Une plate-forme pour la construction de modeleurs déclaratifs

—

Rapport de Recherche IRIN-167

—

Emmanuel DESMONTILS & Jean-Yves MARTIN

IRIN - Université de Nantes & École Centrale de Nantes
2, rue de la Houssinière
BP 92208
F-44322 Nantes, Cedex 03

{Jean-Yves.Martin,Emmanuel.Desmontils}@irin.univ-nantes.fr

janvier 1998

Résumé

Le grand nombre de projets en modélisation déclarative nous a incités à proposer une plate-forme de développement pour une mise en œuvre simple et rapide des prochains modeleurs déclaratifs. La programmation orientée objet nous propose des outils tout à fait adaptés à la situation actuelle et aux caractéristiques des techniques et des connaissances dans notre domaine. Grâce à ces outils, nous proposons une plate-forme, appelée CordiFormes, pour assister le concepteur dans sa construction de modeleur déclaratif. Ses principales caractéristiques sont, outre ceux cités précédemment, la souplesse, l'efficacité, l'extensibilité et la réutilisabilité.

Mots clef : modélisation déclarative, plate–forme, programmation orientée objet, ontologie, synthèse d'images, CAO

Table des matières

Table des figures

Liste des tableaux

1 Introduction

La *modélisation déclarative* en synthèse d'images [1, 2], appelée aussi *cooperative computer aided design* [3] ou *generative computer aided design* [4] a donné naissance à de nombreux travaux. Parmi ces travaux, citons le contrôle spatial (projet VoluFormes [5] et NALIG [6]), la modélisation de sites mégalithiques (projet MégaFormes [7]), la modélisation d'automates cellulaires (AutoFormes [8]), la modélisation de logements (projets «LaHave House» [9], BatiMan [10] et FLATS [3]) ou de fenêtres (GENWIN [11]), la modélisation de polyèdres (PolyFormes [12]), etc.

Ces projets nous ont incités à proposer une plate-forme de développement pour la mise en œuvre des prochains modeleurs déclaratifs. Nous avons en effet constaté que la plupart d'entre-eux proposent des structures et des algorithmes de même nature. L'utilisation d'une telle plate-forme peut alors se révéler d'un grand intérêt tant au point de vue du temps de développement que de l'optimisation des méthodes développées. Cette plate-forme se veut donc un outil d'assistance au développement de modeleurs déclaratifs.

Après avoir présenté les principes généraux de la modélisation déclarative, nous étudierons les caractéristiques de notre plate-forme. Nous montrerons comment la programmation orientée objet permet de répondre aux spécificités de cette plate-forme. En particulier, nous présenterons la structure des connaissances et le modèle de génération de scènes choisi. Nous montrerons la facilité de la mise en œuvre d'applications et illustrerons nos propos par deux exemples.

2 La 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.

La figure 1 illustre cette différence. 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.[13]

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

1. conception, détermination des actions élémentaires et vérification à la charge de l'opérateur ;

2. calcul et visualisation à la charge du système.

L'utilisateur du modeleur doit, à partir de son idée, déterminer les spécifications de son problème. Il imagine alors un «objet mental» pouvant y répondre. A l'aide du modeleur, et en suivant les instructions élémentaires permises par ce dernier, il construit cet objet en machine. Le modeleur lui permet de le visualiser au fur et à mesure de sa construction. Ensuite, un ensemble de tests permet de valider l'objet ainsi engendré. S'ils sont négatifs, l'utilisateur doit revoir son objet mental et modifier les instructions entrées. Le processus de construction recommence jusqu'à obtention d'une solution satisfaisante. S'il n'en trouve pas, l'utilisateur doit reprendre ses spécifications et tout le processus de conception.

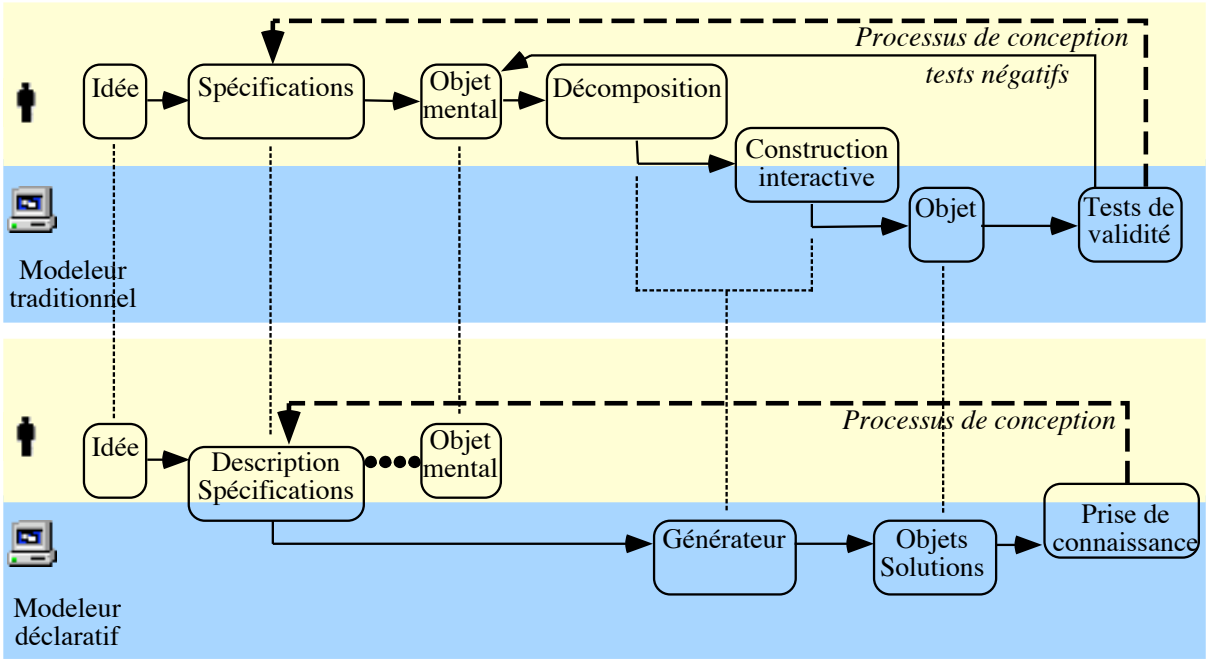

FIG. 1 – *Comparaison de modeleurs traditionnels et déclaratifs*

Le processus de conception à l'aide d'un modeleur traditionnel laisse donc la plus grande part du travail au concepteur. Le système est, en fait, chargé de vérifier le modèle mental du concepteur.

Un modeleur déclaratif s'appuie 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 [14].*» Le processus de conception est donc composé de trois étapes essentielles[1] : la description du problème et de la solution désirée, la génération de l'ensemble des solutions valides (répondant à la description) et la prise de connaissance de ces solutions. 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 peut modifier les propriétés fournies et relancer une exploration.

Beaucoup de travaux décrivent des outils d'interaction de haut niveau en CAO [16, 17, 9, 18, 19] mais rarement dans les 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 qui sont principalement [1, 2] :

- une description souple, de haut niveau, avec des outils adaptés, la plupart du temps sans s'intéresser aux valeurs et aux coefficients effectifs ;
- la génération exhaustive, efficace et contrôlée des solutions ;

1. Ces phases et leurs éléments constitutifs ont été étudiés lors de nombreux travaux. Une étude exhaustive sur les modeleurs déclaratifs est proposée par [15]. Une synthèse du point de vue des utilisateurs ainsi qu'une étude des processus de conception à l'aide de modeleurs déclaratifs sont proposées dans [13].

– la 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 différence importante entre les deux approches est la quantité et le type de travail réalisés par l'homme. Les modeleurs déclaratifs prennent beaucoup plus en charge les tâches de base de la conception laissant l'homme se concentrer sur celles de haut niveau. Ainsi, l'opérateur se trouve libéré des calculs et peut se focaliser davantage sur la phase de création.

3 Le projet CordiFormes

3.1 Objectifs

Un certain nombre de projets ont vu le jour, chacun visant a étudier des aspects particuliers de la modélisation déclarative. Bien que différents, ils mettent en œuvre des techniques similaires. Les travaux récents [15, 13] 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 [13]. Il est 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. Nous nous sommes donc 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 [20].

Pour mettre en place l'unification de ces outils, nous définissons un modèle de représentation des connaissances [21]. Ce modèle 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.

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[2] peut utiliser des méthodes générales et des objets déjà implémentées, 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.

Le concepteur 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. Afin d'éviter toute confusion, nous appellerons *concepteur* une personne construisant un modeleur déclaratif et *utilisateur* une personne utilisant un modeleur déclaratif pour construire les formes qu'il désire.

3.2 Caractéristiques de la plate-forme

Les objectifs que nous nous sommes fixés pour CordiFormes imposent un certain nombre de caractéristiques de la plate-forme. Celles que nous attendons sont entre-autre : la *simplicité*, la *souplesse* de programmation, l'*efficacité*, l'*extensibilité*, la *réutilisabilité*, le *prototypage* rapide du modeleur.

3.2.1 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 pas obligé de s'occuper de la génération ou de la définition des propriétés. CordiFormes utilise pour cela un mode de génération dit récursif. Pour la description, le concepteur bénéficie du calcul automatique des propriétés. Bien évidemment, s'il le désire, le concepteur peut ajuster, modifier ou refaire certains éléments qui ne lui plaisent pas.

3.2.2 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 pour des raisons de performance que nous aborderons plus loin, cette souplesse est importante pour pouvoir ajuster le modeleur à ses propres besoins.

3.2.3 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 de 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.

3.2.4 L'efficacité

L'utilisation d'une plate-forme a un inconvénient : les algorithmes utilisés, en particulier les algorithmes de génération, sont généraux 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 récursive.

3.2.5 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 et d'outils courants qui constituent l'*ontologie* [22] de la modélisation déclarative et de la modélisation géométrique [23]. Cette base permet au concepteur de ne porter sont attention que sur les objets spécifiques du domaine d'application. Il peut éventuellement l'enrichir au fur et à mesure de ses développements.

3.2.6 Outils d'interface et de prototypage

La majeure partie des publications en modélisation déclarative concerne les outils de génération. Toutefois, ces outils 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...).

3.3 Choix du 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 systèmes (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 [24, 25] pour développer CordiFormes. Outre les caractéristiques citées ci-dessus, Java par ses outils d'encapsulation, nous permet d'obliger la définition de méthodes (grâce aux méthodes abstraites), autoriser ou interdire la surcharge de certaines autres (méthodes classiques ou méthodes dites finales). Le langage Java possède toutefois un inconvénient : son efficacité (temps d'exécution). Ce langage, plutôt de la classe des langages interprétés mais utilisant une technique de compilation «Just In Time», c'est-à-dire à l'exécution, a une efficacité moindre qu'un langage compilé. Cependant, il est 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.

3.4 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 2) :

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 qu'il peut enrichir. Cette bibliothèque sera remise à jour par le concepteur qui pourra y introduire les éléments 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.

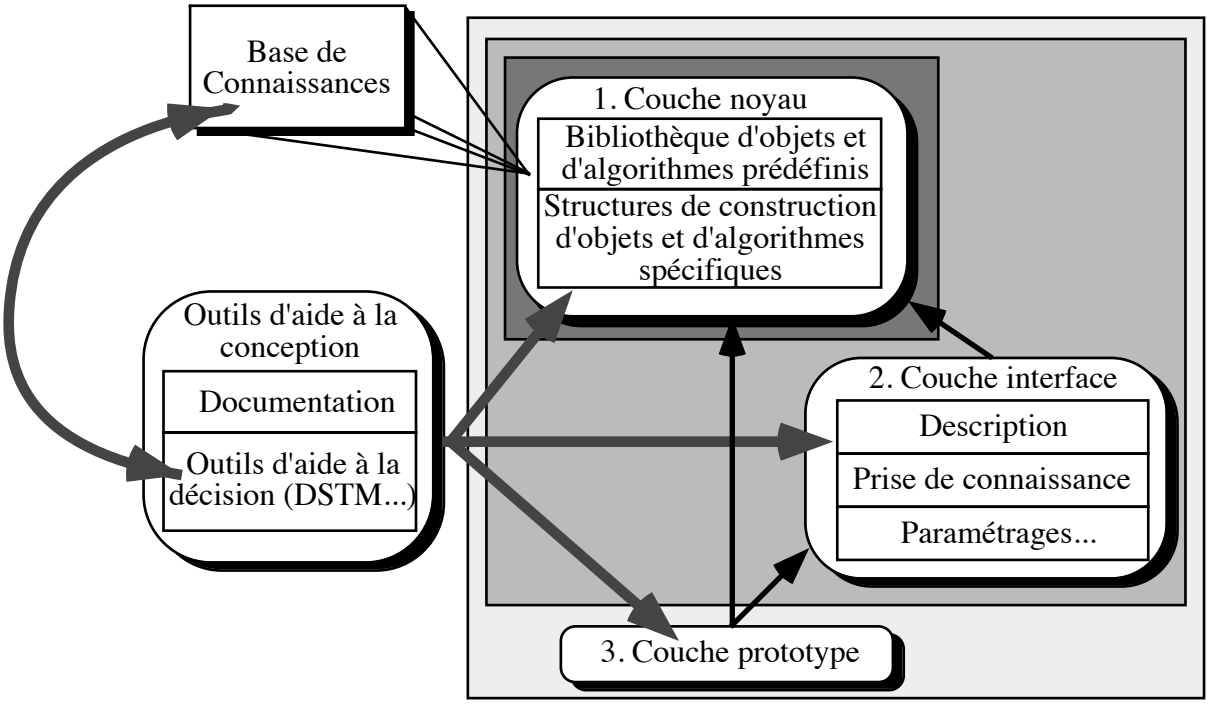

FIG. 2 – *Niveaux d'utilisation et éléments constitutifs de CordiFormes*

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 [26]. 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 sur les choix des techniques à utiliser et les éléments essentiels à programmer. Le but est d'obtenir un système minimal et de produire facilement et rapidement une première maquette de l'application.

4 Structure des connaissances

Après avoir présenté les grandes lignes du projet CordiFormes, nous allons nous intéresser aux éléments constitutifs du noyau. Nous étudierons les éléments les plus importants de ce noyau : les concepts. Ce sont eux qui représentent essentiellement le coeur de la plate-forme, 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 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.

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. Parmi tous ces concepts, nous distinguons les concepts terminaux et les concepts non-terminaux.

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. Nous leur associons un *domaine*, ensemble des valeurs que peut prendre ce concept, noté $[\mathrm{Bm}, \mathrm{BM}]_u$ (où Bm et BM sont les bornes et u l'unité), et un ensemble de *propriétés de base* (comme par exemple : {}, {vérifié}, {faible, moyen, important} ou {petit, moyen, grand}). Ces propriétés de base sont des sous-ensembles flous [27] du domaine. Combinées avec des opérateurs de modification, elles permettent de construire les propriétés de la description [21, 20].

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. Un *concept non-terminal* est un concept défini par un ensemble de concepts appelés *concepts composants*. Ces derniers sont des concepts terminaux ou non-terminaux.

Par exemple, le concept de «couleur», défini selon un modèle RVB, est un concept non-terminal composé des concepts terminaux «rouge», «vert» et «bleu», dont les domaines sont $[0,255]_1$. Dans la figure 3, le concept «maison» (où les concepts terminaux sont en pointillés) est composé d'un «toit» et de «murs». Le «toit» possède une «forme», une «hauteur», une «couleur»... Les «murs» sont définis par une «hauteur», une «texture» (de domaine {pierre, bois, brique})...

Les concepts ne sont pas toujours des objets ou des constituants d'objets. Ils peuvent servir à décrire la construction d'objets ou correspondre à des relations entre au moins deux concepts (par exemple, les concepts supports aux propriétés relatives et plus généralement aux relations).

Nous avons implémenté les concepts sous forme d'une classe dont les principaux attributs sont le nom et la valeur ou *mesure* (objet du domaine d'application qu'il représente). Elle possède également des méthodes de vérification de validité de cette valeur et de production du concept dans un modèle géométrique donné. La classe associée à un concept terminal hérite de la classe concept et comporte deux attributs supplémentaires : le domaine et la liste des propriétés de base associées. La classe associée à un concept non-terminal hérite de la classe concept et comporte une liste de concepts composants

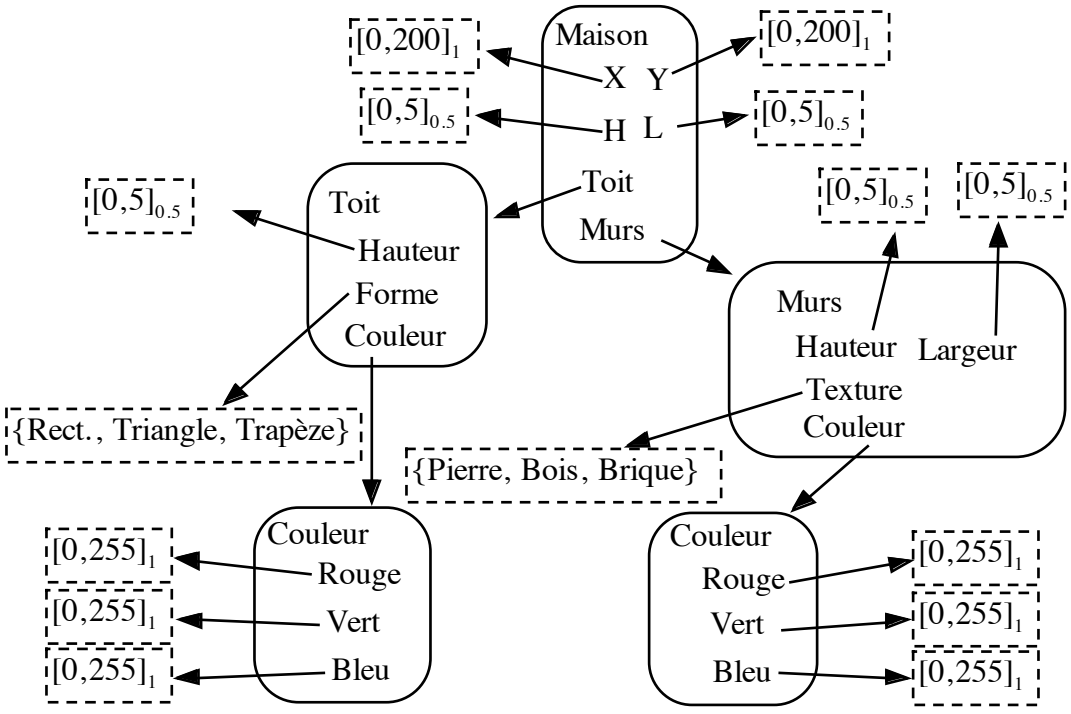

FIG. 3 – *Exemple de concept complexe*

5 Génération des concepts

5.1 Proposition d'une nouvelle classification

Les différents travaux réalisés en modélisation déclarative sont généralement classés suivant *les techniques de génération utilisées* :

- Arbres d'exploration explicites (arbres d'énumération et autres algorithmes de parcours d'arbres) [28, 11, 7] ;
- Arbres d'exploration implicites (moteurs d'inférences) [12, 29] ;
- Grammaires génératives [4, 3, 9] ;
- Systèmes à base de résolution de contraintes [30, 6, 17, 19, 10] et d'arithmétique des intervalles [5] ;
- Calcul de la fonction inverse [16].

Dans le cadre du projet CordiFormes, la classification adoptée repose non pas sur les techniques de génération utilisées mais sur les *objectifs recherchés et les effets escomptés pour l'utilisateur*. Dans ce contexte, seules deux classes de génération sont retenues à savoir l'*énumération* et le *tirage aléatoire sous contraintes*.

Une génération est dite *par énumération* si, étant donné un objet k, 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.

Une génération est dite *aléatoire* si elle n'est pas par énumération, c'est-à-dire s'il n'est pas possible de prédire avec exactitude ce que sera l'objet k+1 connaissant l'objet k.

Toutes les méthodes développées en modélisation déclarative peuvent être réparties selon ces deux classes.

5.2 La génération récursive

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 C_i (déterminer sa mesure) à partir de ses concepts composants générateurs[3] : 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. Cependant, un concept non-terminal peut posséder une méthode de génération qui lui est propre. En conséquence, la génération récursive ne s'appliquera pas à ses composants. Il sera donc considéré, au niveau de la génération, au même titre qu'un concept terminal. Nous appellerons un tel concept un *concept pseudo-terminal*.

Dans ce cadre, la classe de génération d'un concept non-terminal C_i dépend des classes de génération de ces concepts composants. Intuitivement, un concept C_i est généré de façon aléatoire si un 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.

La génération d'un concept terminal ou pseudo-terminal peut être de la classe énumération ou aléatoire. Dans les deux cas, 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 pour la seconde (lois de probabilité utilisées).

5.3 Tâches de génération

Suite aux principes définis précédemment, on associe à chaque concept non-terminal, terminal ou pseudo-terminal du point vue de la génération une *tâche de génération*[4]. Ces tâches sont chargées de mettre en œuvre l'algorithme de génération récursive. Elles sont liées entre elles selon la hiérarchie des concepts. Les tâches sont implémentées sous forme de classe.

La racine de la hiérarchie des concepts correspond généralement à un objet de la scène décrite par l'utilisateur. En fonction de sa description et de l'implémentation du modeleur, ce dernier sera amené à construire plusieurs objets correspondant à des phases de la conception (objets intermédiaires) ou a des objets terminaux. La génération de ces objets dépend du mode de conception choisi par le concepteur.

Chacun de ces objets sera généré de manière indépendante par une hiérarchie de tâches. L'ordre de génération dépend du mode de conception utilisé. Dans le cadre de cet article,

3. Tous les concepts composants d'un concept non-terminal C_i ne sont pas nécessairement indispensables à la génération de C_i. Un concept composant participant à la construction d'un concept est appelé *concept générateur*. Un *ensemble générateur*est un ensemble de concepts générateurs permettant de construire le concept Un concept non-terminal peut posséder plusieurs ensembles générateurs. Dans ce cadre, le concepteur doit choisir l'ensemble le plus efficace en termes de rapidité et contrôle des solutions. Dans cet article, nous considérons que tout concept C_i possède un seul groupe de concepts générateurs composé de tous les concepts composants de C_i.

4. Dans le cadre d'une programmation parallèle, ces tâches sont éventuellement situées sur des processeurs différents

nous considérerons uniquement le mode de conception très primaire consistant à produire tous les objets en même temps et intégralement. D'autres modes de conception beaucoup plus élaborés existent [13]. Nous ne les aborderons pas dans le cadre de ce travail, mais CordiFormes en propose certains.

5.4 Contraintes

5.4.1 définition

Les méthodes de génération proposées dans les paragraphes précédents supposent une indépendance totale entre les différents concepts générateurs d'un concept non-terminal. 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.

Une *contrainte* est une fonction portant sur un *concept cible* et dont les paramètres sont des concepts appelés *concepts de référence* (figure 4).

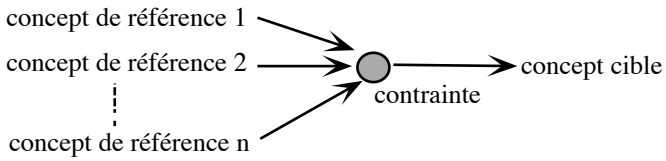

FIG. 4 – *Définition d'une contrainte*

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.

5.4.2 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 contraintes : initialisation, optimisation et mesure.

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.

Au cours de la construction d'un concept non-terminal, certaines combinaisons de concepts composants ne sont pas autorisées. De plus, connaissant les mesures 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*. Elles ne modifient pas l'univers des solutions à explorer mais le réduisent provisoirement afin d'éviter de construire des scènes qui seront certainement rejetées.

Jusqu'ici, nous n'avons considéré les concepts composants que 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 certains 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. Nous la considérerons comme une contrainte prenant en référence un ou plusieurs concepts de la scène et portant sur la mesure du concept cible. La *contrainte de mesure* joue le rôle de fonction de mesure pour un concept donné.

La figure 5 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».

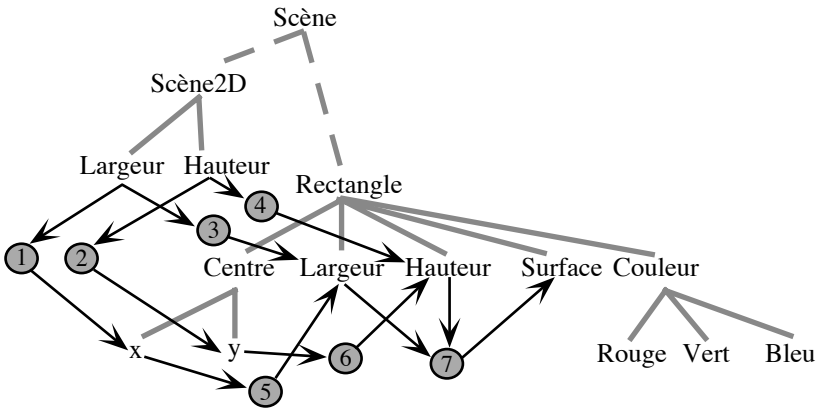

FIG. 5 – *Exemple de contraintes*

5.4.3 Optimisations

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 [31, 5, 32]. En effet, les domaines sont des intervalles possédant une unité. Par conséquent, il est possible d'appliquer cette arithmétique qui 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).

Cette notion de contrainte amène naturellement à s'intéresser aux techniques liées aux CSP[5][33]. En particulier, il est possible d'établir un ordre de génération des concepts générateurs en fonction de leurs liens par les contraintes. La méthode de *Largeur Minimale*[6] développée par des techniques de «lookhead - forward checking» semblent très prometteuses. Toutefois, ces techniques ne sont pas complètement adaptées à notre problème, et une étude plus poussée est nécessaire.

5. Constraint Satisfaction Problem
6. Minimal Width Ordering Heuristic

6 Implémentation pour un domaine d'application

6.1 Les concepts de l'application

Pour créer un concept terminal spécifique, il suffit de renseigner les champs le définissant, c'est-à-dire les valeurs du domaine (bornes et unité ou liste explicite des valeurs) ainsi que les propriétés de base (nombre et termes utilisés). Par exemple, pour définir un concept de «teinte» (dans le modèle de couleur TLS[7]), il suffit de proposer les bornes et l'unité du domaine ($[0,360]_1$) et les propriétés de base {rouge, jaune, vert, cyan, bleu, magenta}.

Pour créer un concept non-terminal, trois méthodes sont possibles : la création directe, la sélection dans la bibliothèque ou la modification d'un concept existant. Pour créer un concept spécifique au domaine d'application, le concepteur doit :

- éventuellement déterminer le concept de la bibliothèque dont il hérite ;
- renseigner ou modifier la liste des concepts composants ;
- déterminer l'ensemble des concepts générateurs ;
- définir les contraintes d'initialisation et d'optimisation ;
- pour les concepts non-générateurs, déterminer les contraintes de mesure ;
- surcharger la méthode permettant de déterminer la mesure de ce concept à partir des concepts composants ;
- surcharger la méthode de production du modèle géométrique éventuellement associé à ce concept ;
- mettre en place des options spécifiques à l'attention du concepteur.

Une des opérations importantes dans la définition d'un nouveau concept est la construction des contraintes utilisées. Pour cela, il suffit d'une part de renseigner la liste des concepts de référence et le concept cible, et d'autre part de surcharger la méthode associée à la fonction d'évaluation.

Par exemple, s'il désire construire un concept «couleur» selon le modèle RVB[8], il créera une sous-classe d'un concept non-terminal. Pour cela, il ajoutera dans la liste des concepts composants générateurs, les instances des concepts terminaux «rouge», «vert» et «bleu» définis sur des domaines de la forme $[0, 255]_1$. De plus, pour permettre des descriptions selon le modèle TLS, il ajoutera les concepts composants non-générateurs «Teinte» (que nous avons déjà définie), «Luminance» (de domaine $[0,100]_1$ et de propriétés de base {sombre, moyen, clair}) et «Saturation» (de domaine $[0,100]_1$ et de propriétés de bases {pastel, moyen, pur}). Il devra aussi ajouter trois contraintes de mesure permettant de déterminer la mesure de ces concepts à partir de celles de l'ensemble générateur.

7. Teinte-Luminance-Saturation
8. Rouge-Vert-Bleu

6.2 Les tâches de génération spécifiques

Suite à nos remarques, sur l'efficacité en particulier, le concepteur a la possibilité de proposer une méthode de génération spécifique pour un concept donné. Pour mettre en place cette méthode, il suffit de créer une nouvelle tâche de génération héritant de la tâche standard en surchargeant une méthode d'initialisation et une méthode de parcours. De plus, il devra renseigner un champ du concept indiquant la nouvelle méthode de génération privilégiée. Ce concept deviendra alors pseudo-terminal pour la génération.

7 Exemples d'applications

Dans cette section, nous allons présenter deux exemples d'application que nous avons développées à l'aide de CordiFormes. Ces exemples sont basiques mais illustrent bien la simplicité et la facilité de conception à l'aide de notre plate-forme.

7.1 ChromoFormes

Habituellement, la détermination d'une couleur n'est pas toujours facile [34, 35, 36]. 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, TLS...). 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. 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.

Le modèle «le plus intuitif et le plus proche de la perception naturelle des couleurs» [36] est incontestablement le modèle TLS (Figure 6). 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...).

Les concepts pouvant faire l'objet d'une description sur une couleur sont donnés par le tableau 1. Selon ce que désire le concepteur, les concepts générateurs sont {Teinte, Saturation, Luminosité} (mode TLS), {Rouge, Vert, Bleu} (mode RVB) ou la liste de couleurs {vert sombre ; rouge brique ; or ; ...} (mode Liste).

Du point de vue de la 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'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[9] sont : (r=174 ;

9. Notons que les solutions sont présentées de manière graphique mais, pour des raisons techniques

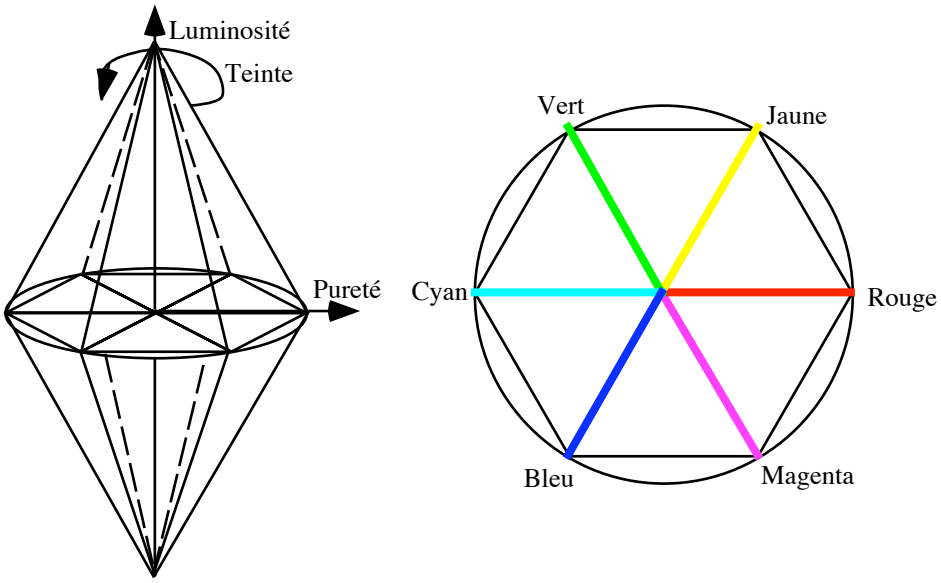

FIG. 6 – *Le modèle TLS*

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 ; ...

TAB. 1 – *Concepts pour une couleur dans ChromoFormes*

v=60 ; b=241) ; (r=245 ; v=31 ; b=229) ; (r=187 ; v=37 ; b=171) (r=202 ; v=60 ; b=191) ; (r=167 ; v=24 ; b=203) ; (r=179 ; v=43 ; b=144).

7.2 EngloFormes

Le projet *EngloFormes* consiste à disposer de manière déclarative des boîtes englobantes de bâtiments ou de parties de bâtiments [23]. Il constitue un cas particulier du projet VoluFormes [5]. 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.

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

(impression noir et blanc), nous nous sommes contentés de donner les composantes selon le modèle classique RVB

boîte soit posée sur le plan inférieur de l'univers.

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.

Ainsi construit, EngloFormes permet de décrire des cubes. Les concepts et les propriétés de base disponibles sont donnés par le tableau 2.

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	Cf. ChromoFormes (mode Liste)
Surface	faible ; moyen(ne) ; important(e)

TAB. 2 – *Concepts pour un cube dans EngloFormes*

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 7 sont des solutions possibles à cette description.

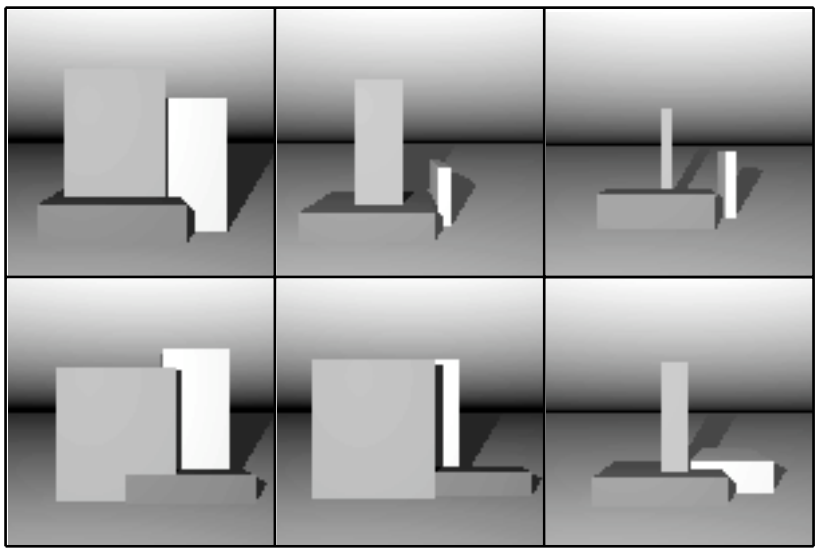

FIG. 7 – *Solutions possibles à une description*

8 Conclusion

Le nombre important et varié de projets sur la modélisation déclarative nous ont amenés à envisager le développement d'une plate-forme pour modeleurs déclaratifs. Cette plate-forme a pris corps avec le projet CordiFormes. Les grandes caractéristiques que nous avons voulues pour ce projet nous ont incités à exploiter une représentation orientée objet. Grâce à cette représentation, CordiFormes est une plate-forme :

- simple, les éléments essentiels étant déjà définis, le concepteur ne se concentre que sur les notions spécifiques au domaine d'application ;
- souple, tous les éléments peuvent être adaptés selon les besoins ;
- efficace, les algorithmes standard ainsi que ceux de la bibliothèque sont parfaitement adaptés aux problèmes liés à la génération de scènes ;
- extensible, le concepteur peut, s'il le désire, proposer ses propres algorithmes en fonction de leur efficacité dans le domaine considéré ;
- possédant une bibliothèque extensible, beaucoup d'objets se trouvent dans la bibliothèque (ceux appartenant aux ontologies des domaines les plus courants) et le concepteur l'enrichie à volonté.

Ces caractéristiques et la simplicité quant à la définition des concepts de la scène font de cette plate-forme un outil d'assistance performant tant au niveau de la facilité et que de la rapidité pour le développement d'un modeleur déclaratif.

A ce stade CordiFormes est opérationnelle. Cependant, il n'existe pratiquement pas d'outils d'aide à la conception. Le concepteur doit maîtriser un minimum les éléments de base et consulter une documentation peu organisée. Actuellement, nous étudions un système d'aide à la conception basé sur des techniques d'intelligence artificielle récentes ([26]).

Le paramétrage de certaines fonctions est automatique. Cependant, si le concepteur les estime inadaptées au domaine d'application, il doit ajuster lui-même un grand nombre de paramètres au maniement parfois délicat. Nous étudions donc un système d'apprentissage pour les régler en fonction des objectifs du concepteur mais aussi des habitudes des utilisateurs.

Enfin, les outils d'interface proposés sont actuellement assez rudimentaires. Il est indispensable d'effectuer une étude exhaustive des outils de description et de prise de connaissance afin sélectionner et d'adapter les plus génériques et les plus performants.

Références

[1] Michel Lucas, Philippe Martin, Dominique Martin, and Dimitri Plemenos. Le projet ExploFormes : quelques pas vers la modélisation déclarative de formes. In BIGRE, editor, *Acte des journées AFCET-GROPLAN*, 67, pages 35–49, Strasbourg, France, 1989.

[2] Michel Lucas. Equivalence Classes in Object Shape Modeling. In *IFIP TC5/WG 5.10 Working Conference on Modeling in Computer Graghics*, pages 35–49, Tokyo, Japan, 1991.

[3] Sandeep Kochhar. CCAD : A Paradigm for Human-Computer Cooperation in Design. *IEEE Computer Graphics and Applications*, 14(3):54–65, May 1994.

[4] Robert F. Woodbury. Searching for Designs : Paradigm and Practice. *Building and Environment*, 26(1):61–73, 1991.

[5] Danièle Chauvat. *Le projet VoluFormes : un exemple de modélisation déclarative avec contrôle spatial.* thèse de doctorat, Université de Nantes, Nantes, France, décembre 1994.

[6] E. Giunchiglia, A. Armando, P. Traverso, and A. Cimatti. Visual Representation of Natural Language Scene Description. *IEEE Transaction on Systems, Man and Cybernetics*, 26(4):575–589, 1996.

[7] François Poulet and Michel Lucas. Modelling Megalithic Sites. In *Eurographics'96*, pages 279–288, Poitiers, France, septembre 1996.

[8] Jean-Yves Martin. *Synthèse d'images à l'aide d'automates cellulaires.* thèse de doctorat, Université de Rennes, Rennes, France, décembre 1990.

[9] Andrew Rau-Chaplin, Brian MacKay-Lyons, and Peter F. Spierenburg. The LaHave House Project : Towards an Automated Architectural Design Service. In *Cadex'96*, pages 24–31, 1996.

[10] Laurent Champciaux. Declarative Modelling : Speeding Up the Generation. In *CISST'97*, pages 120–129, Las Vegas, USA, 1997.

[11] Lachmi Khemlani. GENWIN : A Generative Computer Tool For Window Design in Energy-Conscious Architecture. *Building and Environment*, 30(1):73–81, 1995.

[12] Dominique Martin and Philippe Martin. Declarative Generation of a Family of Polyhedra. In *GraphiCon'93*, St Pertersburg, Russia, septembre 1993.

[13] Christian Colin, Emmanuel Desmontils, Jean-Yves Martin, and Jean-Philippe Mounier. Working with Declarative Modeler. In *Compugraphics'97*, pages 117–126, Villamoura, Portugal, 15-18 décembre 1997.

[14] Michel Lucas. Conception assistée par ordinateur et modélisation déclarative de formes. In *Colloque PRIMECA*, pages 93–98, Chantenay–Malabry, France, Novembre 1993.

[15] Michel Lucas and Emmanuel Desmontils. Les modeleurs déclaratifs. *Revue Internationale de CFAO et d'infographie*, 10(6):559–585, 1995.

[16] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. Painting with Light. In ACM SIGGRAPH, editor, *Siggraph'93, Computer Graphics Proceedings, Annual Conference Series 1993*, pages 143–146, Anaheim, California, USA, 1–6 août 1993.

[17] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH : An Interface for Sketching 3D Scene. In ACM SIGGRAPH, editor, *Siggraph'96, Computer Graphics Proceedings, Annual Conference Series 1996*, pages 163–170, août 1996.

[18] Andrew Rau-Chaplin, Brian MacKay-Lyons, Timmy Doucette, Jedrzej Gajweski, Xiangqun Hu, and Peter F. Spierenburg. Graphics Support for a World-Wide-Web Based Architectural Design Service. In *Compugraphics'96*, pages 83–92, 1996.

[19] Pierre Poulion, Karim Ratib, and Marco Jacques. Sketching Shadows and Highlights to Position Lights. In *Computer Graphics International'97*, pages 56–63, 1997.

[20] Emmanuel Desmontils. *Le projet CordiFormes : une plateforme pour la construction de modeleurs déclaratifs.* thèse de doctorat, Université de Nantes, Nantes, France, janvier 1998.

[21] Emmanuel Desmontils. Une formalisation des propriétés en modélisation déclarative à l'aide des ensembles flous. In *3IA'96*, pages 87–105, Limoges, France, 3–4 avril 1996.

[22] J. Charlet, B. Bachimont, J. Bouaud, and P. Zweigenbaum. Ontologie et réutilisabilité : expérience et discussion. In Cépadues-Editions, editor, *Acquisition et ingénierie des connaissances*, pages 69–87, Toulouse, 1996.

[23] Emmanuel Desmontils and Jean-Yves Martin. Properties Taxonomy in Declarative Modeling. In *CISST'97*, pages 130–138, Las Vegas, USA, 1997.

[24] K. Arnold and J. Gosling. *Le langage Java.* International Thomson Publishing France, Paris, France, 1996.

[25] David Flanagan. *Java in a Nutshell.* OReilly International Thomson, Paris, France, 1996.

[26] Emmanuel Desmontils and Francky Trichet. Introduction d'une Base de Connaissances de type tâche/méthode pour assister la conception de modeleurs déclaratifs. Rapport de recherche No 166, IRIN, Nantes, décembre 1997.

[27] Lofti Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.

[28] Christian Colin. *Modélisation déclarative de scènes à base de polyèdres élémentaires.* thèse de doctorat, Université de Rennes, Rennes, France, décembre 1990.

[29] Dimitri 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, Université de Nantes, Nantes, France, 1991.

[30] Sylvain Liège. *La Modélisation Déclarative Incrémentale : Application à la Conception Urbaine.* thèse de doctorat, Université de Nantes, Nantes, France, décembre 1996.

[31] John M. Snyder. Interval Analysis For Computer Graphics. In ACM SIGGRAPH, editor, *Siggraph'92, Computer Graphics Proceedings, Annual Conference Series 1992*, pages 121–129, Chicago, USA, 26–31 juillet 1992.

[32] Dominique Martin and Philippe Martin. Pluriformes : un environnement pour le développement de modeleurs déclaratifs. Rapport de recherche No 144, IRIN, Nantes, décembre 1996.

[33] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[34] A. Hickethier. *Le cube des couleur.* Dessain et Tolra, Paris, France, 1981.

[35] D. Legrand. *La couleur imprimée: mode d'emploi.* Trait d'union graphique, Paris, France, 1990.

[36] J.-P. Couwenberg. *L'indispensable pour maîtriser la couleur.* Marabout, Alleur, Belgique, 1992.