs proposés dans le paragraphe précédant sont bons pour une valeur moyenne (centre de la propriété « moyen ») au milieu du domaine. Cependant, ce n'est pas toujours le cas. Nous allons donc proposer une généralisation de ce calcul en fonction d'une valeur moyenne spécifique. Soit M cette valeur moyenne. Nous aurons alors : δ=BM−Bm, δ'=M−Bm et δ''=BM−M. Nous allons prendre les mêmes critères et hypothèses que pour les calculs précédents en remplaçant la valeur médiane par une valeur moyenne. Nous aurons alors des propriétés définies par les fonctions d'appartenance présentées dans le tableau 2.

Tableau 20. Définition automatique des fonctions d'appartenance en fonction d'une valeur moyenne

nom	α	a	b	β
faible	40% * δ'	Bm + 50% * δ' - 4%*δ'	Bm + 50% * δ' + 6% * δ'	44% * δ'
moyen	44% * δ'	M	M	44% * δ''
important	44% * δ''	M + 50% * δ'' - 6% * δ''	M + 50% * δ'' + 4% * δ''	40% * δ''

Les calculs se déroulent d'une manière similaire pour obtenir finalement :

- pour la propriété positive $\frac{\delta''}{6}\left(\frac{49}{100}\right) \leq \tau_{ik} \leq \frac{\delta''}{6}\left(\frac{53}{100}\right)$ soit $\tau_{ik} = \frac{\delta''}{6}\left(\frac{51}{100}\right) = 8.5\% * \delta''$;
- par symétrie, $\tau_{ik} = -8.5\% * \delta'$ pour une propriété négative ;

- $\frac{50}{27}\frac{\delta}{\delta'} - \frac{28}{27} \le |\gamma_{ik}| \le \frac{25}{12}\frac{\delta}{\delta'} - \frac{23}{24}$ pour la propriété négative ;
- $\frac{50}{27}\frac{\delta}{\delta''} - \frac{28}{27} \le |\gamma_{ik}| \le \frac{25}{12}\frac{\delta}{\delta''} - \frac{23}{24}$ pour la propriété positive.

Remarque : si nous posons δ'= δ''= δ/2, nous obtenons les résultats du paragraphe précédent.

1.4. Cas de la fonction d'appartenance unique

Jusqu'ici, nous avons étudié la construction automatique des propriétés de base d'un concept en contenant 3. Parfois, le concept ne contient qu'une seule propriété de base (par exemple le concept de « densité »). La fonction d'appartenance d'une telle propriété doit être géométriquement symétrique et surtout placée sur une valeur M du domaine $[Bm,BM]_u$ (par défaut, autour de la valeur médiane Méd=(BM-Bm)/2). La taille du support doit être suffisamment importante. La forme des fonctions L et R ne sont pas importantes. Nous utiliserons les fonctions standard, c'est-à-dire soit L_1-R_1 soit L_4-R_4 (selon les fonctions de l'annexe 2). Le Tableau 21 (avec δ la taille du domaine telle que δ = BM-Bm) propose une définition possible de la fonction d'appartenance d'une telle propriété de base.

Tableau 21. Définition d'une propriété de base unique

nom	α	**a**	**b**	β
Vérifié	20% * δ	M - 5% * δ	M + 5% * δ	20% * δ

Comme pour les autres propriétés de base, nous allons nous intéresser aux coefficients associés à cette propriété. Une telle propriété peut être positive ou négative en fonction du domaine d'application et du concept considérés. De plus, M dépend totalement du concept. Posons les contraintes suivantes :

1. lorsque l'on applique le modificateur « extrêmement », la fonction d'appartenance résultante doit être telle que BM ou Bm soient dans le noyau ;
2. lorsque l'on applique le modificateur « extrêmement peu », la fonction d'appartenance résultante doit être telle que Bm ou BM soient dans le noyau.

Afin que τ_{ik} soit le plus petit possible en valeur absolue, nous allons étudier les solutions où il est calculé par rapport à Bm et par rapport à BM. Ceci permet, dans l'éventualité d'une propriété « vérifiée » définie comme une propriété symétrique (voir paragraphe plus loin) de garder toutes les propriétés modifiées définies au moins sur une partie du domaine.

Si τ_{ik} est calculé par rapport à BM, nous avons : $b'_{ik} \ge BM$ et $a'_{ik} \le BM$.

d'où $\tau_{ik} \ge \frac{BM - b_{ik} + |k| * 5\% * (b_{ik} - a_{ik})}{k}$ et $\tau_{ik} \le \frac{BM - a_{ik} - |k| * 5\% * (b_{ik} - a_{ik})}{k}$

soit avec $b_{ik} = M + 5\% * \delta$ et $a_{ik} = M - 5\% * \delta$:

$$\frac{BM - M - 5\% * \delta + |k| * .5\%\delta}{k} \leq \tau_{ik} \leq \frac{BM - M + 5\% * \delta - |k| * .5\% * \delta}{k}.$$

D'où $\tau_{ik} = \frac{BM - M}{k}$.

De manière symétrique (et avec des contraintes symétriques), nous avons : $\tau_{ik} = \frac{Bm - M}{k}$.

Par conséquent, nous avons :

$$\tau_{ik} = \pm Min(\left|\frac{Bm - M}{k}\right|, \left|\frac{BM - M}{k}\right|) = \pm Min(\frac{M - Bm}{|k|}, \frac{BM - M}{|k|})$$

Donc, si $k = k_{extrêmement}=6$ ou $k = k_{extrêmement\ peu}=-6$: $\tau_{ik} = \pm Min(\frac{M - Bm}{6}, \frac{BM - M}{6})$.

Il reste maintenant à calculer le coefficient d'asymétrie. Selon des calculs similaires à ce que nous avons fait pour les propriétés au paragraphe 1.2 nous pourrions obtenir une évaluation donnant γ_{ik}. Cependant, si l'on analyse la formule des modificateurs, on note aisément que γ_{ik} est un coefficient de τ_{ik} en ce qui concerne la translation. Si l'on veut respecter les règles ci-dessus, il faut $|\gamma_{ik}| * \tau_{ik}$ soit tel que le noyau soit déplacé sur la borne opposée à celle visée par τ_{ik}.

Nous pouvons en déduire facilement : $|\gamma_{ik}| = \frac{Max(\frac{M - Bm}{6}, \frac{BM - m}{6})}{Min(\frac{M - Bm}{6}, \frac{BM - m}{6})}$

Posons δ=BM–Bm, δ'=M–Bm et δ''=BM–M. Pour que cette propriété soit correcte, il faut qu'elle recouvre tout le domaine. Le Tableau 22 donne le signe de γ en fonction de la place de la fonction d'appartenance dans le domaine et du signe de la propriété.

Tableau 22. Détermination de γ en fonction de la position de la propriété

	$\delta' < \delta''$	$\delta' = \delta''$	$\delta' > \delta''$
$\tau > 0$	$\gamma > 1$	$\gamma = 0$	$\gamma < 1$
$\tau < 0$	$\gamma < 1$	$\gamma = 0$	$\gamma > 1$

Le coefficient de translation élémentaire τ_{ik} pour une propriété positive sur la valeur médiane (BM-M = M-Bm = δ/2) est de (8.333% * δ). Nous obtenons ainsi les figures suivantes.

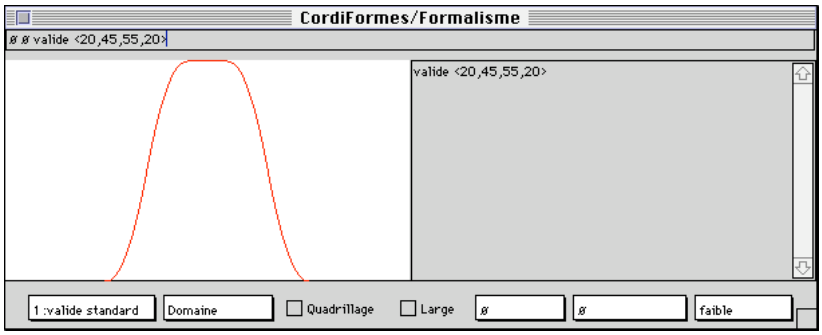

Figure 151. Domaine avec une seule propriété

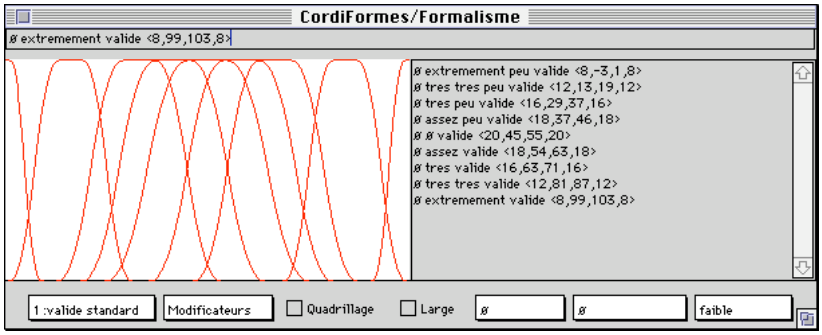

Figure 152. Modificateurs sur la propriété "valide"

Dans certains cas, cette unique fonction peut-être définie comme une propriété de base positive ou négative. Nous retrouverons alors les définitions et les coefficients présentés précédemment. Dans ce cas, la propriété sera alors forcément linguistiquement asymétrique afin de parcourir tout le domaine. La Figure 154 et la Figure 155 illustrent ce cas.

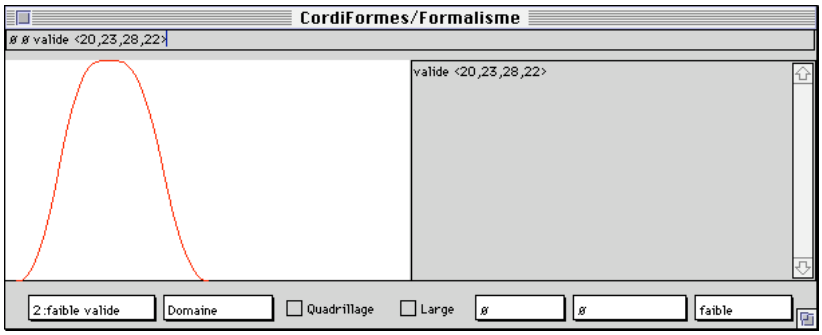

Figure 153. « valide » comme « faible »

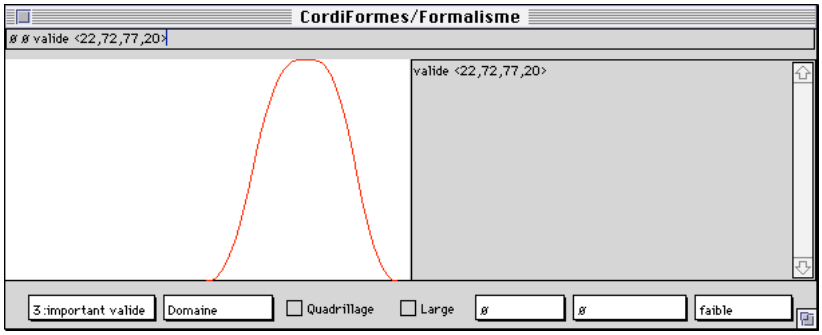

Figure 154. « valide » comme « important »

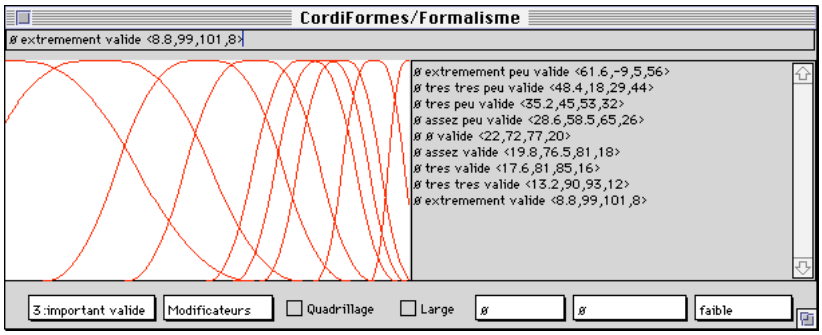

Figure 155. Modificateurs sur « valide » comme « important »

2. Propriétés paramétrées : comparaisons élémentaires

En posant des hypothèses et des contraintes similaires à celles utilisées pour calculer automatiquement les propriétés de base d'un domaine, nous pouvons proposer un calcul automatique de ces propriétés paramétrées.

2.1. Forme et place des fonctions d'appartenance

La forme des propriétés dépend directement de la taille du pseudo-domaine. Soit D_i=[Bm, BM]$_u$ le domaine, et D'_i=[Bm,U]$_u$ et D''_i=[U, BM]$_u$ les pseudo-domaines engendrés par la valeur de référence U. On a alors δ la taille du domaine telle que δ = BM-Bm et δ' et δ'' les tailles des pseudo-domaines telles que δ' = U-Bm et δ'' = BM-U. Les critères de construction des propriétés de base sont :

- la fonction d'appartenance « supérieur à » est dans la première moitié du domaine D''_i ;
- la fonction d'appartenance « inférieur à » est dans la seconde moitié du domaine D'_i ;
- la valeur (b+β) pour la propriété « inférieur à » et (a-α) pour la propriété « supérieur à » se confondent et sont égales à U ;
- les propriétés « inférieur à » et « supérieur à » ont une géométrie légèrement asymétrique.

Compte-tenu du fait que les propriétés sont construites par une forte référence à U, il semble naturel que les fonctions d'appartenances soient géométriquement asymétriques. Pour accentuer le phénomène, nous utiliserons des fonctions L et R différentes. En particulier, la plus proche de U sera très bombée (degrés d'appartenance très vite importants) en utilisant L_2 ou R_2. L'autre fonction sera standard. Nous aurons alors, selon les fonctions de l'annexe 2, L_2-R_4 pour « supérieur à U » et L_4-R_2 pour « inférieur à U ». Nous proposons alors de définir les fonctions d'appartenance des propriétés de bases comme indiquées dans le Tableau 5.

Tableau 23. Définition automatique des fonctions d'appartenance pour les propriétés de comparaison élémentaires

nom	α	a	b	β
supérieur à U	22%*δ''	U+25%*δ''-3%*δ''	U+25%*δ''+2%*δ''	23%*δ''
inférieur à U	23%*δ'	Bm+75%*δ'-2%*δ'	Bm+75%*δ'+3%*δ'	22%*δ'

2.2. Détermination des coefficients de ces propriétés de base

Nous allons reprendre les mêmes contraintes que celles posées pour déterminer les propriétés de base du concept vues au paragraphe 8.2. La différence réside dans la définition des bornes du domaine. Nous allons poser les contraintes suivantes :

1. lorsque l'on applique le modificateur « extrêmement », la fonction d'appartenance résultante doit être telle que U soit dans le noyau pour les deux propriétés ;
2. lorsque l'on applique le modificateur « extrêmement peu », la fonction d'appartenance résultante doit être telle que Bm (resp. BM) soit dans le noyau pour la propriété « inférieure à U » (resp. « supérieure à U »).

Remarque : ces contraintes et ces hypothèses permettent d'assurer, pour chaque valeur du domaine, un degré d'appartenance fort (souvent très proche ou égal à 1) à au moins une propriété issue de chaque propriété de base du concept. De plus, elles assurent que, pour les modificateurs les plus extrêmes, les bornes ont un degré d'appartenance maximum.

2.2.1 Détermination du coefficient de translation élémentaire

La contrainte 1 nous permet de calculer un τ_{ik} correct. Nous aurons pour la propriété « inférieure à U » : $b'_{ik} \geq U$ et $a'_{ik} \leq U$

d'où $\tau_{ik} \geq \dfrac{U - b_{ik} + |k| * 5\% * (b_{ik} - a_{ik})}{k}$ et $\tau_{ik} \leq \dfrac{U - a_{ik} - |k| * 5\% * (b_{ik} - a_{ik})}{k}$

soit avec b_{ik} = Bm + 75% * δ' + 3% * δ' et a_{ik} = Bm + 75% * δ' - 2% * δ' :

$$\boxed{\frac{\delta'}{k}\left(\frac{22 + |k| * .25}{100}\right) \leq \tau_{ik} \leq \frac{\delta'}{k}\left(\frac{27 - |k| * .25}{100}\right)}$$

D'où, comme $|k| < 8$, nous avons : $\boxed{\tau_{ik} = \frac{\delta}{k}\left(\frac{24}{100}\right)}$.

Donc, si $k = k_{\text{extrêmement peu}} = -6$, $\boxed{\tau_{ik} = -\frac{\delta'}{6}\left(\frac{24}{100}\right) = -4\% * \delta'}$

Le coefficient de translation élémentaire τ_{ik} pour une propriété positive à -4%* δ' semble donc une valeur correcte. La propriété « supérieure à U » étant construite de manière symétri-

que, nous aurons alors comme coefficient de translation élémentaire $\tau_{ik} = 4\% * \delta''$. Notons que le coefficient de translation élémentaire est proportionnel à la taille du pseudo-domaine.

2.2.2 Détermination du coefficient d'asymétrie

La contrainte 2 nous permet de calculer un γ_{ik} correct. Nous aurons pour la propriété « inférieure à U » : $b'_{ik} \leq Bm$ et $a'_{ik} \geq Bm$

soit $|\gamma_{ik}| \leq \dfrac{Bm - a_{ik}}{k * \tau_{ik} - |k|(b_{ik} - a_{ik}) * 5\%}$ et $|\gamma_{ik}| \geq \dfrac{Bm - b_{ik}}{k * \tau_{ik} + |k|(b_{ik} - a_{ik}) * 5\%}$

donc : $\boxed{\dfrac{-73\% * \delta'}{k * \tau_{ik} + 0.25\% * \delta' * |k|} \leq |\gamma_{ik}| \leq \dfrac{-78\% * \delta'}{k * \tau_{ik} - 0.25\% * \delta' * |k|}}$

d'où, si on pose $\tau_{ik} = p\% * \delta'$, on a : $\boxed{\dfrac{-73}{k * p + 0.25 * |k|} \leq |\gamma_{ik}| \leq \dfrac{-78}{k * p - 0.25 * |k|}}$

d'où, avec $k=k_{extrêmement}=6$ et $p=-4$, on a : $\boxed{\dfrac{73}{25.5} \leq |\gamma_{ik}| \leq \dfrac{78}{22.5}}$.

Soit, par approximation : $2{,}863 \leq |\gamma_{ik}| \leq 3{,}466$. Un coefficient d'asymétrie γ_{ik} pour une propriété à 3 semble donc une valeur correcte. Comme la propriété « supérieure à U » est construite de la même manière, elle aura le même coefficient d'asymétrie.

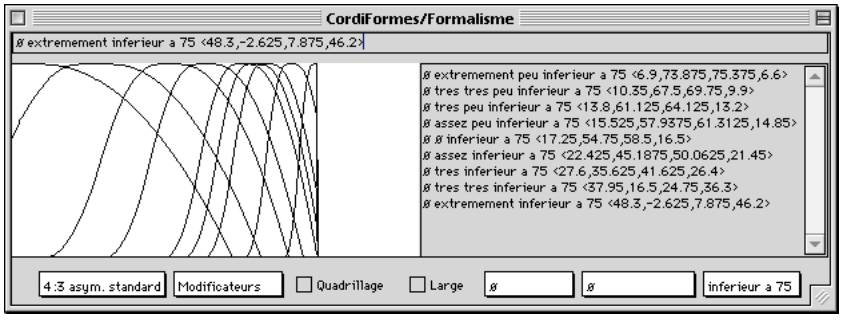

Figure 156. Modificateurs sur la propriété « inférieur à 75 »

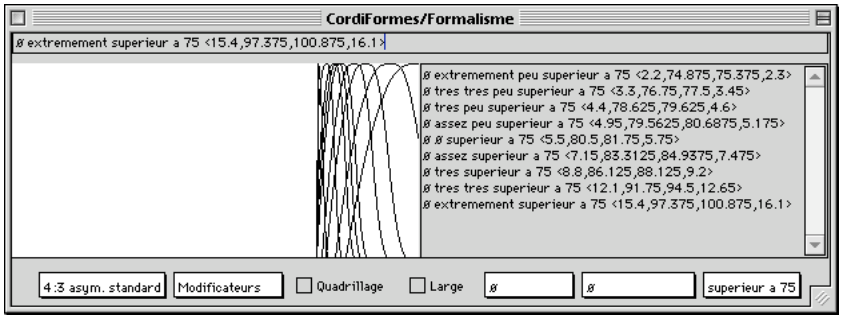

Figure 157. Modificateurs sur la propriété « supérieur à 75 »

ANNEXE 4 : EXEMPLE DE CONCEPT DE LA BASE DE CONNAISSANCES - LA COULEUR

```
package PlateForme.Bibliotheque;

import PlateForme.Concept.CConceptSimple;
import PlateForme.Concept.CConceptTerminal;
import PlateForme.Concept.CConcept;
import PlateForme.Domaine.CDomaine;
import PlateForme.Contrainte.CContrainteMesure;
import PlateForme.Propriete.CProprieteDeBase;
import PlateForme.Fonctions.CFonctionLR;
import PlateForme.Fonctions.CFonctionLRCycle;
import java.awt.Color;

final class CMesureTeinte extends CContrainteMesure {
public CMesureTeinte(CConcept leConcept) { super(leConcept);}

protected Object Calculer() {
  int r = ((Float)Mesure[0]).intValue();
  int g = ((Float)Mesure[1]).intValue();
  int b = ((Float)Mesure[2]).intValue();
  float[] tls = Color.RGBtoHSB(r,g,b,null);
  return new Float(tls[0]*360);
}}

final class CMesureLuminosite extends CContrainteMesure {
public CMesureLuminosite(CConcept leConcept) { super(leConcept);}

protected Object Calculer() {
  int r = ((Float)Mesure[0]).intValue();
  int g = ((Float)Mesure[1]).intValue();
  int b = ((Float)Mesure[2]).intValue();
  float[] tls = Color.RGBtoHSB(r,g,b,null);
  return new Float(tls[2]);
}}

final class CMesureSaturation extends CContrainteMesure {
public CMesureSaturation(CConcept leConcept) { super(leConcept);}

protected Object Calculer() {
  int r = ((Float)Mesure[0]).intValue();
  int g = ((Float)Mesure[1]).intValue();
  int b = ((Float)Mesure[2]).intValue();
  float[] tls = Color.RGBtoHSB(r,g,b,null);
  return new Float(tls[1]);
}}

final class CMesureRouge extends CContrainteMesure {
public CMesureRouge(CConcept leConcept) { super(leConcept);}

protected Object Calculer() {
  float t = ((Float)Mesure[0]).floatValue()/360f;
```

```
  float l = ((Float)Mesure[1]).floatValue();
  float s = ((Float)Mesure[2]).floatValue();
  return new Float(Color.getHSBColor(t,s,l).getRed());
}}

final class CMesureVert extends CContrainteMesure {
public CMesureVert(CConcept leConcept) { super(leConcept);}

protected Object Calculer() {
  float t = ((Float)Mesure[0]).floatValue()/360f;
  float l = ((Float)Mesure[1]).floatValue();
  float s = ((Float)Mesure[2]).floatValue();
  return new Float(Color.getHSBColor(t,s,l).getGreen());
}}

final class CMesureBleu extends CContrainteMesure {
public CMesureBleu(CConcept leConcept) { super(leConcept);}

protected Object Calculer() {
  float t = ((Float)Mesure[0]).floatValue()/360f;
  float l = ((Float)Mesure[1]).floatValue();
  float s = ((Float)Mesure[2]).floatValue();
  return new Float(Color.getHSBColor(t,s,l).getBlue());
}}

public class CConceptCouleur extends CConceptSimple {
public static final int ModeListe = 0;
public static final int ModeRVB = 1;
public static final int ModeTLS = 2;

  //Modele TLS
public CConceptTerminal Teinte;
public CConceptTerminal Luminosite;
public CConceptTerminal Saturation;

  //Modele RVB
public CConceptTerminal Rouge;
public CConceptTerminal Vert;
public CConceptTerminal Bleu;

  //Modele choisi
  int Mode;

public CConceptCouleur() { this(null);}

public CConceptCouleur(CConcept Pere) { this(Pere, ModeTLS);}

public CConceptCouleur(CConcept Pere, int Mode) { this(Pere, Mode, 0);}

public CConceptCouleur(CConcept Pere, int Mode, int sup) {
  super(Pere, "couleur", null,6+sup);
  this.Mode = Mode;

  CDomaine dTeinte=new CDomaine(0,360,0.1f);
  int delta = 30;
  Teinte = new CConceptTerminal(this,"teinte", dTeinte,
                                CConceptTerminal.TYPE_INCONNU, false, null);
  CProprieteDeBase pJaune = new CProprieteDeBase(dTeinte,
```

```
                    new CFonctionLRCycle(delta,60,60,delta,dTeinte,360),
                    "teinte jaune");
pJaune.maFonction.SetCouleur(new Color(0f,0f,0f));
pJaune.SetNom("jaune");
Teinte.AddProprieteDeBase(pJaune);
CProprieteDeBase pVert = new CProprieteDeBase(dTeinte,
                  new CFonctionLRCycle(delta,120,120,delta,dTeinte,360),
                  "teinte vert");
pVert.maFonction.SetCouleur(new Color(0f,0f,0f));
pVert.SetNom("vert(e)");
Teinte.AddProprieteDeBase(pVert);
CProprieteDeBase pCyan = new CProprieteDeBase(dTeinte,
                  new CFonctionLRCycle(delta,180,180,delta,dTeinte,360),
                  "teinte cyan");
pCyan.maFonction.SetCouleur(new Color(0f,0f,0f));
pCyan.SetNom("cyan");
Teinte.AddProprieteDeBase(pCyan);
CProprieteDeBase pBleu = new CProprieteDeBase(dTeinte,
                  new CFonctionLRCycle(delta,240,240,delta,dTeinte,360),
                  "teinte bleu");
pBleu.maFonction.SetCouleur(new Color(0f,0f,0f));
pBleu.SetNom("bleu(e)");
Teinte.AddProprieteDeBase(pBleu);
CProprieteDeBase pMagenta = new CProprieteDeBase(dTeinte,
                  new CFonctionLRCycle(delta,300,300,delta,dTeinte,360),
                  "teinte magenta");
pMagenta.maFonction.SetCouleur(new Color(0f,0f,0f));
pMagenta.SetNom("magenta");
Teinte.AddProprieteDeBase(pMagenta);
CProprieteDeBase pRouge = new CProprieteDeBase(dTeinte,
                     new CFonctionLRCycle(delta,0,0,delta,dTeinte,360),
                     "teinte rouge");
pRouge.maFonction.SetCouleur(new Color(0f,0f,0f));
pRouge.SetNom("rouge");
Teinte.AddProprieteDeBase(pRouge);

AjouterConcept(Teinte,0,Mode == ModeTLS);

CProprieteDeBase pdb;

Luminosite = new CConceptTerminal(this,"luminosite",
                                  new CDomaine(0,1,0.001f),null);
pdb = Luminosite.GetProprieteDeBase("important(e)");
pdb.SetNom("clair(e)");
pdb = Luminosite.GetProprieteDeBase("faible");
pdb.SetNom("sombre");
AjouterConcept(Luminosite,1,Mode == ModeTLS);

Saturation = new CConceptTerminal(this,"saturation",
                                  new CDomaine(0,1,0.001f),null);
pdb = Saturation.GetProprieteDeBase("important(e)");
pdb.SetNom("pur(e)");
pdb = Saturation.GetProprieteDeBase("faible");
pdb.SetNom("pastel(le)");
AjouterConcept(Saturation,2,Mode == ModeTLS);

Rouge = new CConceptTerminal(this,"rouge", new CDomaine(0,255,1f),null);
AjouterConcept(Rouge,3,Mode == ModeRVB);
```

```
  Vert = new CConceptTerminal(this,"vert",new CDomaine(0,255,1f),null);
  AjouterConcept(Vert,4,Mode == ModeRVB);

  Bleu = new CConceptTerminal(this,"bleu",new CDomaine(0,255,1f),null);
  AjouterConcept(Bleu,5,Mode == ModeRVB);

  if (Mode == ModeListe) {
   AjouterObjet(new CCouleur("vert sombre",
                (float)0.13725,(float)0.556863,(float)0.137255));
    AjouterObjet(new CCouleur("rouge brique",
                (float)0.556863,(float)0.137255,(float)0.137255));
    AjouterObjet(new CCouleur("or",
                (float)0.8,(float)0.498039,(float)0.196078));
    AjouterObjet(new CCouleur("gris",(float)0.5,(float)0.5,(float)0.5));
   maTacheGeneratrice = "PlateForme.Tache.TacheAleatoire";
  } else if (Mode == ModeTLS) {
    CMesureRouge MesureRouge = new CMesureRouge(Rouge);
    MesureRouge.AjouterReference(Teinte);
    MesureRouge.AjouterReference(Luminosite);
    MesureRouge.AjouterReference(Saturation);
    Rouge.AjouterContrainteMesure(MesureRouge);

    CMesureVert MesureVert = new CMesureVert(Vert);
    MesureVert.AjouterReference(Teinte);
    MesureVert.AjouterReference(Luminosite);
    MesureVert.AjouterReference(Saturation);
    Vert.AjouterContrainteMesure(MesureVert);

    CMesureBleu MesureBleu = new CMesureBleu(Bleu);
    MesureBleu.AjouterReference(Teinte);
    MesureBleu.AjouterReference(Luminosite);
    MesureBleu.AjouterReference(Saturation);
    Bleu.AjouterContrainteMesure(MesureBleu);
  } else if (Mode == ModeRVB) {
    CMesureTeinte MesureTeinte = new CMesureTeinte(Teinte);
    MesureTeinte.AjouterReference(Rouge);
    MesureTeinte.AjouterReference(Vert);
    MesureTeinte.AjouterReference(Bleu);
    Teinte.AjouterContrainteMesure(MesureTeinte);

   CMesureLuminosite MesureLuminosite = new CMesureLuminosite(Luminosite);
   MesureLuminosite.AjouterReference(Rouge);
   MesureLuminosite.AjouterReference(Vert);
   MesureLuminosite.AjouterReference(Bleu);
   Luminosite.AjouterContrainteMesure(MesureLuminosite);

   CMesureSaturation MesureSaturation = new CMesureSaturation(Saturation);
   MesureSaturation.AjouterReference(Rouge);
   MesureSaturation.AjouterReference(Vert);
   MesureSaturation.AjouterReference(Bleu);
   Saturation.AjouterContrainteMesure(MesureSaturation);
  }
}

public String toString() {
  String s = "Nom :"+Nom+
    "\n\t Domaine courant :"+leDomaine+
```

```
    "\n\tMesure :"+Mesure+ "\tTache :"+maTacheGeneratrice;
  for(int i=0;(i<ConceptsComposants.length);i++)
    s = s + "\n\t--" + ConceptsComposants[i];
  return s;
}

  //Construit la mesure du concept a partir des concepts composants
public Object Construire() {
  CCouleur couleur;
  if (Mode == ModeRVB) {
   couleur = new CCouleur("",((Float)Rouge.GetMesure()).intValue(),
                             ((Float)Vert.GetMesure()).intValue(),
                             ((Float)Bleu.GetMesure()).intValue());
  } else /*if (Mode == ModeTLS)*/ {
   float hsb[] = new float[3];
    hsb[0] = ((Float)Teinte.GetMesure()).floatValue()/360f;
    hsb[2] = ((Float)Luminosite.GetMesure()).floatValue();
    hsb[1] = ((Float)Saturation.GetMesure()).floatValue();
   couleur = new CCouleur("",hsb);
  }
  return couleur;
}

public Object ModeleGeometrique() {
  return ((CCouleur)Mesure).rgb;
}

public String ModeleGeometriquePOV() {
  String Final;
  Color c = ((CCouleur)Mesure).rgb;
  Final = "pigment {"+ c.getRed() + ", " +
          c.getGreen() + ", " + c.getBlue() +"}\r";
  return Final;
}}
```

ANNEXE 5 : DESCRIPTIONS AVEC CHROMOFORMES

1. Premier exemple de description

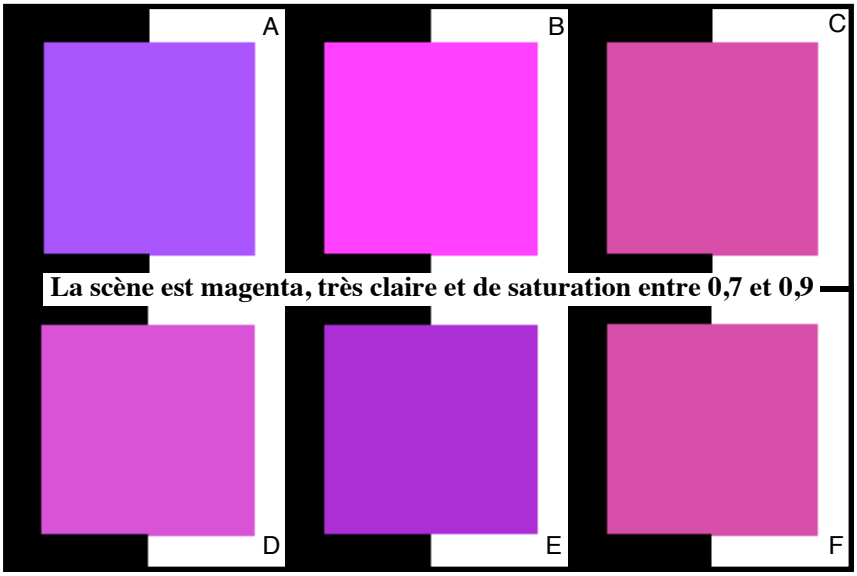

Figure 158. Solutions possibles à une description dans ChromoFormes (1)

2. Deuxième exemple de description

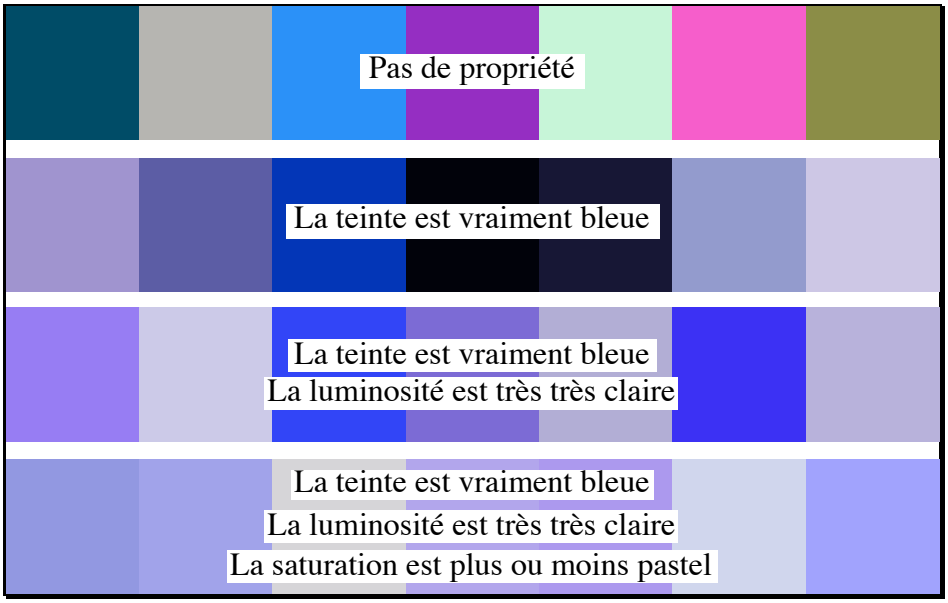

Figure 159. Solutions possibles à une description dans ChromoFormes (2)

RÉSUMÉ

Le projet CordiFormes : une plate-forme pour la construction de modeleurs déclaratifs

Les travaux en modélisation déclarative sont assez nombreux. Il est désormais nécessaire de mettre en place des méthodes générales basées sur celles déjà étudiées et de proposer des outils pour le développement de modeleurs déclaratifs.

Nous proposons un nouveau formalisme basé sur les ensembles flous. Ce formalisme apparaît d'une part comme une synthèse et une unification des travaux existants et d'autre part apporte des éléments nouveaux comme la logique floue, la gestion linguistique de la négation… Nous étudions particulièrement le type de propriété le plus simple : la propriété élémentaire. A partir d'une propriété de base, d'un opérateur flou et d'un modificateur, nous mettons en place une méthode pour déterminer la sémantique d'une propriété élémentaire. Nous proposons aussi un traitement original de sa négation. Plutôt que d'utiliser la négation logique habituelle, nous nous intéressons à une gestion se basant sur des notions linguistiques. Enfin, nous présentons des solutions de traitement pour les autres propriétés. Cependant, ces solutions ne sont pas encore vraiment satisfaisantes.

La plupart des projets en modélisation déclarative mettent en oeuvre des techniques similaires. A partir de ces travaux et du formalisme flou, nous développons le projet CordiFormes, une plate-forme de programmation visant à faciliter la mise en œuvre de futurs modeleurs déclaratifs. Ses caractéristiques sont la simplicité, la souplesse de programmation, l'efficacité, l'extensibilité, la réutilisabilité et le prototypage rapide du modeleur. CordiFormes propose des outils sur trois niveaux : le noyau comportant toutes les structures et algorithmes de base, la couche interface proposant dialogues et composants d'interface et, enfin, la couche application pour produire rapidement un premier modeleur.

Trois applications permettent de valider le formalisme et les outils de la plate-forme.

Mots clés : modélisation déclarative, synthèse d'images, ensembles flous, négation linguistique, plate-forme, propriété, relation, similarité

ABSTRACT

The CordiFormes project : a platform to build declarative modelers

Since 1989, numerous works about declarative modelling have been done. Now, it Currently, it is necessary to make general methods based on those previous studies evolve. New tools to help in building declarative modelers must be proposed.

We propose a new formalism based on the fuzzy sets. This formalism is a synthesis and unification of existent works. It also brings new elements as fuzzy logic, linguistic management of the negation, etc. We have particularly studied the simplest type of property : the elementary property. Using the notion basic properties, fuzzy operators and modifiers, we describe a method to processing the semantic of an elementary property. We propose also an original processing of its negation. Rather than to use the usual logical negation, we have been interested in a management based on linguistic notions. Finally, we present solutions of processing for the other properties. However, these solutions are not truly satisfactory.

Most of projects in declarative modeling implement similar techniques. From these works and the fuzzy formalism, we have developed the CordiFormes project, a programming platform and a set of tools to help the creation of future declarative modelers. Its characteristics are the simplicity, the flexibility of programming, the efficiency, the extendibility, the reusability and the quick modeler's prototyping. CordiFormes proposes three levels of tools : the kernel level containing all basic structures and all basic algorithms, the interface level proposing dialogues and interface components and finally the application level to produce rapidly a first basic modeler.

Three applications have allowed to validate the formalism and tools of the platform.

Keywords : declarative modelling, image synthesis, fuzzy sets, linguistic negation, platform, property, relation, similarity