La Programmation par Contraintes pour le formatage spatial d'un document multimédia interactif

A.MAREDJ, A. IDER, A. HAMMOU
Laboratoire Bases de Données et Systèmes d'information
CE.R.I.S.T, Rue des trois frères Aissou, Ben Aknoun Alger, Algérie.
amaredj@mail.cerist.dz

1. Introduction

Dans le domaine de l'édition et de la présentation de documents multimédia interactifs, outre les problèmes de la spécification et de la vérification de la cohérence temporelle d'un scénario dus essentiellement à la nature des applications (synchronisation temporelle et spatiale de l'enchaînement des médias) et des médias qui les composent (aspect temporisé de la vidéo et du son, l'indéterminisme lié à la disponibilité des média sur les périphériques de restitution en temps souhaité et l'interaction utilisateur) qui font l'objet de quelques travaux, l'aspect du formatage spatial reste très peu abordé. C'est dans cette perspective que nous allons essayer de montrer l'intérêt de la prise en charge de ce problème à travers sa modélisation par une PPC. Les performances de l'approche vont se mesurer essentiellement par sa capacité à satisfaire, d'une part, les objectifs d'une application d'EPDMI, et d'autre part, à proposer des solutions de formatage spatial qui soient les plus proches possibles des spécifications de l'auteur.

Ce qu'on peut globalement résumer dans ce qui suit :

- offrir, sans restriction, à l'auteur l'ensemble des relations spatiales et leur composition lors de sa spécification d'un emplacement spatial d'un objet.

- permettre une édition déclarative du document, à l'inverse d'une édition impérative (langage de programmation ou scripts) qui nécessite des connaissances particulières en informatique ce qui va limiter par conséquent l'utilisation de ce type d'applications qui est de plus en plus utilisé dans des domaines très divers.

- assurer une édition incrémentale de l'application : le placement spatial des éléments doit être un processus incrémental qui s'effectue par ajouts et retraits

successifs de relations entre les éléments, ou par les déplacements de ceux-ci sur l'écran jusqu'à obtenir le résultat souhaité. Pour chacune de ces opérations, le retour visuel doit être le plus immédiat possible.

- maintenir l'intégrité du document : pour chaque opération d'édition effectuée par l'auteur, le système doit vérifier sa cohérence par rapport à l'état courant du document.

- assurer la gestion du recouvrement spatial : dans un document multimédia les éléments qui se présentent aux mêmes instants, ou plus exactement les éléments dont les intervalles de présentation ne sont pas disjoints, ne doivent pas se recouvrir spatialement.

Dans la suite de ce papier nous définissons le placement spatial ainsi que la technique de la programmation par contraintes. Suite à quoi nous présentons et nous commentons les différentes approches de la PPC utilisées pour la résolution des systèmes de contraintes, une intention particulière sera accordée au résolveur cassowary, basé sur le simplex, jugé comme une solution très adaptée pour le formatage spatial de documents multimédia interactifs. Nous concluons à la fin par une discussion quant à l'efficacité d'utiliser la PPC et plus particulièrement le résolveur cassowary pour le formatage spatial de documents multimédia interactifs

2. Le formatage spatial

Contrairement à un document textuel ou le formatage décrit la mise en page et la mise en ligne d'éléments statiques, le formatage spatial d'un document multimédia consiste plutôt en un problème de placement relatif d'éléments dynamiques qui peuvent ne pas être présents au même instant. Il doit, d'une part, permettre d'exprimer des relations de positionnement entre les éléments, comme par exemple le fait que deux éléments soient alignés sur leur bord gauche ou bien qu'un élément se trouve au dessus d'un autre. Et, d'autre part, produire une solution de placement spatial à partir des spécifications de l'auteur. De plus la présentation d'un document multimédia ne repose plus sur un modèle de page dans lequel la dimension verticale est prioritaire mais doit pouvoir exprimer de manière identique des relations selon l'axe vertical et l'axe horizontal.

3. Programmation Par Contraintes (PPC) [1] [5] [6]

Avant d'aborder la PPC, nous allons tout d'abord définir la notion de contrainte et des systèmes de contraintes.

- Contrainte

Une contrainte est une relation qui porte sur une ou plusieurs variables. Par exemple, la contrainte *gauche + largeur = droite* exprime la relation que le bord droit d'un rectangle se trouve espacé du bord gauche d'un nombre d'unité égal à *largeur*. Résoudre ou satisfaire une contrainte consiste à trouver, pour chaque variable, une affectation de valeur qui satisfait la condition définie par celle-ci. Normalement une contrainte doit être maintenue tant qu'elle est active même si l'une ou l'autre des variables qui la composent est modifiée.

- Système de contraintes

Un système de contraintes consiste en un ensemble de variables et en un ensemble de contraintes qui limite les combinaisons de valeurs pour ces variables. Formellement, un système de contraintes est un ensemble de variables et un ensemble de contraintes qui limite les valeurs possibles pour ces variables. Il est défini par un triplet $\{V, D, C\}$ où :

- $V = \{V_1, V_2, ..., V_n\}$ est un ensemble fini de variables.
- $D = \{D_1, D2, ..., D_n\}$ représente l'ensemble des domaines de valeurs, *pour i dans* [1, n], D_i est le domaine

  des valeurs possibles pour la variable V_i.
- $C = \{C_1, C_2, ..., C_n\}$ est un ensemble de contraintes entre les variables,
- c'est-à-dire un sous ensemble de $D_1xD2x...xD_n$. Chaque contrainte peut être définie en intention par une équation ou en extension par l'ensemble des *t-uples* qui la satisfont.

Une solution pour un tel système est une instanciation des variables de V qui satisfait toutes les contraintes de C.

- Programmation par contraintes (PPC)

La PPC est un ensemble de techniques pour résoudre des systèmes de contraintes. Elle repose sur une approche déclarative des problèmes et offre une séparation claire entre la phase de définition et la phase de résolution. L'utilisateur décrit le résultat à obtenir sans préciser la façon de l'obtenir (Fig.1).

La PPC offre plusieurs avantages, notamment celui de fournir un moyen intuitif à l'utilisateur pour décrire des relations entre des objets, comme par exemple spécifier *A avant B* pour exprimer qu'une tâche A précède une tâche B. Elle permet également de simplifier la tâche de programmation. En effet, dans une approche impérative, le programmeur doit à la fois déclarer les relations mises en jeu et écrire le code nécessaire à leur maintien durant la phase d'exécution. Dans une approche basée sur les contraintes, le programmeur doit seulement exprimer les relations entre les objets sans tenir compte de la manière dont elles seront maintenues à l'exécution. Cette tâche incombe à un programme spécifique appelé *résolveur de contrainte,* qui assure automatiquement la satisfaction de l'ensemble des contraintes spécifiées.

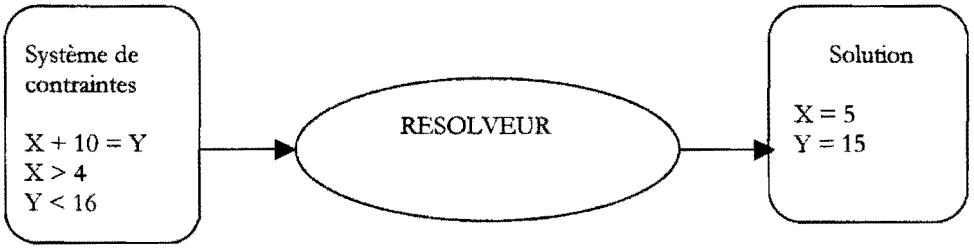


***Figure 1** : Structure d'un programme à base de contraintes*

La PPC présente néanmoins, d'une façon générale, certains inconvénients. D'abord, l'évaluation du temps d'exécution d'un programme peut être dépendante du problème traité et il n'est pas possible d'énoncer des règles claires concernant les performances temporelles, ce qui s'accommode mal avec les exigences de certaines projets. De plus le niveau requis pour programmer efficacement une application à base de contraintes est supérieur à celui exigé pour une qualité comparable en programmation classique. Cependant dans le cas qui nous intéresse ces inconvénients ne se présentent presque pas du moment que la technique est utilisée en amont pour fixer les coordonnées des emplacements des objets; processus qui s'exécute lors de la phase de construction de l'application, de plus les constructeurs de ces applications sont des experts dans leur domaine et restent très souvent bien imprégnés du domaine informatique.

4. Les approches de résolution de contraintes [2] [4]

On rencontre dans la littérature deux catégories d'approches de résolution de contraintes : les approches locales et les approches globales.

4.1. Les approches locales

Dans une approche locale, les contraintes sont traitées individuellement, de façon séquentielle. Elles n'utilisent que des informations qui leur sont disponibles immédiatement, c'est-à-dire les valeurs des variables qu'elles contraignent. Une contrainte peut être vue comme responsable d'elle même, elle a une certaine autonomie et est capable d'action et de réaction. Dans un système de contraintes tel que nous l'avons défini ultérieurement nous associons à chaque contrainte un ensemble de *méthodes de résolution* qui déterminent les différentes façons de résoudre localement la contrainte. Une méthode consiste en zéro ou plusieurs variables en entrée, en une ou plusieurs variables en sortie et en une suite d'instructions qui calculent la ou les variables de sortie en fonction des variables d'entrée de manière à satisfaire la contrainte. Sur l'exemple de la contrainte *C1 : C=A+B* (fig.2), on peut associer trois méthodes de résolution, chacune stipulant qu'une variable est calculée en fonction des deux autres: *A:=C-B* (fig2.a), *B:=C-A* (fig2.b) et *C:=A+B* (fig2.c). Si la valeur de *B* ou *C* est modifiée, alors la contrainte peut être maintenue en exécutant la première méthode.

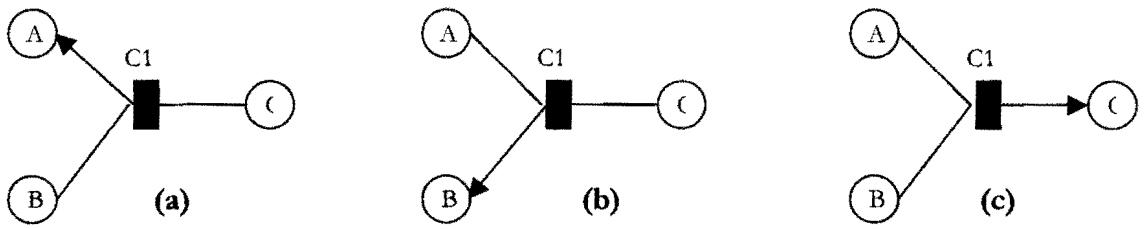


***Figure 2** : Les différentes méthodes de résolution*

L'avantage principal des approches locales réside dans leur faible coût, ce qui en fait des techniques particulièrement bien adaptées dans un contexte interactif. Leur principal inconvénient provient du fait qu'elles ne sont pas complètes, c'est-à-dire qu'elles peuvent ne pas trouver de solution alors qu'il en existe.

L'une des techniques basées sur l'approche locale est la technique par propagation locale, qui est utilisée dans la plupart des applications interactives.

4.1.1 La propagation locale [7]

Les résolveurs de contraintes basés sur cette technique construisent dans un premier temps un graphe de contraintes (Fig.3). Dans un deuxième temps, à chaque contrainte est associée un ensemble de méthodes qui explicitent comment répercuter la modification d'une variable sur une autre variable présente dans la contrainte. Par exemple, une méthode associée à la contrainte C1 peut être A := C – B qui indique comment répercuter une modification de C et/ou B (les entrées de la méthode) sur A (la sortie de la méthode).

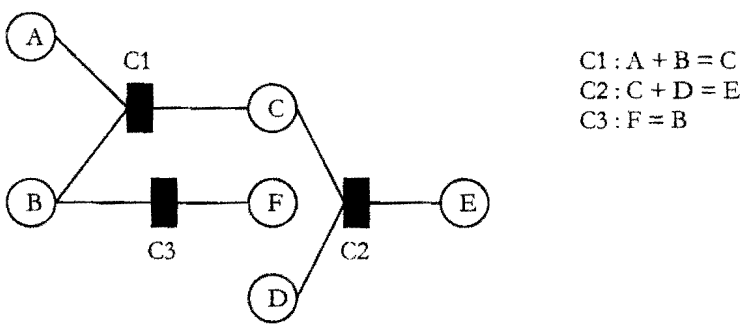


***Figure** .3 : Graphe de contraintes*

Une fois ces informations mises à jour, la fonction d'un résolveur local, après une modification de la valeur d'une variable, est de décider :

- *des contraintes à réévaluer :* seules celles qui risquent d'être violées par la perturbation et par ses conséquences sont reconsidérées.
- *des méthodes à appliquer :* pour chaque contrainte à réévaluer, il faut décider quelle est la méthode associée qui va permettre de la satisfaire à nouveau.
- *Et de l'ordre dans lequel celle-ci doivent être appliquées* afin de satisfaire à nouveau l'ensemble des contraintes.

L'idée sous-jacente de la technique de propagation locale est de dire : dès que la contrainte possède assez d'information pour calculer des valeurs, elle les calcule. La résolution se décompose en deux phases :

1. *Phase de planification* : au cours de laquelle le résolveur sélectionne une méthode pour chacune des contraintes à réévaluer et uniquement pour celles-ci.

Cet ensemble de méthodes est calculé en partant de la variable perturbée et en identifiant toutes les contraintes insatisfaites suite à cette perturbation. Le résultat de cette phase est un graphe orienté (*plan*). Par exemple une orientation possible de l'exemple de la Fig.4 est donnée dans la Fig.4. L'orientation de ce graphe répercute une modification de A sur les variables C, D, F.

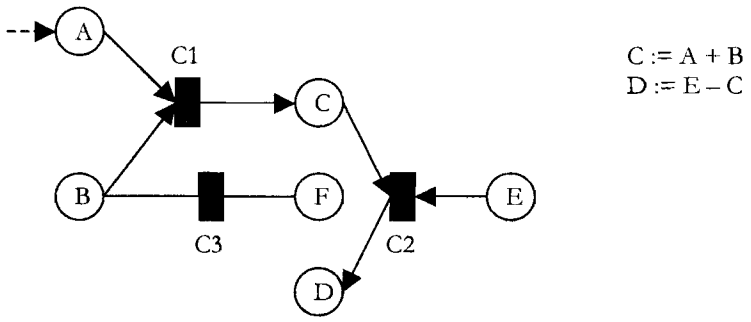


Figure 4 : *Orientation du graphe de contraintes*

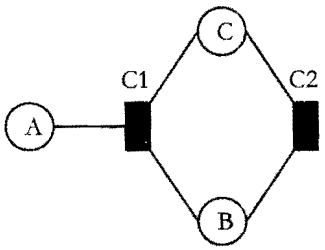
Phase de résolution : pendant laquelle le plan constitué est exécuté en séquence. Dans cette phase, les valeurs connues sont propagées le long des arcs du graphe de contraintes.

Les techniques de la propagation locale présentent plusieurs avantages. En premier lieu, l'efficacité et la facilité de compréhension du comportement des variables. De plus l'utilisation d'un plan est très intéressante dans le contexte d'édition interactive. En effet, lorsque l'auteur sélectionne un objet pour le déplacer à l'écran le résolveur calcule un plan, c'est-à-dire un sous ensemble du graphe de contraintes qui sera suffisant pour répercuter les perturbations (déformations) lors du déplacement. Le résolveur oriente ce sous graphe pour propager la valeur modifiée par l'auteur (par le déplacement). Ainsi, lors de chaque déplacement élémentaire le système n'aura qu'à utiliser le même plan pour propager les modifications sur les autres variables affectées par la perturbation.

Les insuffisances majeures de cette technique sont :

- *L'incomplétude de la technique* : c'est-à-dire le système peut ne pas trouver de solution alors qu'elle en existe une. Cela est lié par exemple à une orientation du graphe vers une évaluation des variables sans solution.

- *L'impossibilité de supporter des cycles dans le graphe de contrainte* : par conséquent, les résolveurs basés sur cette technique ne supportent pas les systèmes de contraintes qui introduisent des cycles. Prenons par exemple le système représenté par le graphe de la Fig.5. Ce système est composé de deux contraintes *C1* et *C2* : $\{C1 : A + B = C ; C2 : C = B\}$. Trois méthodes sont associées à C1 : A := C - B ; C := A +B ; B := C –A et deux autres sont associées à C2 : C := B et B := C.



C1 : C=A + B
C2 : C = B

Figure : *5 Exemple de graphe cyclique*

Supposons que l'on parte de la solution (A, 4), (B, 2), (C, 2) et que l'on modifié A avec la valeur 10. Le résolveur doit choisir une méthode où A est une variable d'entrée, soit B := A – C soit C := A – B puis propager la nouvelle modification. S'il choisit B := A – C, il doit maintenir la contrainte C2 en exécutant une méthode où B est en entrée soit C := B. La variable C étant modifiée, le résolveur doit de nouveau maintenir C1, ce qui engendre un cycle. Une solution serait d'arrêter la propagation après un seul parcours de boucle, mais on obtient alors une solution incohérente : (A,10), (B,8), (C,8). Si le résolveur avait choisi au départ d'exécuter la méthode C := A – B le résultat aurait été identique.

- *La manipulation des contraintes uniquement fonctionnelles.* Une contrainte est fonctionnelle, si chacune de ses variables possède une valeur unique en fonction de la valeur des autres variables de la contrainte. Les contraintes d'inégalité, par exemple, ne sont pas fonctionnelles, car si on a la contrainte A <= B, et étant donné la valeur de A, on ne peut déterminer de façon unique la valeur de B. Les contraintes non fonctionnelles sont très utiles pour spécifier certains aspects des interfaces utilisateurs, comme le fait qu'un objet A doit être à gauche d'un objet B ou que les objets d'une fenêtre ne doivent pas dépasser les limites de celle-ci.

De nombreux résolveurs reposants sur cette technique ont été conçus. Nous citons par exemple, les résolveurs DeltaBlue, SkyBlue, Quick Plan, ainsi que UltraViolet / Indigo qui ont été tous développés à l'université de Washington.

4.2 Les approches globales [8]

Dans une approche globale, les méthodes de résolution peuvent fournir toutes les solutions d'un système de contraintes. Pour cela, toutes les contraintes du système sont considérées, c'est-à-dire que les méthodes parcourent tous les noeuds du graphe de contraintes. De plus ces méthodes traitent généralement les contraintes de façon simultanée, comme par exemple les méthodes numériques pour résoudre un système d'équations.

L'avantage de ces approches est leur complétude car elles permettent de toujours trouver une solution si celle-ci existe. Leur inconvénient principal réside dans leur coût généralement très élevé et dans le fait qu'elles sont spécifiques à un type de données particulier (variables numériques par exemple).

Il existe de nombreux algorithmes qui se basent sur cette approche et qui se différencient de part leur manière de parcourir l'ensemble des valeurs. Pour illustrer cette approche, On ne présentera qu'un seul algorithme de cette classe qui est représentatif de l'approche, c'est L'algorithme basé sur le retour arrière (backtrack)

4.2.1. Algorithme de backtrack

Nous rappelons simplement les caractéristiques essentielles de cet algorithme. Dans l'état initial, aucune variable n'est affectée. À une étape donnée, l'ensemble des variables est divisé en deux groupes, les variables affectées et les variables non affectées. La solution donnée par les variables affectées est cohérente par rapport aux sous-ensembles de contraintes qui ne portent que sur ces variables. La prochaine étape va consister à affecter une nouvelle variable, le système choisit une variable et une valeur dans l'ensemble de celles qui sont possibles, et vérifie la cohérence. Si le système est cohérent, il passe à l'étape suivante, si le système est incohérent, il essaie une autre valeur pour la variable considérée. Si toutes les valeurs ont été considérées, il désaffecte une des variables affectées, et repart à l'étape n-1.

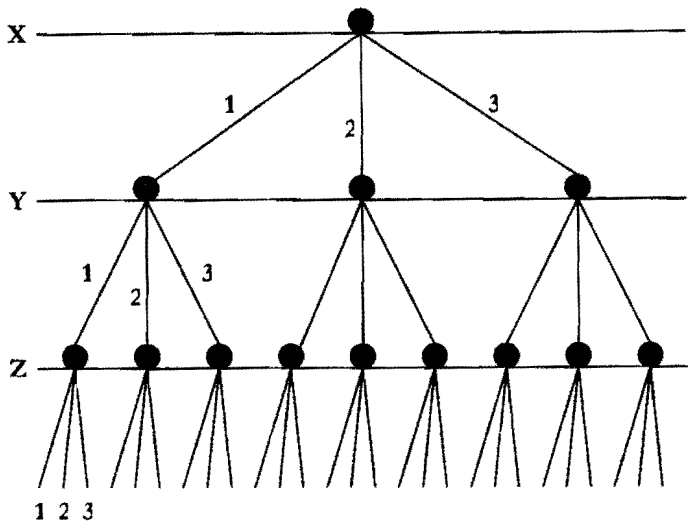


***Figure 6** : Arbre de recherche*

Dans l'exemple de la Fig.6, on peut voir un arbre de recherche dans le cas où nous aurions des contraintes qui portent sur trois variables X, Y et Z (chacune étant définie dans l'intervalle [1...3]). Le système dans un premier temps évaluera X, en lui affectant par exemple la valeur 1, dans un deuxième temps il évaluera Y puis Z. Si cette évaluation ne satisfait pas le système de contraintes, il choisira une nouvelle valeur pour Z. Si aucune des valeurs de Z ne permet de satisfaire l'ensemble de contraintes, le système choisira une nouvelle valeur pour Y, et ainsi de suite.

Il existe des heuristiques qui permettent d'améliorer et de personnaliser cet algorithme de manière à orienter la recherche de solutions (par exemple les heuristiques portant sur le choix de la variable à remettre en cause lors d'un retour arrière).

L'avantage majeur de cette technique est sa complétude : on trouve toujours la solution si elle existe, et la possibilité pour l'utilisateur de contrôler la recherche de solution en cas de réponses multiples. Ses inconvénients sont :

- l'obligation de travailler sur des domaines finis, or nos systèmes de contraintes manipulent des domaines infinis.
- le coût généralement très grand de cette technique, qui procède par énumération, lorsque le domaine des variables est trop grand.

4.2.2 Approches basées sur le simplexe

Ces approches se basent sur la méthode du simplexe, qui a été proposée par Dantzig vers 1947. Cette méthode simple, robuste et efficace permet d'attaquer avec succès des problèmes linéaires comportant plusieurs dizaines de milliers de variables et de contraintes. De nombreuses variantes ont été réalisées de manière à améliorer les performances temporelles de la résolution et dans un deuxième temps de définir une fonction d'optimisation (*f*) appropriée. L'objectif de l'algorithme est de maximiser la valeur de cette fonction tout en satisfaisant l'ensemble des contraintes. Ces approches se révèlent mieux adaptées au contexte des applications graphiques notamment pour le placement spatial. En effet, contrairement aux approches locales qui n'acceptent pas les systèmes de contraintes qui présentent des cycles ou des inégalités, celles basées sur le simplexe les gèrent efficacement. De plus, leurs performances temporelles sont proches de celles des approches locales contrairement au backtrack qui est très long et mal adapté à notre contexte. Donc, ces approches combinent les performances temporelles et la complétude des solutions.

Les nombreux travaux qui ont été fait ces dernières années sur le simplexe, ont abouti à la réalisation d'un résolveur de contraintes appelé Cassowary.

- Cassowary

Cassowary est un résolveur de contraintes qui permet de traiter des équations et des inéquations linéaires. Cet algorithme, basé sur l'algorithme du simplexe, a été développé à l'université de Washington par Badros et Borning [Badros98]. Il se décompose en deux phases :

1. la phase de prétraitement : qui vise à traduire le problème sous une forme ne contenant ni inégalité, ni variables négatives, de manière à résoudre l'ensemble de contraintes ainsi obtenues par le simplexe :

- Les inégalités de la forme $x \geq y$ sont traduites sous la forme $x = s + y$, où s est une variable non négative appelée *variable d'écart* (*slack*).
- Les variables négatives v sont substituées par des équations de la forme $e = -v$, toutes les occurrences de v sont ensuite substituées par $- e$. La résolution des variables v se fera dans une étape postérieure, une fois la valeur de e est connue.

Ensuite le système de contraintes est partagé en deux tableaux C_S, et C_U. Le tableau C_S contiendra les variables contraintes (i.e l'ensemble des variables restreint à ne pas prendre de valeurs négatives), et C_U celles qui sont non contraintes. Cette première étape a été améliorée pour rendre la deuxième phase plus efficace. Initialement toutes les équations sont placées dans le tableau C_S. Un filtrage basé sur la méthode d'élimination de Gauss-Jordan a été rajouté. Cette phase a pour but de supprimer toutes les variables non contraintes du tableau C_S pour les placer dans C_U. Pour réaliser cela, on cherche dans C_S une équation contenant une variable v non contrainte. On retire cette équation de C_S. On résout l'équation pour v, en générant une nouvelle équation e de la forme : e1 = v (ou e1 est une expression). On ajoute e1 à C_U. On substitue alors toutes les occurrences de v dans C_S, C_U et f.

2. La phase de résolution : une solution est obtenu par la méthode du simplexe classique.

Par ailleurs, Cassowary classe les contraintes suivant une hiérarchie de priorité. Les contraintes utilisées dans l'exemple que nous venons de donner sont dites *requises*, et doivent être satisfaites impérativement. Leur non satisfaction correspond à un échec de la résolution. En plus des contraintes requises, Cassowary définit un autre type de contraintes internes appelées *contraintes non-requises*. Ces dernières ont un niveau de priorité inférieur à celui des contraintes requises, et de ce fait elles ne sont satisfaites qu'en dernier lieu, c'est-à-dire après les contraintes requises. Lorsqu'elles sont en contradiction avec une contrainte requise, le résolveur ne les prennent pas en considération.

On trouve deux classes de contraintes non-requises :

Les contraintes d'édition : Elles sont utilisées pour changer la valeur d'une variable.

Les contraintes de maintien : Elles sont utilisées pour contraindre une variable à maintenir sa valeur initiale.

Les contraintes non requises sont représentées sous forme d'équation de la forme :

$$v = \alpha + \delta_v^+ - \delta_v^-$$

Où δ_v^+ et δ_v^- sont des variables non négatives représentant la déviation de v de la valeur désirée, c'est-à-dire les erreurs. Si la contrainte est satisfaite elles aurons

toutes les deux la valeur 0. Dans le cas contraire δ_v^+ sera positive et δ_v^- nulle si v est trop grand par rapport à la valeur souhaitée, et vice-versa si v est trop petit. Puisque le but est que les valeurs de δ_v^+ et δ_v^- soient minimales, Cassowary les met dans la fonction objective. Il utilise alors une paire de variables d'erreur, au lieu d'une seule variable, pour satisfaire la restriction de non négativité des variables qui font partie de la fonction objective.

Les contraintes non-requises sont très utiles dans le domaine des applications graphiques. En effet, les contraintes d'édition sont utilisées lors d'un déplacement d'un élément par la souris et sont satisfaites tant que ce déplacement est possible. Lorsque le déplacement n'est plus possible (lorsqu'un élément a atteint un des bords de l'écran par exemple), les contraintes d'édition associée ne sont plus satisfaites et l'élément concerné ne suit pas le mouvement de la souris. Cassowary ne considère pas pour autant ceci comme un échec dans la résolution et n'affiche pas par conséquent un message d'erreur. Par ailleurs, les contraintes de maintien sont intéressantes dans la mesure où leur utilisation permet aux éléments qui ne sont pas concerné par un déplacement de rester à leurs places. Ceci donne une stabilité à l'image affichée dans l'écran.

D'autre part, les travaux menés par Alan Borning pour mettre au point le résolveur Cassowary ont aussi porté sur l'aspect incrémental des opérations d'ajout et de retrait de contraintes.

- Ajout de contraintes

Nous allons décrire dans cette section comment ajouter une nouvelle contrainte d'une manière incrémentale. Cette technique est utilisée pour trouver une solution initiale au problème original, qui est sous forme de base réalisable, en commençant par un ensemble de contraintes vide auquel on ajoute les contraintes une par une.

Pour ajouter une nouvelle contrainte requise au tableau, en premier lieu Cassowary la convertit si c'est une inégalité en une équation par l'ajout de variables d'écart. Il utilise le tableau courrant pour substituer toutes les variables de base, ce qui donne une équation de la forme $e = c$ où e est une expression linéaire et c une constante. Si c est négative, alors il multiplie les deux cotés par -1 pour qu'elle devienne positive. Si e contient une variable non contrainte il l'utilise

pour remplacer ses occurrences dans le tableau et ajouter l'équation dans la partie C_U. Sinon, il crée une variable artificielle restreinte a, ajoute l'équation $a = e - c$ dans la partie C_S du tableau et considère l'expression $e - c$ comme étant la fonction objective du système. Le but ici est de minimiser cette fonction. Si le minimum résultant est différent de zéro, le système n'a pas de solution et la contrainte est rejetée. Dans le cas contraire a est soit de base, soit hors base. Si a est hors base, sa colonne peut être tout simplement enlevée du tableau. Si elle est de base, la rangée où elle se trouve doit avoir la constante 0 (puisque nous pouvions atteindre la valeur 0 pour notre fonction objective qui est égale à a). Si la rangée est simplement $a = 0$, le résolveur la supprime. Sinon la rangé est de la forme $a = 0 + bx + e$ où $b \neq 0$. Dans ce dernier cas Cassowary fait rentrer x dans la base en utilisant cette rangée et enlève la colonne de a.

- Retrait de contraintes

Après qu'une série de pivots ait été exécutée, l'information représentée par la contrainte n'est plus contenue dans une seule rangée. Donc le résolveur a besoin d'identifier les influences de la contrainte dans le tableau pour pouvoir la retirer. Pour cela il utilise une variable appelée *marqueur*, qui est présente à l'origine uniquement dans les équations représentant la contrainte à retirer. Il retrouve par la suite l'effet de la contrainte sur le tableau en recherchant les occurrences de ce marqueur. Pour les contraintes d'inégalité, la variable d'écart s, ajoutée pour qu'elles deviennent une égalité, peut lui servir comme marqueur, puisque s à l'origine n'est présente que dans cette équation. L'équation représentant une contrainte non-requise aura une variable d'erreur qui peut servir comme marqueur. Pour les contraintes d'égalité requises, le résolveur ajoute une variable contrainte appelée *variable factice* (*dummy*) à l'équation originale pour lui servir de marqueur. Cette variable ne rentre jamais dans la base, puisqu'elle doit toujours prendre la valeur 0 pour ne pas avoir d'influences sur le système.

Pour la suppression d'une contrainte, Cassowary fait rentrer la variable marqueur dans la base par une série de pivotements. Ensuite, il enlève tout simplement la rangée où se trouve le marqueur.

- **Prototype de placement de boites**

Afin de tester les performances du résolveur cassowary nous en avons fait une implémentation adaptée à notre contexte. Celle-ci repose sur le principe de manipulation d'objets sous forme de boites (Fig.7) à travers une interface graphique. Ils sont caractérisés par six paramètres : le bord gauche, le bord droit, le bord supérieur, le bord inférieur et les deux dimensions. L'ensemble de ces paramètres représente les variables du système de contraintes. Les objets sont liés entre eux par des contraintes sur leurs dimensions ou leurs positions relatives. Les contraintes sont exprimées de façon statique avant l'exécution. En l'absence de contraintes, les objets ont un positionnement par défaut.

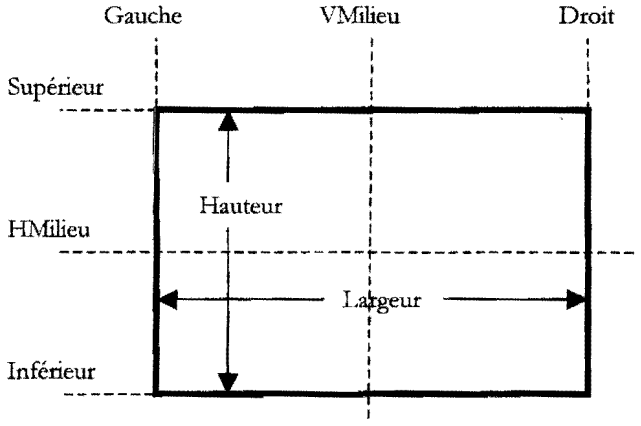


***Figure 7** : Représentation graphique d'un élément*

Soit l'exemple où les contraintes suivantes sont exprimés par rapports à des objets de couleurs :

Rouge *Aligné-Haut* Bleue, Rouge *Même-Hauteur* Bleue, Verte *A-Droite-De(0)* Rouge,

Jaune *Centré-Vertical* Orange et Mauve *En-Dessous-De* Grise

Les contraintes sur les objets sont traduites sous forme de contraintes sur leurs variables. Ainsi, "Rouge *Aligné-Haut* Bleue" est transformée en contrainte *Rouge.haut* = *Bleue.haut* et "Verte *A-Droite-De(0)* Rouge" est transformée en contrainte *Verte.gauche* = *Rouge.gauche* + *Rouge.largeur*. La dernière contrainte signifie que la boîte Mauve peut se trouver n'importe où en dessous de la boîte

Grise. Pour cela nous avons défini une contrainte d'inégalité entre deux variables *A* et *B*.

- **Déplacement d'un élément**

Lors du déplacement d'un objet, les variables concernées sont mises à jour instantanément. Dans notre exemple, un mouvement vertical de la boîte Rouge entraîne un mouvement vertical identique de la boîte Bleue et inversement, mais ne déplace pas la boîte Verte qui n'est pas liée sur le même axe. Les boîtes Jaune et Orange sont centrées sur l'axe vertical, le déplacement de l'une entraîne un déplacement identique de l'autre. La modification de la largeur de l'une entraîne également un déplacement de l'autre de façon à ce que leurs axes verticaux restent confondus.

- **Redimensionnement**

L'augmentation ou la réduction de la hauteur de la boîte Bleue entraîne une modification équivalente de la hauteur de la boîte Rouge. Une modification de la largeur de la boîte Bleue n'entraîne aucune modification sur la boîte Rouge car ces deux boîtes ne possèdent pas de contraintes sur leur largeur.

- **Les inégalités**

Un déplacement ou un agrandissement de la boîte Mauve vers le haut n'affecte pas la boîte Grise tant que cette dernière reste au dessus. Par contre, dès que le bord supérieur de la boîte Mauve atteint la valeur du bord inférieur de la boîte Grise, cette dernière suit le mouvement. De même lorsque le bord inférieur de la boîte Grise est modifié, le mouvement n'est répercuté que lorsque ce bord atteint la valeur du bord supérieur de la boîte Mauve.

7. Conclusion

Dans ce papier, nous avons présenté la programmation par contraintes, ses avantages et ses inconvénients quant à son application pour le formatage spatial dans des applications EPDMI. Nous avons accès essentiellement l'étude sur le résolveur cassowary pour démontrer sans efficacité, après adaptation, dans la prise en charge des problèmes du formatage spatial pour les documents multimédia interactifs. D'autant plus, et jusqu'à présent, le résolveur globale Cassowary n'a fait l'objet d'aucun travail dans le domaine du formatage spatial des documents multimédia, à l'inverse des résolveurs basés sur l'approche locale qui ont fait l'objet de nombreux travaux, notamment DeltaBlue qui a été utilisé pour calculer le placement spatial des éléments dans l'éditeur Madeus [Carcone97].

Au terme de cette contribution, nous pouvons faire les constats suivants :

- l'utilisation de la programmation par contraintes reste une approche souple et puissante dans le domaine de l'Edition et de la Présentation de Documents Multimédia Interactifs, et ce, en répondant avec des temps de réaction très appréciables par rapport aux exigences du formatage interactif.
- le résolveur Cassowary, offre un fort pouvoir d'expression en couvrant l'essentiel des relations possibles dans le domaine de l'Edition et de la Présentation de Documents Multimédia Interactifs. Il se révèle particulièrement intéressant avec ses performances temporelles de résolution. De plus il se démarque, des approches locales, en palliant à leur incapacité de traiter des systèmes de contraintes avec des cycles ou des inégalités, qui sont considérés à juste titre, comme des besoins réels pour un problème de placement d'objets.
- la gestion du recouvrement, qui n'est pas initialement un aspect considéré dans cassowary, est assurée par une extension simplifiée du résolveur par l'ajout d'un module de recouvrement.

Bibliographie

[1] L. Carcone, M. Jourdan, C. Roisin, ``Présentation de documents multimédia basée sur les contraintes", Workshop on Electronic Page Models - LAMPE, Lausanne (Suisse), vol. , 22-23 Septembre 1997.

[2] Bjorn Freeman-Benson, John Maloney and Alan Borning, `` An incremental Constraint Solver", Communication of the ACM, Vol. 33, num. 1, pp54-63, January 1990.

[3] L. Hardman, G. van Rossum, D. Bulterman, ``Structured Multimedia Authoring", ACM Multimedia 93, Anaheim, California, Août 1993.

[4] Hiroshi Hosobe, Satoshi Matsuoka, Akinori Yonezawa, ``Generalized local propagation: A framework for solving constraint hierarchies", Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Boston, , August 1996.

[5] Muriel Jourdan, Nabil Layaida, Loay Sabry-smail, ``Time Representation and Management in MADEUS, an authoring environment for multimedia documents", vol. SPIE 2667, SanJosé, USA, , pp. 68-79, February 1997.

[6] Nabil Layaida, "Madeus : système d'édition et de présentation de documents structurés multimédia", Thèse, Université Joseph Fourier, Grenoble I, Juin 1997.

[7] T. Suzuki, N. Kakinuma, T. Tokuda, ``An Experimental Comparison of Three Modified DeltaBlue Algorithms", Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, vol. Boston, , August 1996.

[8] UW Cassowary Constraint Solving Toolkit
www.cs.washington.edu/research/constraints/cassowary/