Quels langages pour mieux contrôler le formatage des documents multimédia ?

Frédéric Bes, Cécile Roisin

Projet Opéra - Inria Rhône-Alpes
Grenoble, France

{Frederic.Bes,Cecile.Roisin}@inrialpes.fr

L 'évolution des techniques de traitement des documents a conduit à l'émergence de nombreux langages de présentation et d'outils de formatage associés. Dans cet article, un bilan de ces langages et formateurs est effectué pour évaluer leur comportement face au nouveau défi que constitue le déploiement de grandes variétés de terminaux de restitution. En particulier nous identifions la notion d'échec de formatage. Nous montrons que les besoins en souplesse de formatage induisent une extension de l'expressivité des langages actuels de façon à permettre à l'auteur de contrôler plus finement le comportement du formateur. Enfin, nous décrivons notre proposition qui consiste à offrir des opérateurs de formatage de haut niveau et des expressions de priorité et de repli en complément des langages déclaratifs existants : ces spécifications sont exploitées par notre formateur pour éviter les cas d'échec.

1. Introduction

On n'a jamais autant communiqué par l'intermédiaire de documents électroniques, que ce soit de documents en formats propriétaires et surtout en formats standard comme à travers le web. Cependant on peut constater très souvent une baisse de la qualité de la forme finale des documents ainsi échangés. Qui n'a pas déploré le mauvais placement d'une figure dans sa page ou un découpage peu heureux d'une suite de vignettes à l'écran ! Ce n'est pas seulement dû au manque de maîtrise des outils par les auteurs, mais aussi à la complexité grandissante des fonctions que l'on souhaite faire réaliser par les formateurs. Cette frustration du résultat du formatage est actuellement amplifiée par l'évolution actuelle des techniques de production de documents. En effet, de documents statiques édités manuellement à l'unité, on arrive à des systèmes de génération de documents multimédia à partir de sources multiples pour des terminaux de restitution ayant des capacités très variées. Les concepteurs de ces systèmes de production documentaire doivent pouvoir disposer de langages de présentation puissants et de formateurs qui puissent générer des présentations adaptées à chaque situation de lecture. Comme nous le verrons dans la section 2 ci-dessous, les langages standard actuels couvrent seulement en partie les besoins de ces applications. Une autre approche est d'utiliser un langage laissant plus de souplesse dans la spécification pour que le formateur ait à sa disposition un espace de solutions dans lequel il calculera la solution adaptée à chaque contexte. Les expériences les plus probantes dans ce domaine ont été faites à partir de résolveurs de contraintes (voir section 3). Cependant, même avec cette méthode, les systèmes de présentation multimédia n'offrent pas encore de solutions tout à fait satisfaisantes. En particulier, les formateurs entrent en situation d'échec, soit parce qu'ils ne peuvent trouver de solution avec le jeu de contraintes issu de la spécification de l'auteur, soit que la solution fournie n'est pas celle qui est attendue.

Il est donc nécessaire de proposer de nouveaux moyens pour exprimer les besoins de présentation incluant à la fois la variété et la qualité des résultats de formatage. Notre démarche a été d'évaluer précisément les manques des langages et formateurs actuels (sections 2 et 3) et d'en proposer des extensions (section 4) ainsi que leur expérimentation (section 5).

2. Les langages déclaratifs de présentation

Le cadre de notre travail s'inscrit dans celui des langages de présentation *déclaratifs* ouverts. Il n'est plus envisageable à l'heure actuelle de proposer des solutions qui ne s'intègreraient pas facilement avec les technologies de décoration (CSS) ou de transformation (XSL) de documents structurés. En effet, avec le développement de nombreux langages structurés basés sur XML, comme XHTML, SMIL, MATHML ou SVG, la partie réservée à la présentation est maintenant très clairement détachée de la description logique. Ainsi, CSS permet à l'auteur de définir la présentation souhaitée des documents XML en attachant des propriétés de style aux éléments de la structure source, avec un mécanisme de base d'héritage de propriétés qui s'appuie sur cette structure.

2.1. Limites du pouvoir d'expression des langages de présentation

Lorsque l'on considère les besoins de production actuels des documents, la première limitation vient de leur manque de souplesse. On peut expliquer cette situation par le fait que les langages ont d'abord eu pour objectif de permettre de spécifier le formatage de *flots textuels* tel que le choix de la fonte, la taille des caractères, le style associé, la césure dans les paragraphes ou encore le découpage en pages. Les besoins actuels des présentations multimédia nécessitent de spécifier le placement spatial et temporel d'objets dans les différents axes comme la hauteur, la largeur, la profondeur et le temps. Les langages de placement d'intervalles et de boîtes (les régions dans SMIL, les positions des objets dans SVG, les placements sur un axe temporel dans Director, etc.) offrent en général peu de souplesse dans la spécification car ces placements sont souvent définis par valeurs et non par relations. Les seuls placements relatifs ne sont possibles qu'entre objets liés hiérarchiquement comme celui d'une boîte vis-à-vis de son englobante dans SMIL).

Une autre rigidité des langages de présentation vient de la dépendance du langage vis-à-vis du formateur. C'est évidemment le cas des langages des outils propriétaires comme Word ou Director, et de Latex [LAM86]. Cependant une brèche à cette dépendance a été ouverte grâce à l'émergence de standards comme CSS, SMIL et XSL-FO. Une des conséquences attendues est que les documents peuvent être formatés par partie avec des formateurs spécialisés adaptés à chaque type d'objet à formater. C'est le cas de l'architecture de formatage de XSL-FO. Un premier niveau de souplesse est fourni dans les langages SMIL, CSS et XSL-FO par des opérateurs de choix et d'*alternatives*. Par exemple en XSL-FO, il est possible de spécifier le résultat du formatage lors du dépassement dans un bloc en donnant à la propriété *fo :overflow* l'une des valeurs : *visible*, *hiden* ou *scroll* ou encore sélectionner un élément à afficher grâce à l'élément *fo :multi-switch*. L'opérateur *switch* du langage SMIL offre également le moyen de spécifier l'affichage de structures temporelles en fonction de paramètres comme la langue (*system-language*) ou le débit réseau (*systemBitrate*). Cependant la souplesse introduite reste limitée à un seul paramètre comme la langue de l'utilisateur ou la largeur de l'écran dont la valeur est connue lors du formatage.

Pour finir, quels que soient les langages considérés, il n'est pas possible actuellement pour les auteurs de spécifier des critères de présentation globaux, comme la durée totale de présentation, le nombre d'images dans une page, qui puissent être pris en compte par le formateur. Ce besoin est essentiel pour assurer un résultat de formatage satisfaisant lors des processus de génération de présentation dans lesquels le nombre d'objets à formater n'est pas connu statiquement. Comme c'est le cas du résultat d'une requête de base de données. Le besoin de s'appuyer sur des langages ayant un pouvoir d'abstraction et de choix global apparaît très fortement dans les travaux concernant la génération et l'adaptation de documents multimédia [AND00], [VIL00], [VAN01] et [BES01]

Nous verrons en section 3 comment les approches de spécification par contraintes permettent de lever en partie ces limitations.

2.2. Qualité du formatage

Le processus le plus répandu de conception d'un document avec ces langages est un aller-retour entre édition du contenu et édition des feuilles de style, jusqu'à l'obtention du résultat de présentation souhaité. Les moyens d'édition peuvent être très différents selon le type de langage et surtout selon que l'on dispose d'un outil interactif ou non pour éditer le contenu et/ou le style. La nature de la présentation visée, statique ou dynamique, impose également des modes d'édition différents, notamment parce qu'il est difficile de percevoir l'enchaînement temporel avec les outils d'édition classique. Mais dans toutes ces situations se pose la question de la qualité du résultat du formatage. En effet, l'auteur ne contrôle pas directement le formateur, il ne peut contrôler le résultat du formatage qu'à travers les moyens de spécification, c'est-à-dire les langages de présentation utilisés. Qui même n'a pas été amené à "biaiser" le formateur, c'est-à-dire utiliser des éléments du langage non prévus pour la fonction recherchée, pour obtenir le résultat souhaité ! Aussi nous définissons la notion d'échec de formatage de la façon suivante :

Le formateur d'un système de présentation est dit en *situation d'échec* s'il se trouve dans l'une des deux situations suivantes :

1. Il est incapable de produire un résultat de formatage, par exemple il tombe en erreur ou s'arrête sans fournir de solution.
2. Le résultat de formatage produit ne satisfait pas l'utilisateur.

Cette définition comporte un élément informel lié à l'appréciation de l'auteur et donc impossible à mesurer de façon automatique. Il est cependant nécessaire pour répondre aux besoins réels des auteurs.

Pour éviter à un système de présentation d'entrer en situation d'échec, les actions sont possibles à différents niveaux :

– le langage doit permettre à l'auteur d'exprimer les propriétés souhaitées.
– le formateur doit produire la solution la plus fidèle possible à l'ensemble des propriétés spécifiées.

Si ces règles paraissent triviales dans l'absolu, elles ne le sont pas dans la réalité du fait de la complexité des règles de présentation et de leur interdépendance. De plus, comme on l'a vu, le besoin d'obtenir plusieurs présentations pour un même document nécessite d'avoir d'autres moyens d'expression de la présentation que les règles attachées aux blocs, par l'introduction de plus de souplesse dans la spécification, comme avec les opérateurs *overflow* de XSL-FO et *switch* de SMIL. Il est clair que cette extension dans le pouvoir d'expression augmente le risque d'entrer en situation d'échec. Malheureusement l'auteur n'a pas actuellement le moyen de contrôler le résultat du formatage dans ces situations. Par exemple, l'opérateur *overflow* de XSL-FO permet d'indiquer au formateur quelle présentation effectuer lorsqu'un objet dépasse de sa zone d'affichage, en affichant le débordement, en le cachant ou en ajoutant une zone avec ascenseur, mais il n'est pas possible d'indiquer une autre stratégie s'il y a un tel débordement, par exemple pouvoir supprimer un objet précédent.

Notre démarche n'est pas de remplacer les formateurs actuels qui sont performants parce qu'issus d'une longue expérience dans le domaine du formatage de documents. Il s'agit plutôt de proposer des solutions de formatage pour compléter ces formateurs. Le mode d'expression des langages le permet grâce à leur approche déclarative et modulaire.

Dans la section suivante nous développons la classe d'approche par contraintes qui permet le plus d'évolution dans la qualité du processus de formatage et sur la base de laquelle nous allons appuyer notre proposition.

3. Contrôle du formatage dans les systèmes à base de contraintes

Depuis plusieurs années, différentes approches proposent d'utiliser de nouvelles techniques à base de contraintes pour permettre d'étendre l'expressivité des langages de présentation. C'est le cas de Isis [KIM95] et Madeus [LAY97] pour le temporel, CCSS [BAD99] et CSVG [BAD01] pour le spatial ou d'autres approches un peu plus inter-dimensionnelles comme ZYX [BOL01] et Cuypers [RUT00]. L'auteur spécifie des intervalles, continus ou discrets, associés aux domaines de définition d'un attribut de présentation comme la duréeou la dimension d'un objet spécifié par un intervalle temporel ou spatial. Il peut également utiliser un opérateur de placement paramétré par des contraintes : liste avec répartition dans un intervalle, opérateur parallèle avec égalité des intervalles des composants. Ces approches permettent ainsi de réaliser des placements complexes et en partie adaptables de part leur grande souplesse. En ce sens ils offrent une amélioration très intéressante du pouvoir d'expression offert aux auteurs et de la capacité d'adaptation des spécifications produites.

Toutefois nous restons avec ce genre d'approche dans un cas de figure où l'auteur est limité quant à son pouvoir décisionnel sur la solution de présentation choisie. Au final, c'est bel et bien toujours l'implémentation du processus de formatage qui va choisir cette solution. Cette nouvelle expressivité ne fait donc que renforcer le besoin de faire remonter le contrôle du formatage jusqu'à l'auteur puisqu'elle introduit plus de souplesse dans le document, permet de prendre en compte des critères plus globaux et devient donc plus complexe par sa description et ses traitements.

L'approche que nous proposons a pour but d'étendre l'expressivité des langages en ajoutant des capacités de *contrôle* qui puissent être pris en compte par le processus de formatage.

4. Notre approche

4.1. Introduction

Notre approche essaye de répondre à trois attentes. La première est de pouvoir donner le plus possible à l'auteur la possibilité d'exprimer ce qu'il veut obtenir. C'est l'attente la plus immédiate à laquelle tout langage tente de répondre. Par exemple, un auteur veut faire tenir un contenu textuel donné dans une seule page. Cette simple tâche dépend de beaucoup de paramètres : principalement des dimensions de la page, des marges, de la taille des caractères, des fontes utilisées et de la taille des espaces entre les paragraphes. Quand cela est rendu possible par le langage, CSS2 par exemple, le fait d'encadrer le contenu par un élément de type *page* ne suffit pas. En effet, cela ne garantit pas que le contenu tiendra dans la page. Il risque d'y avoir débordement. Ce n'est bien sûr pas ce que veut l'auteur, et il n'a pas de moyen d'exprimer son but. La solution proposée de pouvoir gérer le débordement en ajoutant un ascenseur ou de laisser le contenu débordant visible, comme l'offre CSS2, n'est pas satisfaisante. Tant que son contenu tient dans une page, la spécification donnée suffit, mais dès que le contenu déborde, l'auteur est bloqué par cette spécification trop pauvre qui ne correspond pas à son besoin.

La seconde attente est celle de disposer de plus de souplesse dans les langages afin d'éviter au processus de formatage de tomber dans un cas d'échec quand il existe une solution qui pourrait l'éviter. Dans l'exemple précédent, il existe peut-être une solution au problème posé. L'auteur a d'ailleurs l'habitude de la rechercher en biaisant le formateur en changeant lui-même la taille des marges, en réduisant la taille des fontes ou encore en jouant sur l'importance des interlignes. En effet, parmi tous les paramètres dont dépend l'affichage du contenu pour tenir dans une page, certains sont plus souples que d'autres pour l'auteur. Il aurait été préférable que l'auteur puisse préciser qu'il voulait que certains de ces paramètres soient modifiables pour que le formateur fasse automatiquement cette recherche de solution à la place de l'auteur.

La troisième attente, qui est la plus difficile et qui découle des deux précédentes, est celle d'arriver à mieux contrôler le déroulement du formatage. Avec la deuxième attente, on comprend qu'il ne suffit pas d'imposer que le contenu doive rentrer dans la page. Le formateur tomberait tout de suite sur un cas d'échec, celui-ci ne pouvant décider à la place de l'auteur quels paramètres sont modifiables et dans quelle mesure. Mais une fois ces paramètres indiqués par l'auteur, rien ne garantit encore la qualité du résultat du formatage. Comment le formateur va-t-il gérer tous les degrés de liberté dont il dispose ? Comment va t'il choisir de réduire les interlignes plutôt que la taille des marges ? Une fois encore la notion d'échec est toujours présente. Une solution satisfaisante pour un auteur ne le sera pas pour un autre ou pour ce même auteur dans un contexte différent. L'auteur doit donc pouvoir également exprimer comment il souhaite que le formateur réalise sa recherche de solution quand celle-là a un impact sur le rendu de celle-ci.

Dans la suite de cette section nous essayons de voir quels éléments rajouter aux langages existants pour arriver à mieux exprimer les exigences des auteurs et les besoins de contrôle de la phase de formatage.

4.2. Les éléments du langage pour le contrôle du formatage

Nous avons identifié au moins trois éléments fondamentaux qui permettent d'ajouter à un langage de présentation des fonctions de contrôle de son formatage. Ces trois éléments, dépendants les uns des autres, sont *les niveaux de priorité*, *les propriétés abstraites et globales* et *les techniques de repli*. L'intérêt de notre approche est d'intégrer au maximum dans un seul langage ces différents éléments de présentation dans l'optique d'un meilleur contrôle du formatage par l'auteur.

4.2..1 Niveaux de priorité

C'est un moyen donné à l'auteur pour exprimer des préférences dans le comportement du formateur. L'auteur peut alors donner à des éléments, des attributs ou des relations différents niveaux d'importance. Cela lui permet d'indiquer au formateur l'ordre dans lequel effectuer les traitements. Un ordre de priorité donné par l'auteur peut lui permettre de guider le formateur quand celui-ci rencontre une alternative ou quand il doit faire un choix entre plusieurs techniques de repli. Sur un domaine de définition continu d'un attribut, l'ordre de priorité peut se résumer à une valeur préférentielle ou alors à un sens de parcours de ce domaine. Il est possible d'associer plusieurs ordres de priorité sur un même ensemble d'éléments par exemple si chaque ordre est attaché à une opération particulière (un ordre pour le temporel, un autre pour le spatial) ou une technique de repli. Il est enfin possible de déclarer des ordres de priorité ou d'importance plus complexes : aléatoires, dépendant d'une fonction, ...

Dans l'exemple décrit en figure 3, nous avons introduit un élément *priorites* qui regroupe les priorités affectées aux différentes parties de la présentation. Comme dans l'exemple, ces définitions peuvent être hiérarchiques.

4.2..2 Propriétés abstraites et propriétés globales

Elles permettent d'exprimer un placement spatial ou temporel, de prendre une décision par rapport à la sémantique du document au travers d'un ou plusieurs critères dont la valeur doit être optimisée. Par exemple, l'auteur peut vouloir exprimer qu'il ne veut pas plus de quatre éléments de type interview dans son document, qu'il veut le plus grand nombre possible de figure dans une page sans qu'elles ne se chevauchent, que la durée de son document soit exactement 15 minutes... Sans même rentrer dans le processus de formatage, on constate que cette notion entraîne facilement ce dernier vers des traitements incomplets et définit des problèmes NP-Difficiles. Toutefois, il est possible d'utiliser cette notion pour permettre un meilleur contrôle du processus de formatage sans pour autant le rendre trop complexe. Si l'optimisation d'un placement spatial en deux dimensions est difficile et ne peut pas être généralisé à l'heure actuelle en des temps raisonnables pour un formateur, l'optimisation d'un placement spatial dans une dimension, après avoir enlevé un degré de liberté en fixant une des deux dimensions, est parfaitement réalisable. La version la plus abordable de cette notion est donc par exemple le retaillage (temporel ou spatial) d'éléments dans une séquence en essayant de l'équilibrer, c'est-à-dire de rendre le retaillage de chaque élément de cette séquence proportionnel à l'importance de celui-ci.

```
<sequence id="rel1" repli="alt(rel2)" region="region1"/>
  <intervalle id="int1" ref="image1" durée="min:4s,pref:6s,max:10s"/>
    ...
  <intervalle id="int20" ref="image20" durée="min:4s,pref:6s,max:10s"/>
<\sequence>
<groupe id="rel2" type="alt"/>
  <parallèle id="rel3">
      ...
  <\parallèle>
<\groupe>
```

FIG. 1 – Exemple d'alternative sur une relation temporelle

4.2..3 Les techniques de repli

Les techniques de repli permettent de proposer des pistes aux formateurs quand ils sont en situation d'échec. Dans notre langage, un *repli* est une liste de spécifications de formatage qui portent sur une relation et ses éléments ou directement sur un élément d'une relation. Dans la suite nous parlons de repli associé à un noeud du document comme représenté sur la figure 4. Nous distinguons deux types de spécification dans un repli : l'**alternative** ou le **contrôle**. Le premier va modifier fortement la structure du noeud concerné alors que le second va plus simplement modifier la spécification de formatage associée à ce noeud. Nous précisons ces deux types de repli dans les paragraphes suivants.

Remarquons tout d'abord que la souplesse offerte par un langage de définition d'intervalles de valeurs entraîne une capacité intrinsèque de repli : les domaines de définition des variables constituent en effet des *alternatives implicites* pour les valeurs de ces variables. Les résolveurs de contraintes tirent naturellement parti de ces intervalles de définition mais cette souplesse est la plupart du temps trop vaste pour qu'ils sachent la gérer sans autre information. L'ajout de spécifications de repli, sous forme d'*alternatives explicites*, permet à l'auteur d'indiquer ses choix lorsque le formateur va devoir jouer avec la souplesse laissée par la spécification ou quand les limites de cette souplesse seront atteintes.

Les alternatives. Les alternatives telles que spécifiées dans les langages SMIL et XSL-FO permettent d'exprimer un choix pour décider de la valeur d'un attribut de présentation, parmi un ensemble fini ou non de valeurs discrètes ou continues, ou pour décider de la sélection des éléments ou des relations à formater. Nous étendons ces alternatives au cas du repli, c'est-à-dire l'utilisation d'une alternative non pas par rapport à la valeur d'un paramètre mais sur l'échec de la résolution. Quelques travaux se sont déjà intéressés à ce genre d'alternatives avec des approches un peu différentes. [RUT00] utilise des contraintes et s'intéresse aux alternatives pour la génération, l'effort d'adaptation est réalisé en amont du document et n'apparaît pas dans le langage de présentation. Dans [BOL99], des alternatives sont utilisées sur des médias et des fragments de médias dans un contexte d'adaptation, la meilleure des alternatives est choisie suivant le contexte de l'utilisateur lors du formatage final en

```
<groupe id="relg1" repli="flexible(prio1),flexible(),alt(prio1),supression(prio1)"/>
  ...
</groupe>
```

FIG. 2 – Exemple de solution de repli plus complexe

maintenant une cohérence sémantique du document. Dans notre approche, la cohérence sémantique est confiée à l'auteur. C'est lui qui va exprimer l'organisation et les solutions de repli. Les alternatives pouvant être nombreuses et porter sur des parties complexes de documents, il n'y a pas d'optimisation dans le choix de l'alternative. Dans l'exemple de la figure 1, une alternative, *alt(rel2)* est placée sur une séquence temporelle *rel1* de vingt images. En cas d'échec du formatage, cette séquence sera remplacée par le groupe temporel *rel2* composé par exemple de deux sequences présentant en parallèle ces images dans deux régions différentes. La groupe *rel2* est de type *alt*, il sera inclu et formaté dans le document uniquement si le formatage de *rel1* devait échouer. En conclusion, si les alternatives permettent de répondent à des besoins précis de spécification de repli, elles rendent les documents plus complexes à définir et maîtriser et plus difficiles à maintenir pendant leur durée de vie. En effet, elles imposent une redéfinition complète des noeuds à remplacer. Nous proposons donc un deuxième type de repli, plus simple dans son expression et plus limité sur son impact puisque la structure de présentation n'est pas remise en cause. Il peut être suffisant dans de nombreuses situations.

Les contrôles. Notre langage offre la possibilité d'ajouter à un noeud une suite de solutions de repli que nous appelons *les contrôles*. Un contrôle sert à modifier la spécification du formatage de la relation ou de l'élément sur laquelle elle s'applique. Dans la figure 2, nous voyons deux exemples de contrôle : *flexible* et *suppression*. Le contrôle *flexible* permet de spécifier que le formatage de la relation pourra rendre flexibles les éléments qui le permettent et le contrôle *suppression* autorise la suppression d'éléments pendant le formatage. D'autres contrôles peuvent être utilisés : la *reduction* ou l'*extension* qui n'autorisent que la réduction ou l'extension des éléments dans les limites des domaines de définition, le contrôle *fixe* permet de geler la valeur à celle de la durée ou des coordonnées courantes, enfin, *préférentielle* est un contrôle par défaut qui force les éléments à prendre la valeur préférentielle donnée par l'auteur. La figure 2 nous montre que les contrôles peuvent s'appuyer sur les priorités. Ainsi, le contrôle *alt(prio1)* spécifie que tous les éléments de la relation *relg1* qui sont définis dans la priorité *prio1* vont, en respectant l'ordre donné, être remplacés si nécessaire par leur alternative respective. Quand il n'y a pas de priorité, le contrôle s'applique à tous les éléments. Après avoir rendu flexibles uniquement les éléments définis par leur priorité *prio1*, tous les éléments contenus dans la relation *relg1* risquent de le devenir. La suite de la définition indique qu'en cas d'échec du formatage après absorption de la flexibilité de tous les éléments, le formateur tentera les alternatives des éléments, puis la suppression de ces éléments, selon la priorité *prio1*. Dans cet exemple, la spécification du repli est totalement indépendante de l'organisation temporelle du groupe *relg1*. Elle est construite de toute pièce et n'est donc pas figée dans un opérateur.

5. Notre expérimentation

5.1. Description de l'exemple

On veut formater une séquence temporelle qui contient le résumé en image d'une journée de championnat de football. Cette séquence temporelle se compose d'une suite de résumés de matchs. Le lecteur veut que cette séquence soit optimale, c'est-à-dire qu'elle comporte principalement ses équipes favorites, puis, s'il reste du temps, les résumés des matchs des autres équipes. La durée de la séquence doit respecter le temps donné par le lecteur. Un résumé de match se compose d'un ensemble de médias qui peuvent être présents ou non : vidéo des matchs et des interviews des joueurs, tableau des résultats, images des équipes, commentaires audio...

L'auteur du document a défini un scénario temporel pour présenter un résumé de match dans lequel il a associé des intervalles aux durées des médias. Il a également exprimé des degrés d'importance sur les médias qui composent le scénario : la vidéo du match est plus importante que l'interview des joueurs dans les vestiaires. Il fournit également des règles de repli pour le formatage de la séquence : si la séquence est trop longue, d'abord réduire les résumés des matchs les moins importants, puis réduire ceux des matchs favoris, puis si cela ne suffit pas, supprimer progressivement les résumés des matchs les moins importants.

La tâche du formateur est donc la suivante : calculer une valeur à chaque attribut temporel des éléments qui composent la séquence à partir des préférences du lecteur sur ses équipes favorites, de la durée de la présentation qu'il a donnée, de la souplesse laissée à la durée de chaque média, de l'importance des médias les uns par rapport aux autres et des règles de repli posées sur la séquence. La figure 3 présente la définition de cet exemple dont les commentaires numérotés sont :

1. Définition des médias (textes avec la description des équipes, photos des équipes, vidéos des matchs).
2. Définition des priorités : (po1) priorité définie à partir des priorités (po2) et (po3) qui établit un ordre entre toutes les séquences de résumé de match ; (po3) précise les préférences sur les matchs principaux. 4 matchs sont répartis entre 3 niveaux de priorité ; enfin (po2) donne un ordre relatif sur les matchs secondaires.
3. Définition de l'organisation temporelle.
4. Séquence temporelle principale qui relie les séquences de résumés de chaque match. Cette séquence comporte trois attributs particuliers. Le premier est celui de la durée, elle est donnée par le lecteur. Le deuxième est

```
<?xml version="1.0"
      encoding="ISO-8859-1"?>
<DocumentMultimedia>

<medias>                     <!-- 1 -->
  <texte id="mt1">Equipe 1</text>
  <texte id="mt2">Equipe 2</text>
  <image id="mi1" src="equipe1.jpg"/>
  <image id="mi2" src="equipe2.jpg"/>
  <video id="mv1" src="extrait.mpg"/>
  ...
</medias>

<priorites>                  <!-- 2 -->
  <ordre id="po1" ordre="croissant">
    <ordre id="po3">
      <niveau nb="1">
        <element ref="ResumeMatch1"/>
        <element ref="ResumeMatch2"/>
      </niveau>
      <niveau nb="2">
        <element ref="ResumeMatch3"/>
      </niveau>
      <niveau nb="3">
        <element ref="ResumeMatch4"/>
      </niveau>
    </ordre>
    <ordre id="po2" ordre="croissant">
      <element ref="ResumeMatch5"/>
      <element ref="ResumeMatch6"/>
      ...
      <element ref="ResumeMatch12"/>
    </ordre>
  </ordre>
  ...
</priorites>
```

```
<relations>
<temporal>                        <!-- 3 -->

<sequence id="rts1"               <!-- 4 -->
  durée="10min"
  equilibrage="proportionnel(po1)"
  repli="flexible(po2);
         flexible(po3);
         supression(po2)">

 <sequence id="ResumeMatch1"      <!-- 5 -->
         repli="flexible()">
    <parallele id="rtp1"
         durée="min:4s,pref:6s,max:10s">
      <intervalle id="rtsi11" ref="mt1"/>
      <intervalle id="rtsi12" ref="mi1"/>
    </parallele>
    <parallele id="rtp2"
         durée="min:4s,pref:6s,max:10s">
      <intervalle id="rtsi13" ref="mt2"/>
      <intervalle id="rtsi14" ref="mi21"/>
     </parallele>
     <intervalle id="rtsi15" ref="mv1"/>
 </sequence>

 <sequence id="ResumeMatch2">...</sequence>
 <sequence id="ResumeMatch3">...</sequence>
  ...
 <sequence id="ResumeMatch12">...</sequence>

</sequence>
</temporal>
<spatial> ... </spatial>
</relations>
</DocumentMultimedia>
```

FIG. 3 – Document multimédia intégrant des éléments de contrôle du formatage

"équilibrage". Il donne un critère global d'optimisation pour le formatage de cette séquence : chaque résumé est formaté suivant l'ordre de priorité (po1) donné et le résultat global doit être égal à la durée donné par l'auteur. Le troisième est "repli". Il indique quels sont les moyens de repli en cas d'échec du formatage.

5. Première des séquences de résumé de match. Cette séquence comprend un élément composé de deux éléments qui sont joués en parallèle (la description textuelle de la première équipe et une photo de cette équipe) puis d'un autre élément (idem pour l'autre équipe) puis de la vidéo résumé du match pour terminer cette séquence temporelle. Les deux éléments "parallèle" ont tous les deux des intervalles de valeur sur leur durée de présentation.

Les trois éléments présentés dans le langage apparaissent dans cet exemple. Il contient des alternatives avec les domaines de définition sur les intervalles en parallèle et dans les solutions de repli. Il utilise pleinement les définitions de priorités. Le placement des éléments dans la séquence principale respecte le critère de proportionnalité par rapport aux ordres de priorité. Et enfin, plusieurs solutions de repli sont proposées.

5.2. Réalisation

Nous avons effectué notre implémentation en langage Java autour du résolveur de contraintes hiérarchiques linéaires Cassowary [BAD98]. L'utilisation de replis implique l'introduction de disjonctions dans les traitements. Cette remarque nous a obligé en amont de notre réflexion à nous limiter certains aspects du langage. Par exemple, le mode de sélection de la solution de repli ne permet pas de choisir la meilleure solution de repli mais la première qui ne donne pas d'échec. Toutefois, l'utilisation de contraintes et de résolveurs qui gère l'incrémentalité, nous permet de limiter ce problème de disjonction, surtout dans le cas de repli sur un contrôle. Le coût d'un tel repli revient à ajouter ou à retirer quelques contraintes du résolveur en relançant une phase de résolution incrémentale au lieu de refaire entièrement une nouvelle résolution pour tout le document.

L'utilisation de contraintes hiérarchiques nous a permis de réaliser une première implémentation qui tire au maximum parti de leurs capacités. Sa réalisation est spécifique à la problématique de génération donnée dans la sous-section précédente. Nous avons obtenu un exemple de formatage d'une telle séquence temporelle qui est réalisé au mouvement de la souris près par manipulation directe de la durée totale de la séquence. Le côté spécifique de cet exemple vient du fait que pratiquement toutes les contraintes liées aux solutions de repli sont introduites dans le résolveur en leur associant des niveaux de priorités différents. Ainsi, profitant du mécanisme de relaxation des contraintes de moindre niveaux hiérarchiques, nous avons diminué le nombre d'appels au résolveur pour exécuter une nouvelle résolution.

Notre approche a toutefois l'objectif d'être générique de manière à être indépendante des organisations temporelles ou spatiales sur lesquelles elle s'applique. Nous avons basé une seconde implémentation sur le modèle décrit dans la figure 4, qui permet d'être générique dans le traitement des solutions de repli. Chaque noeud est traduit en un jeu de contraintes (1) puis le résolveur de contraintes résoud le système posé (2). Si une solution est trouvée, le document est formaté (3). En cas d'échec (4), une solution de repli est recherchée (5), puis appliquée sur le noeud en question (6). Un nouveau jeu de contraintes est alors créé en tenant compte des contraintes existantes et de la nouvelle spécification de repli (7). Notre système est donc constitué de trois modules :

1. Le module de traduction. Il s'occupe de la conversion de la spécification déclarative vers les contraintes numériques. Il gère également la cohérence des jeux de contraintes quand, à la suite d'un nouveau repli, il faut ajouter ou supprimer des contraintes.

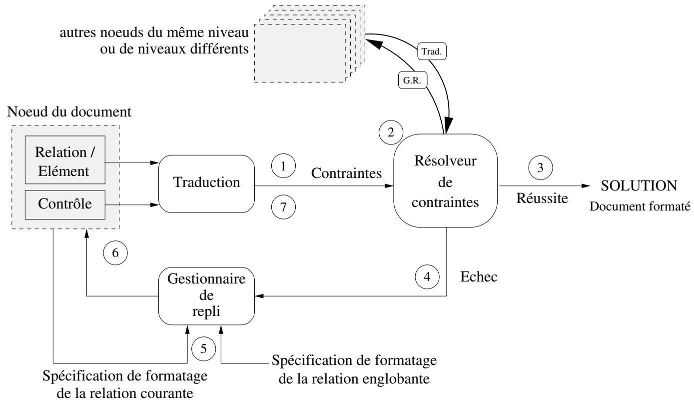

FIG. 4 – Formatage avec solution de repli

2. Le résolveur de contraintes. C'est le résolveur Cassowary. Il est unique à l'ensemble du formateur.
3. Le gestionnaire de repli. En fonction de la spécification présente sur le noeud courant, ce gestionnaire va rechercher la nouvelle solution de repli et la transmettre à ce même noeud. En cas d'alternative, il est chargé de remplacer l'ancien noeud par le ou les nouveaux noeuds. La hiérarchie influence le fonctionnement de ce module. En effet, principalement lorsque le noeud courant est un élément, ce sont les informations contenues sur le noeud père, ou héritées par celui-ci, qui vont permettre de choisir la spécification à adopter.

Pour évaluer les performances de notre implémentation nous avons pris plusieurs séquences équivalentes à la séquence *rts1* de la figure 3 au sein d'un même document. Le but était de tester le comportement de plusieurs replis complexes en même temps lors d'un formatage. Nous avons comparé les performances en faisant varier le nombre de séquences dans le document, le nombre d'éléments par séquences et le nombre de replis appliqués à chaque séquence (en changeant la durée souhaitée pour cette séquence). Lorsque les trois replis successifs donnés par la séquence *rts1* sont appliqués sur toutes les séquences du document, celui-ci se formate en 878 ms pour 1 séquence de 30 éléments, en 1078 ms pour 5 séquences de 10 éléments et 1758 ms pour 10 séquences de 10 éléments alors que deux replis de contrôle de réduction donnent pour les mêmes documents : 671 ms, 703 ms et 1049 ms. Les résultats de ces tests sont conformes à nos attentes pour ce qui est de la spécification du contrôle et du déroulement des replis. Au niveau des temps d'exécution, un travail d'optimisation est encore nécessaire pour obtenir des performances plus satisfaisantes. On constate notamment que le coût d'un éventuel repli sur chaque intervalle alourdit le processus. En effet, pour un même nombre d'éléments, les performances sont meilleures pour un document comportant le plus de séquences (et doncle plus de replis effectués) : les performances de 5 séquences de 40 éléments sont moins bonnes que celles de 10 séquences de 20 éléments. Nos tests ont également montré que le coût d'un repli par contrôle de suppression était plus important qu'un repli par contrôle de réduction.

6. Conclusion

Ce papier a présenté notre approche pour introduire des éléments de contrôle du formatage dans les langages de présentation multimédia. Nous avons vu les limitations des langages de présentation actuels et des systèmes basés sur l'utilisation de contraintes par rapport à cette préoccupation de contrôle. Notre réponse à ces limitations est d'offrir une capacité d'expression de haut niveau par l'utilisation de trois concepts principaux dans notre langage : les priorités, les propriétés globales et abstraites et les techniques de repli. Ce niveau d'expression est obtenu d'une part par la définition de priorités sur les noeuds du document et d'autre part par l'ajout, aux relations classiques de présentation, de critères globaux d'optimisation et de replis. Le langage présenté ici est un premier travail destiné à valider l'utilisation de ces trois concepts. Il est possible de l'enrichir en ajoutant de nouvelles solutions de repli et d'autres propriétés globales. Nos préoccupations d'amélioration concernent cependant en priorité le traitement du langage présenté ici pour arriver à obtenir un formateur plus performant et surtout une plus grande maîtrise des composants de la résolution.

De part son niveau d'abstraction intéressant et sa capacité à exprimer des solutions de repli, le langage que nous avons défini paraît répondre à certaines attentes exprimées dans le domaine de la génération et de l'adaptation de documents multimédia. C'est notamment le cas des architectures à base de transformation [VIL00], [BES01] où le formatage doit prendre en charge une part importante de l'adaptation qui n'a pas pu être résolue plus tôt. Une suite à ce travail est donc d'étudier plus en détail l'apport d'un tel langage dans ce domaine.

Nous envisageons enfin comme perspectives d'utiliser notre langage pour apporter aux langages de présentation existants des capacités de contrôle du formatage. Notre ambition est double : intégration du langage et intégration du traitement. Nous voulons non seulement être capable d'intégrer notre langage dans les langages existants mais également de proposer des services de formatage visant à intégrer son traitement dans les formateurs existants.

Références

[AND00] André E., "The Generation of Multimedia Documents", *R. Dale, H. Moisl and H. Somers : A Handbook of Natural Language Processing : Tech-*

niques and Applications for the Processing of Language as Text, Marcel Dekker Inc., pp. 305-327, 2000.

[BAD98] Badros G. J. et Borning A., *"The Cassowary Linear Arithmetic Constraint Solving Algorithm : Interface and Implementation"*, Technical Report UW- CSE-98-06-04, 1998.

[BAD99] Badros G. J., Borning A., Marriott K. et Stuckey P., "Constraint cascading style sheets for the web", *Proceedings of the 1999 ACM Conference on User Interface Software and Technology*, novembre 1999.

[BAD01] Badros G. J., Tirtowidjojo J. J., Marriott K., Meyer B., Portnoy W. et Borning A., "A Constraint Extension to Scalable Vector Graphics", *The Tenth International World Wide Web Conference*, pp. 489-498, Hong Kong, 1-5 mai 2001.

[BES01] Bes F., Jourdan M., Khantache F., "A Generic Architecture for Automated Construction of Multimedia Presentations", *Proceedings of 8th International Conference Multimedia Modeling (MMM01)*, Amsterdam, 5-7 novembre 2001.

[BOL99] Boll S., Klas W. et Wandel J., "A Cross-Media Adaptation Strategy for Multimedia Presentations", *Proceedings of ACM Multimedia '99*, pp. 37-46, Orlando, Floride, 30 octobre - 5 novembre 1999.

[BOL01] Boll S. et Klas W., "ZYX-A Multimedia Document Model for Reuse and Adaptation of Multimedia Content", *Transations on Knowledge and Dare Engineering (Vol.13,N°3)*, IEEE, pp.361-382 , mai-juin 2001.

[KIM95] Kim M. Y. et Song J., "Multimedia Documents with Elastic Time", *Proceedings of the Third ACM International Conference on Multimedia*, édité par ACM Press, pp. 143-154, Polle Zellweger, San Fransisco, Californie, 5-9 novembre 1995.

[LAM86] Lamport L., LATEX : a document preparation system, *Addison-Wesley*, New-York, 1986.

[LAY97] Layaïda N., Madeus : Système d'édition et de presentation de documents structures multimedia, Thèse de doctorat, Université Joseph Fourier, 12 juin 1997.

[RUT00] Rutledge L., Bailey B., van Ossenbruggen J., Hardman L. et Geurts J., "Generating Presentation Constraints from Rhetorical Structure", *Proceedings of the 11th ACM conference on Hypertext and Hypermedia*, pp. 19-28, San Antonio, Texas, USA, 30 mai - 3 juin 2000.

[VAN01] van Ossenbruggen J., Geurts J., Cornelissen F., Rutledge L. et Hardman L., "Towards Second and Third Generation Web-Based Multimedia", *The Tenth International World Wide Web Conference*, pp. 479-488, Hong Kong, 1-5 mai 2001.

[VIL00] Villard L., Roisin C. et Layaïda N., "An XML-based multimedia document processing model for content adaptation", *Proceedings of Eighth International Conference on Digital Documents and Electronic Publishing*, 14 septembre 2000.